5

Workflow Scheduling Algorithms for Grid Computing

Jia Yu, Rajkumar Buyya and Kotagiri Ramamohanarao

Grid Computing and Distributed Systems (GRIDS) Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, VIC 3010 Austraila
{jiayu,raj,rao}@csse.unimelb.edu.au

Summary. Workflow scheduling is one of the key issues in the management of workflow execution. Scheduling is a process that maps and manages execution of inter-dependent 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 significant impact on the performance of the system. In this chapter, we investigate existing workflow scheduling algorithms developed and deployed by various Grid projects.

Key words: Workflow scheduling, Inter-dependent tasks, Distributed resources, Heuristics.

5.1 Introduction

Grids [22] have emerged as a global cyber-infrastructure for the next-generation of e-Science and e-business applications, by integrating large-scale, distributed and heterogeneous resources. A number of Grid middleware and management tools such as Globus [21], UNICORE [1], Legion [27] and Gridbus [13] have been developed, in order to provide infrastructure that enables users to access remote resources transparently over a secure, shared scalable world-wide network. More recently, Grid computing has progressed towards a service-oriented paradigm [7, 24] which defines a new way of service provisioning based on utility computing models. Within utility Grids, each resource is represented as a service to which consumers can negotiate their usage and Quality of Service.

Scientific communities in areas such as high-energy physics, gravitational-wave physics, geophysics, astronomy and bioinformatics, are utilizing Grids to share, manage and process large data sets. In order to support complex scientific experiments, distributed resources such as computational devices, data, applications, and scientific instruments need to be orchestrated while

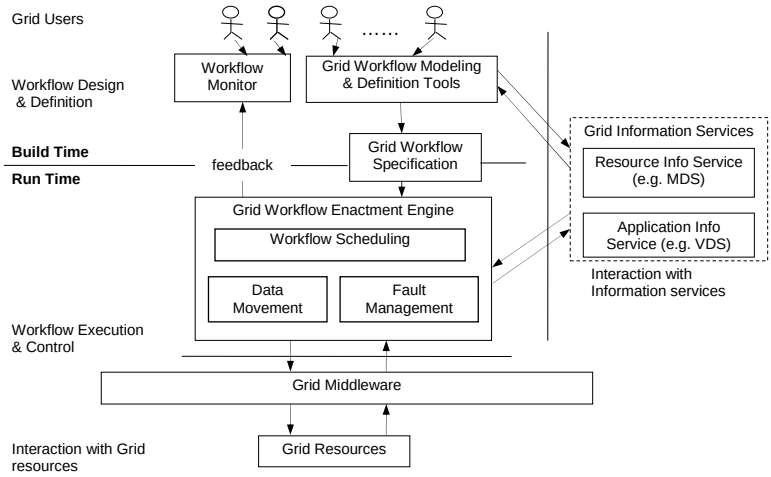


Fig. 5.1. Grid Workflow Management System.

managing the application workflow operations within Grid environments [36]. Workflow is concerned with the automation of procedures, whereby files and other data are passed between participants according to a defined set of rules in order to achieve an overall goal [30]. A workflow management system defines, manages and executes workflows on computing resources.

Fig. 5.1 shows an architecture of workflow management systems for Grid computing. In general, a workflow specification is created by a user using workflow modeling tools, or generated automatically with the aid of Grid information services such as MDS(Monitoring and Discovery Services) [20] and VDS (Virtual Data System) [23] prior to the run time. A workflow specification defines workflow activities (tasks) and their control and data dependencies. At run time, a workflow enactment engine manages the execution of the workflow by utilizing Grid middleware. There are three major components in a workflow enactment engine: the workflow scheduling, data movement and fault management. Workflow scheduling discovers resources and allocates tasks on suitable resources to meet users' requirements, while data movement manages data transfer between selected resources and fault management provides mechanisms for failure handling during execution. In addition, the enactment engine provides feedback to a monitor so that users can view the workflow process status through a Grid workflow monitor. Workflow scheduling is one of the key issues in the workflow management [59].

A scheduling is a process that maps and manages the execution of interdependent tasks on the distributed resources. It allocates suitable resources to workflow tasks so that the execution can be completed to satisfy objective functions imposed by users. Proper scheduling can have significant impact on the performance of the system. In general, the problem of mapping tasks on

distributed services belongs to a class of problems known as NP-hard problems [53]. For such problems, no known algorithms are able to generate the optimal solution within polynomial time. Solutions based on exhaustive search are impractical as the overhead of generating schedules is very high. In Grid environments, scheduling decisions must be made in the shortest time possible, because there are many users competing for resources, and time slots desired by one user could be taken up by another user at any moment.

Many heuristics and meta-heuristics based algorithms have been proposed to schedule workflow applications in heterogeneous distributed system environments. In this chapter, we discuss several existing workflow scheduling algorithms developed and deployed in various Grid environments.

5.2 Workflow Scheduling Algorithms for Grid Computing

Many heuristics [33] have been developed to schedule inter-dependent tasks in homogenous and dedicated cluster environments. However, there are new challenges for scheduling workflow applications in a Grid environment, such as:

- Resources are shared on Grids and many users compete for resources.
- Resources are not under the control of the scheduler.
- Resources are heterogeneous and may not all perform identically for any given task.
- Many workflow applications are data-intensive and large data sets are required to be transferred between multiple sites.

Therefore, Grid workflow scheduling is required to consider non-dedicated and heterogeneous execution environments. It also needs to address the issue of large data transmission across various data communication links.

The input of workflow scheduling algorithms is normally an *abstract workflow model* which defines workflow tasks without specifying the physical location of resources on which the tasks are executed. There are two types of abstract workflow model, *deterministic* and *non-deterministic*. In a deterministic model, the dependencies of tasks and I/O data are known in advance, whereas in a non-deterministic model, they are only known at run time.

The workflow scheduling algorithms presented in the following sections are based on the deterministic type of the abstract workflow model and are represented 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), corresponding to the data communication between task T_i and T_j, where T_i is called an immediate parent task of T_j, and T_j the immediate child task of T_i. We assume that a child task cannot be executed until all of its parent tasks are completed. Then, the workflow application can be described as a tuple $\Omega(\Gamma, \Lambda)$.

In a workflow graph, a task which does not have any parent task is called an *entry task*, denoted as T_{entry} and a task which does not have any child task is called an *exit task*, denoted as T_{exit}. If a workflow scheduling algorithm requires a single entry task or a single exit task, and a given workflow contains more than one entry task or exit task in the workflow graph, we can produce a new workflow by connecting entry points to a zero-cost pseudo entry and exiting nodes to an exit task, without without affecting the schedule [45].

To date, there are two major types of workflow scheduling (see Fig. 5.2), *best-effort based* and *QoS constraint based* scheduling. The best-effort based scheduling attempts to minimize the execution time ignoring other factors such as the monetary cost of accessing resources and various users' QoS satisfaction levels. On the other hand, QoS constraint based scheduling attempts to minimize performance under most important QoS constraints, for example time minimization under budget constraints or cost minimization under deadline constraints.

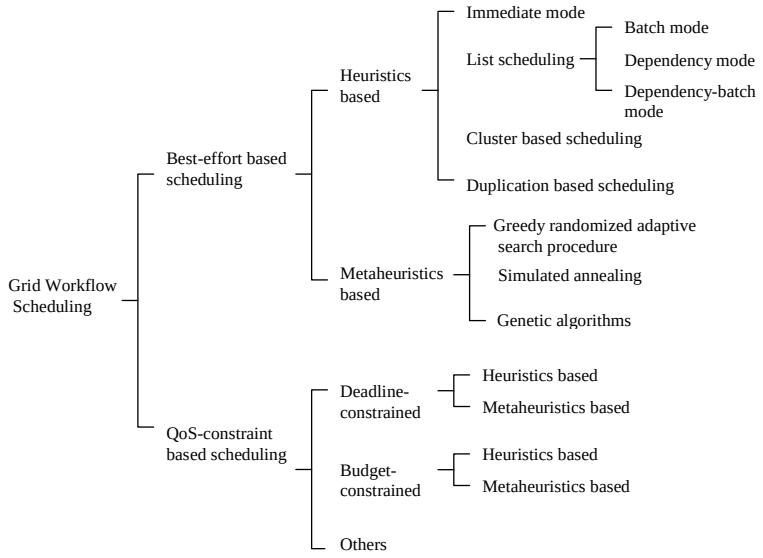


Fig. 5.2. A taxonomy of Grid workflow scheduling algorithms.

5.3 Best-effort based workflow scheduling

Best-effort based workflow scheduling algorithms are targeted towards Grids in which resources are shared by different organizations, based on a community

model (known as *community Grid*). In the community model based resource allocation, monetary cost is not considered during resource access. Best-effort based workflow scheduling algorithms attempt to complete execution at the earliest time, or to minimize the makespan of the workflow application. The makespan of an application is the time taken from the start of the application, up until all outputs are available to the user [14].

In general, best-effort based scheduling algorithms are derived from either *heuristics based* or *meta-heuristics based* approach. The heuristic based approach is to develop a scheduling algorithm which fit only a particular type of problem, while the meta-heuristic based approach is to develop an algorithm based on a meta-heuristic method which provides a general solution method for developing a specific heuristic to fit a particular kind of problem [29].Table 5.1 5.2 show the overview of best-effort based scheduling algorithms.

Table 5.1. Overview of Best-effort Workflow Scheduling Algorithms (Heuristics).

Scheduling Method		**Algorithm**	**Project**	**Organization**	**Application**
Individual task scheduling		Myopic	Condor DAG Man	University of Wisconsin-Madison, USA.	N/A
List scheduling	Batch mode	Min-Min	vGrADS	Rice University, USA.	EMAN bio-imaging
			Pegasus	University of Southern California, USA.	Montage astronomy
		Max-min	vGrADS	Rice University, USA.	EMAN bio-imaging
		Sufferage	vGrADS	Rice University, USA.	EMAN bio-imaging
	Dependency mode	HEFT	ASKALON	University of Innsbruck, Austria.	WIEN2K quantum chemistry & Invmod hydrological
	Dependency-batch mode	Hybrid	Sakellarious & Zhao	University of Manchester, UK.	Randomly generated task graphs
Cluster based scheduling Duplication based scheduling		THAN	Ranaweera & Agrawal	University of Cincinnati, USA	Randomly generated task graphs

Table 5.2. Overview of Best-effort Workflow Scheduling Algorithms (Meta-heuristics).

Scheduling Method	Project	Organization	Application
Greedy randomized adaptive search procedure (GRASP)	Pegasus	University of Southern California, USA.	Montage astronomy
Genetic algorithms (GA)	ASKALON	University of Innsbruck, Austria.	WIEN2K quantum chemistry
Simulated annealing (SA)	ICENI	London e-Science Centre, UK.	Randomly generated task graphs

5.3.1 Heuristics

In general, there are four classes of scheduling heuristics for workflow applications, namely *individual task scheduling*, *list scheduling*, and *cluster and duplication based scheduling*.

Individual task scheduling

The individual task scheduling is the simplest scheduling method for scheduling workflow applications and it makes schedule decision based only on one individual task. The *Myopic* algorithm [55] has been implemented in some Grid systems such as Condor DAGMan [49]. The detail of the algorithm is shown in Algorithm 4. The algorithm schedules an unmapped ready task to the resource that is expected to complete the task earliest, until all tasks have been scheduled.

Algorithm 4 Myopic scheduling algorithm.

1: **while** $\exists t \in \Gamma$ is not completed **do**
2: $task \leftarrow$ get a ready task whose parent tasks have been completed
3: $r \leftarrow$ for $t \in task$, get a resource which can complete t at the earliest time
4: schedule t on r
5: **end while**

List scheduling

A list scheduling heuristic prioritizes workflow tasks and scheldules the tasks based on their priorities. There are two major phases in a list scheduling heuristic, the *task prioritizing* phase and the *resource selection* phase [33]. The task prioritizing phase sets the priority of each task with a rank value and generates a scheduling list by sorting the tasks according to their rank values. The resource selection phase selects tasks in the order of their priorities and map each selected task on its optimal resource.

Different list scheduling heuristics use different attributes and strategies to decide the task priorities and the optimal resource for each task. We categorize workflow-based list scheduling algorithms as either *batch*, *dependency* or *dependency-batch* mode.

The batch mode scheduling group workflow tasks into several independent tasks and consider tasks only in the current group. The dependency mode ranks workflow tasks based on its weight value and the rank value of its inter-dependent tasks, while the dependency-batch mode further use a batch mode algorithm to re-ranks the independent tasks with similar rank values.

Batch mode

Batch mode scheduling algorithms are initially designed for scheduling parallel independent tasks, such as bag of tasks and parameter tasks, on a pool of resources. Since the number of resources is much less than the number of tasks, the tasks need to be scheduled on the resources in a certain order. A batch mode algorithm intends to provide a strategy to order and map these parallel tasks on the resources, in order to complete the execution of these parallel tasks at earliest time. Even though batch mode scheduling algorithms aim at the scheduling problem of independent tasks; they can also be applied to optimize the execution time of a workflow application which consists of a lot of independent parallel tasks with a limited number of resources.

Batch Mode Algorithms. *Min-Min*, *Max-Min*, *Sufferage* proposed by Maheswaran et al. [39] are three major heuristics which have been employed for scheduling workflow tasks in vGrADS [4] vGrADS [4] and pegasus [11]. The heuristics is based on the performance estimation for task execution and I/O data transmission. The definition of each performance metric is given in Table 5.3.

Table 5.3. Performance Matrices.

Symbol	Definition
$EET(t,r)$	**Estimated Execution Time:** the amount of time the resource r will take to execute the task t, from the time the task starts to execute on the resource.
$EAT(t,r)$	**Estimated Availability Time:** the time at which the resource r is available to execute task t.
$FAT(t,r)$	**File Available Time:** the earliest time by which all the files required by the task t will be available at the resource r.
$ECT(t,r)$	**Estimated Completion Time:** the estimated time by which task t will complete execution at resource r: $ECT(t,r) = EET(t,r) + \max(EAT(t,r), FAT(t,r))$
$MCT(t)$	**Minimum Estimated Completion Time:** minimum ECT for task t over all available resources.

The Min-Min heuristic schedules sets of independent tasks iteratively (Algorithm 5: 1-4). For each iterative step, it computes ECTs(Early Completion Time) of each task on its every available resource and obtains the MCT(Minimum Estimated Completion Time) for each task (Algorithm 5: 7-12). A task having minimum MCT value over all tasks is chosen to be scheduled first at this iteration. It assigns the task on the resource which is expected to complete it at earliest time.

Algorithm 5 Min-Min and Max-Min task scheduling algorithms.

1: **while** $\exists t \in \Gamma$ is not scheduled **do**
2: $availTasks \leftarrow$ get a set of unscheduled ready tasks whose parent tasks have been completed
3: schedule($availTasks$)
4: **end while**
5: **PROCEDURE:** schedule($availTasks$)
6: **while** $\exists t \in availTasks$ is not scheduled **do**
7: **for all** $t \in availTasks$ **do**
8: $availResources \leftarrow$ get available resources for t
9: **for all** $r \in availResources$ **do**
10: compute $ECT(t, r)$
11: **end for**
12: // get $MCT(t, r)$ for each resource
$R_t \leftarrow \min_{r \in availResources} ECT(t, r)$
13: **end for**
14: // Min-Min: get a task with minimum $ECT(t, r)$ over tasks
$T \leftarrow \arg \min_{t \in availTasks} ECT(t, R_t)$
// Max-Min: get a task with maximum $ECT(t, r)$ over tasks
$T \leftarrow \arg \min_{t \in availTasks} ECT(t, R_t)$
15: schedule T on R_T
16: remove T from $availTasks$
17: update $EAT(R_T)$
18: **end while**

The Max-Min heuristic is similar to the Min-Min heuristic. The only difference is the Max-Min heuristic sets the priority to the task that requires longest execution time rather than shortest execution time. After obtaining MCT values for each task (Algorithm 5: 7-13), a task having maximum MCT is chosen to be scheduled on the resource which is expected to complete the task at earliest time. Instead of using minimum MCT and maximum MCT, the Sufferage heuristic sets priority to tasks based on their sufferage value. The sufferage value of a task is the difference between its earliest completion time and its second earliest completion time (Algorithm 6: 12-14).

Comparison of batch mode algorithms. The overview of three batch mode algorithms are shown in Table 5.4. The Min-Min heuristic schedules

tasks having shortest execution time first so that it results in the higher percentage of tasks assigned to their best choice (which can complete the tasks at earlist time) than Max-Min heuristics [12]. Experimental results conducted by Maheswaran et al. [39] and Casanova et al. [14] have proved that Min-Min heuristic outperform Max-Min heuristic. However, since Max-min schedule tasks with longest execution time first, a long execution execution task may have more chance of being executed in parallel with shorter tasks. Therefore, it might be expected that the Max-Min heuristic perform better than the Min-Min heuristic in the cases where there are many more short tasks than long tasks [12, 39].

Algorithm 6 Sufferage task scheduling algorithm.

1: **while** $\exists t \in \Gamma$ is not completed **do**
2: $availTasks \leftarrow$ get a set of unscheduled ready tasks whose parent tasks have been completed
3: schedule($availTasks$)
4: **end while**
5: **PROCEDURE:** schedule($availTasks$)
6: **while** $\exists t \in availTasks$ is not scheduled **do**
7: **for all** $t \in availTasks$ **do**
8: $availResources \leftarrow$ get available resources for t
9: **for all** $r \in availResources$ **do**
10: compute $ECT(t, r)$
11: **end for**
12: // compute earliest ECT
$R_t^1 \leftarrow arg \min_{r \in availResources} ECT(t, r)$
13: // compute second earliest ECT
$R_t^2 \leftarrow arg \min_{r \in availResources \& r \neq R_t^1} ECT(t, r)$
14: // compute sufferage value for task t
$suf_t \leftarrow ECT(t, R_t^2) - ECT(t, R_t^1)$
15: **end for**
16: $T \leftarrow \arg \max_{t \in availTasks} suf_t$
17: schedule T on R_T^1
18: remove T from $availTasks$
19: update $EAT(R_T)$
20: **end while**

On the other hand, since the Sufferage heuristic consider the adverse effect in the completion time of a task if it is not scheduled on the resource having with minimum completion time [39], it is expected to perform better in the cases where large performance difference between resources. The experimental results conducted by Maheswaran et al. shows that the Sufferage heuristic produced the shortest makespan in the high heterogeneity environment among three heuristics discussion in this this section. However, Casanova et al. [14]

argue argue that the Sufferage heuristic could perform worst in the case of data-intensive applications in multiple cluster environments.

Table 5.4. Overview of batch mode algorithms.

Algorithm	Features
$Min - Min$	It sets high scheduling priority to tasks which have the shortest execution time.
$Max - Min$	It sets high scheduling priority to tasks which have long execution time.
$Sufferage$	It sets high scheduling priority to tasks whose completion time by the second best resource is far from that of the best resource which can complete the task at earliest time.

Extended batch mode algorithms. XSufferage is an extension of the Suffereage heuristic. It computes the sufferage value on a cluster level with the hope that the files presented in a cluster can be maximally reused. A modified Min-Min heuristic, *QoS guided Min-Min*, is also proposed in [28]. In addition to comparing the minimum completion time over tasks, it takes into account different levels of quality of service (QoS) required by the tasks and provided by Grid resources such as desirable bandwidth, memory and CPU speed. In general, a task requiring low levels of QoS can be executed either on resources with low QoS or resources with high QoS, whereas the task requiring high levels of QoS can be processed only on resources with high QoS. Scheduling tasks without considering QoS requirements of tasks may lead to poor performance, since low QoS tasks may have higher priority on high QoS resources than high QoS tasks, while resources with low QoS remain idle [28]. The QoS guided Min-Min heuristic starts to map low QoS tasks until all high QoS tasks have been mapped. The priorities of tasks with the same QoS level are set in the same way of the Min-Min heuristic.

Dependency Mode

Dependency mode scheduling algorithms are derived from the algorithms for scheduling a task graph with interdependent tasks on distributed computing environments. It intends to provide a strategy to order and map workflow tasks on heterogeneous resources based on analyzing the dependencies of the entire task graph, in order to complete these interdependent tasks at earliest time. Unlike batch mode algorithms, it ranks the priorities of all tasks in a workflow application at one time.

Many dependency mode heuristics rank tasks are based on the weights of task nodes and edges in a task graph. As illustrated in Fig. 5.3, a weight w_i is assigned to a task T_i and a weight $w_{i,j}$ is assigned to an edge (T_i, T_j). Many list scheduling schemes [33] developed for scheduling task graphs on homogenous systems set the weight of each task and edge to be equal to its

estimation execution time and communication time, since in a homogenous environment, the execution times of a task and data transmission time on all available resources are identical. However, in a Grid environment, resources are heterogeneous. The computation time varies from resource to resource and the communication time varies from data link to data link between resources. Therefore, it needs to consider processing speeds of different resources and different transmission speeds of different data links and an approximation approach to weight tasks and edges for computing the rank value.

Zhao and Sakellariou [62] proposed six possible approximation options, *mean value*, *median value*, *worst value*, *best value*, *simple worst value*, and *simple best value*. These approximation approaches assign a weight to each task node and edge as either the average, median, maximum, or minimum computation time and communication time of processing the task over all possible resources. Instead of using approximation values of execution time and transmission time, Shi time, Shi and Dongarra [46] assign a higher weight task with less capable resources. Their motivation is quite similar to the QoS guided min-min scheduling, i.e., it may cause longer delay if tasks with scarce capable resources are not scheduled first, because there are less choices of resources to process these tasks.

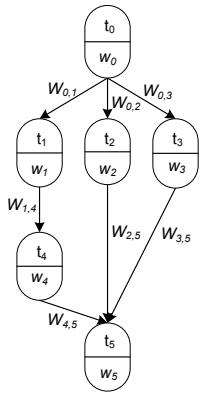


Fig. 5.3. a weighted task graph example.

Dependency Mode Algorithm. The *Heterogeneous-Earliest-Finish-Time* (HEFT) algorithm proposed by Topcuoglu et al. [51] has been applied by the ASKALON project [18, 55] to provide scheduling for a quantum chemistry application, WIEN2K [10], and a a quantum chemistry application, WIEN2K [10], and a hydrological application, Invmod [43], on the

As shown in Algorithm 7, the algorithm first calculates average execution time for each task and average communication time between resources of two successive tasks. Let $time(T_i, r)$ be the execution time of task T_i on resource

r and let R_i be the set of all available resources for processing T_i. The average execution time of a task T_i is defined as

$$\varpi_i = \frac{\sum_{r \in R_i} time(T_i, r)}{|R_i|} \tag{5.1}$$

Let $time(e_{ij}, r_i, r_j)$ be the data transfer time between resources r_i and r_j which process the task Ti and task T_j respectively. Let R_i and R_j be the set of all available resources for processing Ti and T_j respectively. The average transmission time from T_i to T_j is defined by:

$$\overline{c_{ij}} = \frac{\sum_{r_i \epsilon R_i r_j \epsilon R_j} time(e_{ij}, r_i, r_j)}{|R_i||R_j|} \tag{5.2}$$

Then tasks in the workflow are ordered in HEFT based on a rank fuction. For a exit task T_i, the rank value is:

$$rank(T_i) = \varpi_i \tag{5.3}$$

The rank values of other tasks are computed recursively based on 5.15.25.3 as shown in 5.4.

$$rank(T_i) = \varpi_i + \max_{T_j \epsilon succ(T_i)} (\overline{c_{ij}} + rank(T_j)) \tag{5.4}$$

where $succ(T_i)$ is the set of immediate successors of task T_i. The algorithm then sorts the tasks by decreasing order of their rank values. The task with higher rank value is given higher priority. In the resource selection phase, tasks are scheduled in the order of their priorities and each task is assigned to the resource that can complete the task at the earliest time.

Algorithm 7 Heterogeneous-Earliest-Finish-Time (HEFT) algorithm.

1: compute the average execution time for each task $t \in \Gamma$ according to equation 5.1
2: compute the average data transfer time between tasks and their successors according to equation 5.2
3: compute rank value for each task according to equations 5.3 and 5.4
4: sort the tasks in a scheduling list Q by decreasing order of task rank value
5: **while** Q is not empty **do**
6: $t \leftarrow$ remove the first task from Q
7: $r \leftarrow$ find a resource which can complete t at earliest time
8: schedule t to r
9: **end while**

Even though original HEFT proposed by Topcuoglu et al. [51] computes the rank value for each task using the mean value of the task execution time and communication time over all resources, Zhao time over all resources, Zhao and Sakellariou [62] performances of the HEFT algorithm produced by other different approximation methods on different cases. The results of the expeirments showed that the mean value method is not the most effiecient choice, and the performance could differ significantly from one application to another [62].

Dependency-Batch Mode

Sakellariou and Zhao [45] proposed a hybrid heuristic for scheduling DAG on heterogeneous systems. The heuristic combines dependency mode and batch mode. As described in Algorithm 8, the heuristic first compute rank values of each task and ranks all tasks in the decreasing order of their rank values (Algorithm 8:line 1-3). And then it creates groups of independent tasks (Algorithm 8:line 4-11). In the grouping phase, it processes tasks in the order of their rank values and add tasks into the current group. Once it finds a task which has a dependency with any task within the group, it creates another new group. As a result, a number of groups of independent tasks are generated. And the group number is assigned based on the order of rank values of their tasks, i.e., if $m > n$, the ranking value of tasks in group m is higher than that of the tasks in group n. Then it schedules tasks group by group and uses a batch mode algorithm to reprioritize the tasks in the group.

Algorithm 8 Hybrid heuristic.

1: compute the weight of each task node and edge according to equations 5.1 and 5.2
2: compute the rank value of each task according to equations 5.3 and 5.4
3: sort the tasks in a scheduling list Q by decreasing order of task rank value
4: create a new group G_i and $i = 0$
5: **while** Q is not empty **do**
6: $t \leftarrow$ remove the first task from Q
7: **if** t has a dependence with a task in G_i **then**
8: $i++$; create a new group G_i
9: **end if**
10: add t to G_i
11: **end while**
12: $j = 0$
13: **while** $j <= i$ **do**
14: scheduling tasks in G_i by using a batch mode algorithm
15: $j++$
16: **end while**

Cluster based and Duplication based scheduling

Both cluster based scheduling and duplication based scheduling are designed to avoid the transmission time of results between data interdependent tasks, such that it is able to reduce the overall execution time. The cluster based scheduling clusters tasks and assign tasks in the same cluster into the same resource, while the duplication based scheduling use the idling time of a resource to duplicate some parent tasks, which are also being scheduled on other resources.

Bajai and Agrawal [3] proposed a *task duplication based scheduling algorithm for network of heterogeneous systems*(TANH) . The algorithm combine cluster based scheduling and duplication based scheduling and the overview of the algorithm is shown in Algorithm 9. It first traverses the task graph to compute parameters of each node including earliest start and completion time, latest start and completion time, critical immediate parent task, best resource and the level of the task. After that it clusters tasks based on these parameters. The tasks in a same cluster are supposed to be scheduled on a same resource. If the number of the cluster is greater than the number of resources, it scales down the number of clusters to the number of resources by merging some clusters. Otherwise, it utilizes the idle times of resources to duplicate tasks and rearrange tasks in order to decrease the overall execution time.

Algorithm 9 TANH algorithm.

```
1: compute parameters for each task node
2: cluster workflow tasks
3: if the number of clusters greater than the number of available resources then
4:     reducing the number of clusters to the number of available resources
5: else
6:     perform duplication of tasks
7: end if
```

5.3.2 Meta-heuristics

Meta-heuristics provide both a general structure and strategy guidelines for devoping a heuristic for solving computational problems. They are generally applied to a large and complicated problem. They provide an efficient way of moving quickly toward a very good solution. Many metahuristics have been applied for solving workflow scheduling problmes, including GRASP, Genetic Algorithms and Simulated Annealing. The details of these algorithms are presented in the sub-sections that follow.

Greedy Randomized Adaptive Search Procedure (GRASP)

A *Greedy Randomized Adaptive Search Procedure* (GRASP) is an iterative randomized search technique. Feo and Resende [19] proposed guidelines for developing heuristics to solve combinatorial optimization problems based on the GRASP concept. Binato et al. [8] have shown that the GRASP can solve job-shop scheduling problems effectively. Recently, the GRASP has been investigated by Blythe et al. [11] for workflow scheduling on Grids by comparing with the Min-Min heuristic on both computational- and data-intensive applicaitons.

Algorithm 10 GRASP algorithm.

```
 1: while stopping criterion not satisfied do
 2:    schedule ← createSchedule(workflow)
 3:    if schedule is better than bestSchedule then
 4:       bestSchedule ← schedule
 5:    end if
 6: end while
 7: PROCEDURE: createSchedule(workflow)
 8: solution ← constructSolution(workflow)
 9: nSolution ← localSearch(solution)
10: if nSolution is better than solution then
11:    return nSolution
12: end if
13: return solution
14: END createSchedule
15: PROCEDURE: constructSolution(workflow)
16: while schedule is not completed do
17:    T ← get all unmapped ready tasks
18:    make a RCL for each t ∈ T
19:    subSolution ← select a resource randomly for each t ∈ T from its RCL
20:    solution ← solution ⋃ subSolution
21:    update information for further RCL making
22: end while
23: return solution
24: END constructSolution
25: PROCEDURE: localSearch(solution)
26: nSolution ← find a optimal local solution
27: return nSolution
28: END localSearch
```

Algorithm 10 describes a GRASP. In a GRASP, a number of iterations are conducted to search a possible optimal solution for scheduling tasks on resources. A solution is generated at each iterative step and the best solution is kept as the final schedule (Algorithm 10:line 1-6). A GRASP is terminated when the specified termination criterion is satisfied, for example, after com-

pleting a certain number of interations. In general, there are two phases in each interation: *construction phase* and *local search phase*.

The construction phase (Algorithm 10:line 8 and line 15-24) generates a feasible solution. A feasible solution for the workflow scheduling problem is required to meet the following conditions: a task must be started after all its predecessors have been completed; every task appears once and only once in the schedule. In the construction phase, a *restricted candidate list* (RCL) is used to record the best candidates, but not necessarily the top candidate of the resources for processing each task. There are two major mechanisms that can be used to generate the RCL, *cardinality-based RCL* and *value-based RCL*.

Algorithm 11 Construction phase procedure for workflow scheduling.

1: **PROCEDURE:** constructSolution(Ω)
2: **while** schedule is not completed **do**
3: $availTasks \leftarrow$ get unmapped ready tasks
4: $subSolution \leftarrow$ schedule($availTasks$)
5: $solution \leftarrow solution \bigcup subSolution$
6: **end while**
7: **return** solution
8: **END** constructSolution
9: **PROCEDURE:** schedule($tasks$)
10: $availTasks \leftarrow tasks$
11: $pairs \leftarrow$
12: **while** $\exists t \in tasks$ not scheduled **do**
13: **for all** $t \in availTasks$ **do**
14: $availResources \leftarrow$ get available resources for t
15: **for all** $r \in availResources$ **do**
16: compute $increaseMakespan(t, r)$
17: $pairs \leftarrow pairs \bigcup (t, r)$
18: **end for**
19: **end for**
20: $minI \leftarrow$ minimum makespan increase over $availPairs$
21: $maxI \leftarrow$ maximum makespan increase over $availPairs$
22: $availPairs \leftarrow$ select pairs whose makespan increase is less than $minI + \alpha(maxI - minI)$
23: $(t', r') \leftarrow$ select a pair at random from $availPairs$
24: remove t' from $availTasks$
25: $solution \leftarrow solution \bigcup (t', r')$
26: **end while**
27: **return** solution
28: **END** schedule

The cardinality-based RCL records the k best rated solution components, while the value-based RCL records all solution components whose performance evaluated values are better than a better than a given threshold [31]. In the

GRASP, resource allocated to each task is randomly selected from its RCL (Algorithm 10: line 19). After allocating a resource to a task, the resource information is updated and the scheduler continues to process other unmapped tasks.

Algorithm 11 shows the detailed implementation of the construction phase for workflow scheduling presented by Blythe et al. [11] which uses a value-based RCL method. The scheduler estimates the makespan increase for each unmapped ready task (Algorithm 11: line 3-4 and line 13-19) on each resource that is able to process the task. A *makespan increase* of a task t on a resource r is the increase of the execution length to the current completion length (makespan) if r is allocated to t. Let $minI$ and $maxI$ be the lowest and highest makespan increase found respectively. The scheduler selects a task assignment randomly from the task and resource pair whose makespan increase is less than $minI + \alpha(maxI - minI)$, where is a parameter to determine how much variation is allowed for creating RCL for each task and $0 \leq \alpha \leq 1$.

Once a feasible solution is constructed, a local search is applied into the solution to improve it. The local search process searches local optima in the neighborhood of the current solution and generates a new solution. The new solution will replace the current constructed solution if its overall performance is better (i.e. its makespan is shorter than that of the solution generated) in the in the construction phase. Binato et al. [8] implementation of the local search phase for job-shop scheduling. It identifies the critical path in the disjunctive graph of the solution generated in the construction phase and swaps two consecutive operations in the critical path on the same machine. If the exchange improves the performance, it is accepted.

Genetic Algorithms (GAs)

Genetic Algorithms (GAs) [25] provide robust search techniques that allow a high-quality solution to be derived from a large search space in polynomial time by applying the principle of evolution. Using genetic algorithms to schedule task graphs in homogeneous and dedicated multiprocessor systems have been proposed in [31, 56, 64]. Wang et al. [54] have developed a genetic-algorithm-based scheduling to map and schedule task graphs on heterogeneous envoriments. Prodan and Fahringer [42] have employed GAs to schedule WIEN2k workflow [10] on Grids. Spooner et al. [47] have employed GAs to schedule sub-workflows in a local Grid site.

A genetic algorithm combines exploitation of best solutions from past searches with the exploration of new regions of the solution space. Any solution in the search space of the problem is represented by an individual (chromosome). A genetic algorithm maintains a population of individuals that evolves over generations. The quality of an individual in the population is determined by a *fitness function* . The fitness value indicates how good the individual is compared to others in the population.

A typical genetic algorithm is illustrated in Fig. 5.4. It first creates an initial population consisting of randomly generated solutions. After applying genetic operators, namely selection, crossover and mutation, one after the other, new offspring are generated. Then the evaluation of the fitness of each individual in the population is conducted. The fittest individuals are selected to be carried over next generation. The above steps are repeated until the termination condition is satisfied. Typically, a GA is terminated after a certain number of iterations, or if a certain level of fitness value has been reached [64].

The construction of a genetic algorithm for the scheduling problem can be divided into four parts [32]: the choice of representation of individual in the population; the determination of the fitness function; the design of genetic operators; the determination of probabilities controlling the genetic operators.

As genetic algorithms manipulate the code of the parameter set rather than the parameters themselves, an encoding mechanism is required to represent individuals in the population. Wang et al. [54] encoded each chromosome with two separated parts: the *matching string* and the *scheduling string*. Matching string represents the assignment of tasks on machines while scheduling string represents the execution order of the tasks (Fig. 5.5a.). However, a more intuitive scheme, *two-dimensional coding scheme* is employed by many [32,56,64] for scheduling tasks in distributed systems. As illustrated in Fig. 5.5c, each schedule is simplified by representing it as a 2D string. One dimension represents the numbers of resources while the other dimension shows the order of tasks on each resource.

A fitness function is used to measure the quality of the individuals in the population. The fitness function should encourage the formation of the solution to achieve the objective function. For example, the fitness function developed in [32] is $C_{max} - FT(I)$, where C_{max} is the maximum completion time observed so far and $FT(I)$ is the completion time of the individual I. As the objective function is to minimize the execution time, an individual with a large value of fitness is fitter than the one with a small value of fitness.

After the fitness evaluation process, the new individuals are compared with the previous generation. The selection process is then conducted to retain the fittest individuals in the population, as successive generations evolve. Many methods for selecting the fittest individuals have been used for solving task scheduling problems such as *roulette wheel selection*, *rank selection* and *elitism*.

The roulette wheel selection assigns each individual to a slot of a roulette wheel and the slot size occupied by each individual is determined by its fitness value. For example, there are four individuals (see Table 5.5) and their fitness values are 0.45, 0.30, 0.25 and 0.78, respectively. The slot size of an individual is calculated by dividing its fitness value by the sum of all individual fitness in the population. As illustrated in Fig. 5.6, *individual 1* is placed in the slot ranging from $0-0.25$ while *individual 2* is in the slot ranging from $0.26-0.42$. After that, a random number is generated between 0 and 1, which is used to determine which individuals will be preserved to the next generation. The

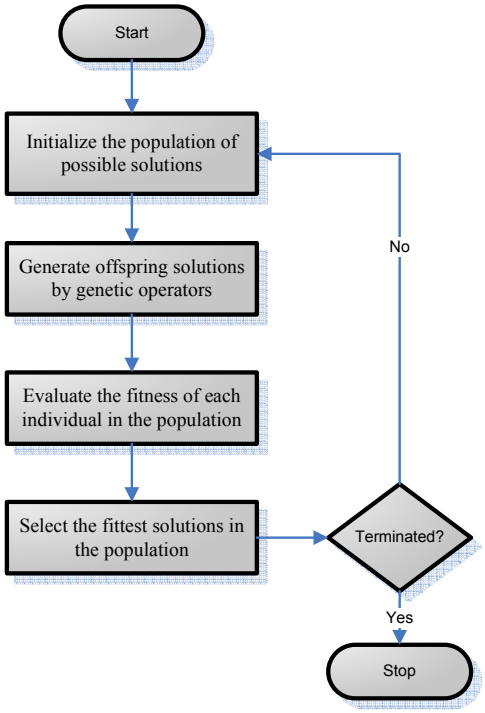


Fig. 5.4. Genetic Algorithms.

individuals with a higher fitness value are more likely to be selected since they occupy a larger slot range.

Table 5.5. Fitness Values and Slots for Roulette Wheel Selection.

Individual	Fitness value	Slot Size	Slot
1	0.45	0.25	0.25
2	0.30	0.17	0.42
3	0.25	0.14	0.56
4	0.78	0.44	1
Total	1.78	1	

The roulette wheel selection will have problems when there are large differences between the fitness values of individuals in the population [41]. For example, if the best fitness value is 95of all slots of the roulette wheel, other individuals will have very few chances to be selected. Unlike the roulette wheel selection in which the slot size of an individual is proportional to its fitness value, a rank selection process firstly sorts all individuals from best to worst according to their fitness values and then assigns slots based on their rank. For

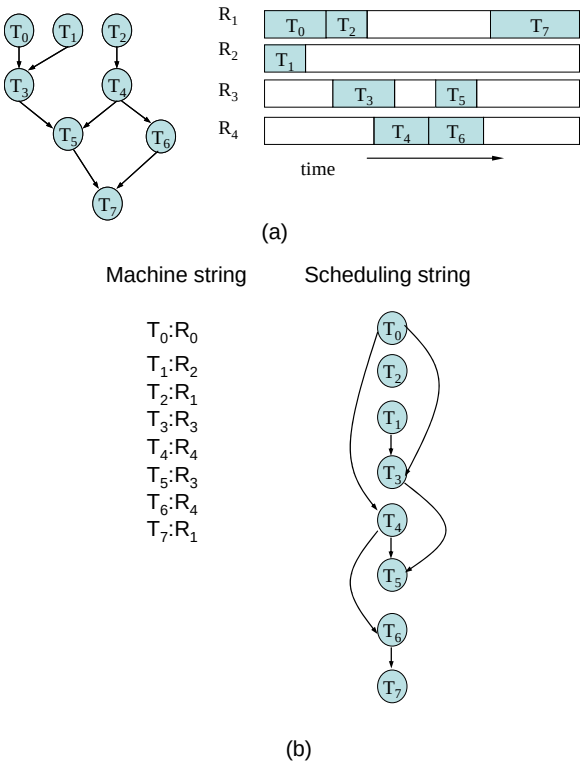


Fig. 5.5. (a) workflow application and schedule. (b) seperated machine string and scheduling string. (c) two-dimensional string.

Table 5.6. Fitness Values and Slots for Rank Selection.

Individual	Fitness value	Rank	Slot Size	Slot
1	0.45	3	0.3	0.3
2	0.30	2	0.2	0.5
3	0.25	1	0.1	0.6
4	0.78	4	0.4	1

example, the size of slots for each individual implemented by DOĞAN and Özgüner [16] is proportional to their rank value. As shown in Table 5.6, the size of the slot for individual I is defined as $PI = \frac{R(I)}{\sum_{i=1}^{n} R(i)}$, where $R(I)$ is the rank value of I and n is the number of all individuals. Both the roulette wheel selection and the rank selection select individuals according to their fitness value. The higher the fitness value, the higher the chance it will be selected into the next generation. However, this does not guarantee that the individual with the highest value goes to the next generation for reproduction. Elitism can be incorporated into these two selection methods, by first copying the fittest individual into the next generation and then using the rank selection

or roulette wheel selection to construct the rest of the population. Hou et al. [32] showed that the elitism method can improve the performance of the genetic algorithm.

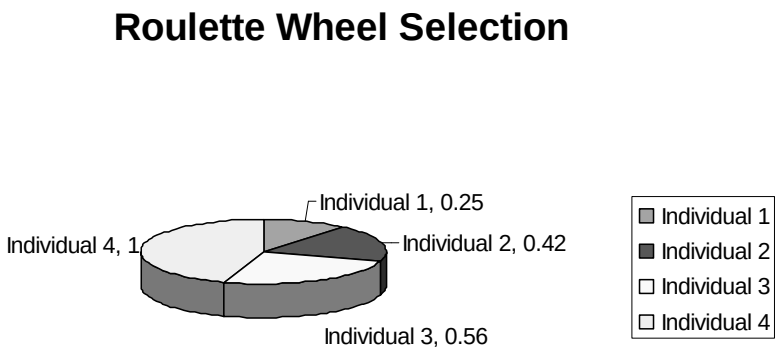


Fig. 5.6. Roulette Wheel Selection Example.

In addition to selection, crossover and mutation are two other major genetic operators. Crossovers are used to create new individuals in the current population by combining and rearranging parts of the existing individuals. The idea behind the crossover is that it may result in an even better individual by combining two fittest two fittest individuals [32]. Mutations allow a certain child to obtain features that are not possessed by either parent. It helps a genetic algorithm to explore new and potentially better genetic material than was previously considered. The frequency of mutation operation occurrence is controlled by the mutation rate whose value is determined experimentally [32].

Simulated Annealing (SA)

Simulated Annealing (SA) [38] derives from the Monte Carlo method for statistically searching the global. The concept is originally from the way in which crystalline structures can be formed into a more ordered state by use of the annealing process, which repeats the heating and slowly cooling a structure. SA has been used by YarKhan and Dongarra [57] to select a suitable size of a set of machines for scheduling a ScaLAPACK applicaton [9] in a Grid environment. Young et al. [58] have investigated performances of SA algorithms for scheduling workflow applications in a Grid envrionment.

A typical SA algorithm is illustrated in Fig. 5.7. The input of the algorithm is an initial solution which is constructed by assigning a resource to each task at random. There are several steps that the simulated annealing algorithm needs to go through while the temperature is decreased by a specified rate. The annealing process runs through a number of iterations at

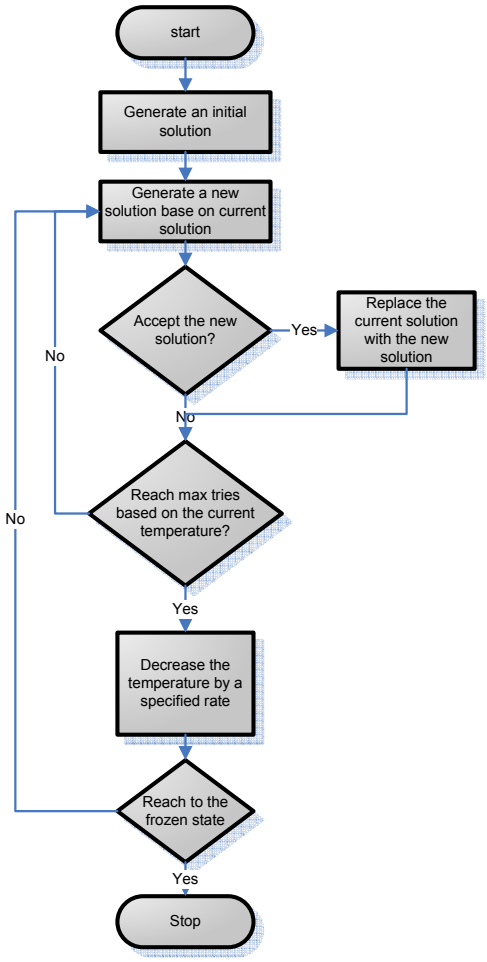


Fig. 5.7. Simulated Annealing.

each temperature to sample the search space. At each cycle, it generates a new solution by applying random change on the current solution. Young et al. [58] implemented this randomization by moving one task onto a different resource. Whether or not the new solution is accepted as a current solution is determined by the Metropolis algorithm [38, 58] shown in Algorithm 12. In the Metropolis algorithm, the new solution and the current solution are compared and the new solution is unconditionally accepted if it is better than the current one. In the case of the minimization problem of workflow scheduling, the better solution is one which has a lower execution time and the improved value is denoted as $d\beta$. In other cases, the new solution is accepted with the Boltzmann probability $e^{\frac{-d\beta}{T}}$ [38] where T is the current temperature. Once a

specified number of cycles have been completed, the temperature is decreased. The process is repeated until the lowest allowed temperature has been reached. During this process, the algorithm keeps the best solution so far, and returns this solution at termination as the final optimal solution.

Algorithm 12 Metropolis algorithm.

```
1: if dβ then
2:    return true
3: else if a random number less than e^(−dβ/T) then
4:    return true
5: else
6:    return false
7: end if
```

5.3.3 Comparison of best-effort scheduling algorithms

The overview of the best effort scheduling is presented in Table 5.7 and 5.8. In general, the heuristic based algorithms can produce a reasonable good solution in a polynomial time. Among the heuristic algorithms, individual task scheduling is simplest and only suitable for simple workflow structures such as a pipeline in which several tasks are required to be executed in sequential. Unlike individual task scheduling, list scheduling algorithms set the priorities of tasks in order to make an efficient schedule in the situation of many tasks compete for limited number of resources. The priority of the tasks determines their execution order. The batch mode approach orders the tasks required to be executed in parallel based on their execution time whereas the dependency mode approach orders the tasks based on the length of their critical path. The advantage of the dependency mode approach is that it intent to complete tasks earlier whose interdependent tasks required longer time in order to reduce the overall execution time. However, its complexity is higher since it is required to compute the critical path of all tasks. Another drawback of the dependency mode approach is that it cannot efficiently solve resource competition problem for a workflow consisting of many parallel tasks having the same length of their critical path. The dependency-batch mode approach can take advantage of both approaches, and Sakellariou and Zhao [45] shows that it outperforms the dependency mode approach in most cases. However, computing task priorities based on both batch mode and dependency mode approach results in higher scheduling time.

Even though data transmission time has been considered in the list scheduling approach, it still may not provide an efficient schedule for data intensive workflow applications, in which the majority of computing time is used for transferring data of results between the inter-dependent tasks. The main focus of the list scheduling is to find an efficient execution order of a set of parallel tasks and the determination of the best execution resource

Table 5.7. Comparison of Best-effort Workflow Scheduling Algorithms (Heuristics).

Scheduling Method		Algorithm	Complexity*	Features
Individual task scheduling		Myopic	$O(vm)$	Decision is based on one task.
List scheduling	Batch mode	Min-min	$O(vgm)$	Decision based on a set of parallel independent tasks.
	Dependency mode	HEFT	$O(v^2m)$	Decision based on the critical path of the task.
	Dependency-batch mode	Hybrid	$O(v^2m + vgm)$	Ranking tasks based on their critical path and re-ranking adjacent independent tasks by using a batch mode algorithm.
Cluster based scheduling Duplication based scheduling		THAN	$O(v^2)$	Replicating tasks to more than one resources in order to reduce transmission time.

*where v is the number of tasks in the workflow, m is the number of resources and g is the number of tasks in a group of tasks for the batch mode scheduling.

Table 5.8. Comparison of Best-effort Workflow Scheduling Algorithms (Meta-heuristics).

Scheduling Method	Features
Greedy randomized adaptive search procedure (GRASP)	Global solution obtained by comparing differences between randomized schedules over a number of iteration.
Genetic algorithms (GA)	Global solution obtained by combining current best solutions and exploiting new search region over generations.
Simulated annealing (SA)	Global solution obtained by comparing differences between schedules which are generated based on current accepted solutions over a number of iterations, while the acceptance rate is decreased.

for each task is based only on the information of current task. Therefore, it may not assign data inter-dependent tasks on resources among which an optimized data transmission path is provided. Both cluster based and duplication based scheduling approach focus on reducing communication delay among interdependent tasks. The clustering based approach minimizes the data transmission time by grouping heavily communicating tasks to a same task cluster and assigns all tasks in the cluster to one resource, in order to minimize the data transmission time, while duplication based approach duplicates data-interdependent tasks to avoid data transmission. However, the

restriction of the algorithms based on these two approaches up to date may not be suitable for all Grid workflow applications, since it assumes that heavily communicating tasks can be executed on a same resource. Tasks in Grid workflow applications can be highly heterogeneous and require different type of resources.

The meta-heuristics based workflow scheduling use guided random search techniques and exploit the feasible solution space iteratively. The GRASP generates a randomized schedule at each iteration and keeps the best solution as the final solution. The SA and GAs share the same fundamental assumption that an even better solution is more probably derived from good solutions. Instead of creating a new solution by randomized search, SA and GAs generate new solutions by randomly modifying current already know good solutions. The SA uses a point-to-point method, where only one solution is modified in each iteration, whereas GAs manipulate a population of solutions in parallel which reduce the probability of trapping into a local optimum [65]. Another benefit of producing a collection of solutions at each iteration is the search time can be significantly decreased by using some parallelism techniques.

Compared with the heuristics based scheduling approaches, the advantage of the meta-heuristics based approaches is that it produces an optimized scheduling solution based on the performance of entire workflow, rather than the partial of the workflow as considered by heuristics based approach. Thus, unlike heuristics based approach designed for a specified type of workflow application, it can produce good quality solutions for different types of workflow applications (e.g. different workflow structure, data- and computational-intensive workflows, etc). However, the scheduling time used for producing a good quality solution required by meta-heuristics based algorithms is significantly higher. Therefore, the heuristics based scheduling algorithms are well suited for a workflow with a simple structure, while the meta-heuristics based approaches have a lot of potential for solving large and complex structure workflows. It is also common to incorporate these two types of scheduling approaches by using a solution generated by a heuristic based algorithm as a start search point for the meta-heuristics based algorithms to generate a satisfactory solution in shorter time.

5.3.4 Dynamic Scheduling Techniques

The heuristics presented in last section assume that the estimation of the performance of task execution and data communication is accurate. However, it is difficult to predict accurately execution performance in community Grid environments due to its dynamic nature. In a community Grid, the utilization and availability of resources varies over time and a better resource can join at any time. Constructing a schedule for entire workflow before the execution may result in a poor schedule. If a resource is allocated to each task at the beginning of workflow execution, the execution environment may be very different at the time of task execution. A ‘best’ resource may become a

'worst' resource. Therefore, the workflow scheduler must be able to adapt the resource dynamics and update the schedule using up-to-date system information. Several approaches have been proposed to address these problems. In this section, we focus on the approaches which can apply the algorithms into dynamic environments.

For individual task and batch mode based scheduling, it is easy for the scheduler to use the most up-to-date information, since it takes into account only the current task or a group of independent tasks. The scheduler could map tasks only after their parent tasks become to be executed.

For dependency mode and metahueristics based scheduling, the scheduling decision is based on the entire workflow. In other words, scheduling current tasks require information about its successive tasks. However, it is very difficult to estimate execution performance accurately, since the execution environment may change a lot for the tasks which are late executed. The problems appear more significant for a long lasting workflow. In general, two approaches, task partitioning and iterative re-computing, have been proposed to allow these scheduling approaches to allocate resources more efficiently in a dynamic environment.

Task partitioning is proposed by Deelman et al. [17]. It partitions a workflow into multiple sub-workflows which are executed sequentially. Rather than mapping the entire workflow on Grids, allocates resources to tasks in one sub-workflow at a time. A new sub-workflow mapping is started only after the last mapped sub-workflow has begun to be executed. For each sub-workflow, the scheduler applies a workflow scheduling algorithm to generate an optimized schedule based on more up-to-date information.

Iterative re-computing keeps applying the scheduling algorithm on the remaining unexecuted partial workflow during the workflow execution. It does not use the initial assignment to schedule all workflow tasks but reschedule unexecuted tasks when the environment changes. A low-cost rescheduling policy has been developed by developed by Sakellariou and Zhao [44]. It overhead produced by rescheduling by conducting rescheduling only when the delay of a task execution impacts on the entire workflow execution.

In addition to mapping tasks before execution using up-to-date information, task migration [4, 42] has been widely employed to reschedule a task to another resource after it has been executed. The task will be migrated when the task execution is timed out or a better resource is found to improve the performance.

5.4 QoS-constraint based workflow scheduling

Many workflow applications require some assurances of quality of services (QoS) . For example, a workflow application for maxillo-facial surgery planning [16] needs results to be delivered before a certain time. For thus applications, workflow scheduling is required to be able to analyze users' QoS

requirements and map workflows on suitable resources such that the workflow execution can be completed to satisfy users' QoS constraints.

However, whether the execution can be completed within a required QoS not only depend on the global scheduling decision of the workflow scheduler but also depend on the local resource allocation model of each execution site. If the execution of every single task in the workflow cannot be completed as what the scheduler expects, it is impossible to guarantee the entire workflow execution. Instead of scheduling tasks on community Grids, QoS-constraint based schedulers should be able to interact with service-oriented Grid services to ensure resource availability and QoS levels. It is required that the scheduler can 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. Users normally would like to specify a QoS constraint for entire workflow. The scheduler needs to determine a QoS constraint for each task in the workflow, such that the QoS of entire workflow is satisfied.

In general, service-oriented Grid services are based on utility computing models. Users need to pay for resource access and service pricing is based on the QoS level and current market supply and demand. Therefore, unlike the scheduling strategy deployed in community Grids, QoS constraint based scheduling may not always need to complete the execution at earliest time. They sometimes may prefer to use cheaper services with a lower QoS that is sufficient to meet their requirements.

To date, supporting QoS in scheduling of workflow applications is at a very preliminary stage. Most QoS constraint based workflow scheduling heuristics are based on either *time* or *cost* constraints. Time is the total execution time of the workflow (known as *deadline*). Cost is the total expense for executing workflow execution including the usage charges by accessing remote resources and data transfer cost (known as *budget*). In this section, we present scheduling algorithms based on these two constraints, called *Deadline constrained* scheduling and *Budget constrained* scheduling . Table 5.9 and 5.10 presents the overview of QoS constrained workflow scheduling algorithms.

5.4.1 Deadline constrained scheduling

Some workflow applications are time critical and require the execution can be completed within a certain timeframe. Deadline constrained scheduling is designed for these applications to deliver results before the deadline. The distinction between the deadline constrained scheduling and the best-effort scheduling is that the deadline constrained scheduling also need to consider monetary cost when it schedules tasks. In general, users need to pay for service assess. The price is based on their usages and QoS levels. For example, services which can process faster may charges higher price. Scheduling the tasks based on the best-effort based scheduling algorithms presented in the previous sections, attempting to minimize the execution time will results in

Table 5.9. Overview of deadline constrained workflow scheduling algorithms.

Algorithm	Project	Organization	Application
Back-tracking	Menascé& Casalicchio	George Mason University, USA Univ. Roma "Tor Vergata", Italy	N/A
Deadline distribution	Gridbus	University of Melbourne, Australia	Randomly generated task graphs
Genetic algorithms	Gridbus	University of Melbourne, Australia	Randomly generated task graphs

Table 5.10. Overview of budget constrained workflow scheduling algorithms.

Algorithm	Project	Organization	Application
LOSS and GAIN	CoreGrid	University of Cyprus, Cyprus University of Manchester, UK	Randomly generated task graphs
Genetic algorithms	Gridbus	University of Melbourne, Australia	Randomly generated task graphs
Genetic algorithms	Gridbus	University of Melbourne, Australia	Randomly generated task graphs

high and unnecessary cost. Therefore, a deadline constrained scheduling algorithm intends to minimize the execution cost while meeting the specified deadline constraint.

Two heuristics have been developed to minimize the cost while meeting a specified time constraint. One is proposed by Menascé and Casalicchio [37] denoted as *Back-tracking*, and the other is proposed by Yu et al. [60] denoted as *Deadline Distribution*.

Back-tracking

The heuristic developed by Menascè and Casalicchio assigns available tasks to least expensive computing resources. An available task is an unmapped task whose parent tasks have been scheduled. If there is more than one available task, the algorithm assigns the task with the largest computational demand to the fastest resources in its available resource list. The heuristic repeats the procedure until all tasks have been mapped. After each iterative step,

the execution time of current assignment is computed. If the execution time exceeds the time constraint, the heuristic back-tracks the previous step and remove the least expensive resource from its resource list and reassigns tasks with the reduced resource set. If the resource list is empty the heuristic keep back-tracking to the previous step, reduces corresponding resource list and reassign the tasks.

Deadline/Time Distribution (TD)

Instead of back-tracking and repairing the initial schedule, the TD heuristic partitions a workflow and distributes overall deadline into each task based on their workload and dependencies. After deadline distribution, the entire workflow scheduling problem has been divided into several sub-task scheduling problems.

As shown in Fig. 5.8, in workflow task partitioning, workflow tasks are categorized as either *synchronization tasks* or *simple tasks*. A synchronization task is defined as a task which has more than one parent or child task. For example, T_1, T_{10} and T_{14} are synchronization tasks. Other tasks which have only one parent task and child task are simple tasks. For example, $T_2 - T_9$ and $T_{11} - T_{13}$ are simple tasks. Simple tasks are then clustered into a *branch*. A branch is a set of interdependent simple tasks that are executed sequentially between two synchronization tasks. For example, the branches in the example are $\{T_2, T_3, T_4\}$ and $\{T_5, T_6\}$, $\{T_7\}$, $\{T_8, T_9\}$, $\{T_{11}\}$ and $\{T_{12}, T_{13}\}$.

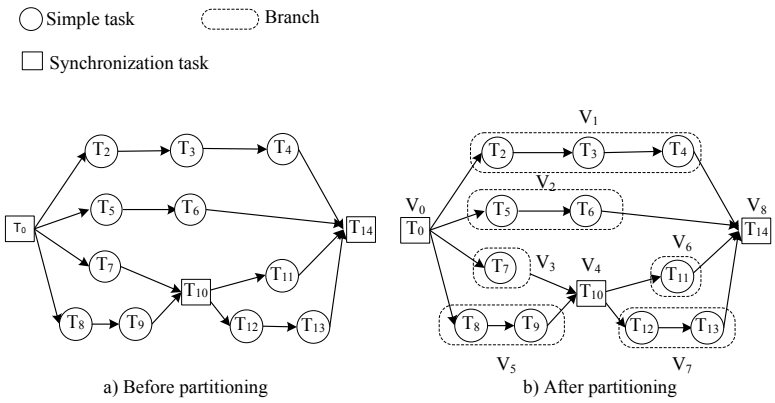


Fig. 5.8. Workflow Task Partition.

After task partitioning, workflow tasks Γ are then clustered into partitions and the overall deadline is distributed over each partition. The deadline assignment strategy considers the following facts:

- The cumulative expected execution time of a simple path between two synchronization tasks is same.

- The cumulative expected execution time of any path from an entry task to an exit task is equal to the overall deadline.
- The overall deadline is divided over task partitions in proportion to their minimum processing time.

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($dl[V_i]$), ready time ($rt[V_i]$), and expected execution time($eet[V_i]$). The ready time of is the earliest time when its first task can be executed. It can be computed according to its parent partitions and defined by:

$$rt[V_i] = \begin{cases} 0 & , T_{entry} \in V_i \\ \max\limits_{V_j \in PV_i} dl[V_j] & , otherwise \end{cases} \tag{5.5}$$

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:

$$eet[V_i] = dl[V_i] - rt[V_i] \tag{5.6}$$

A sub-deadline can be also assigned to each task based on the deadline of its task partition. If the task is a synchronization task, its sub-deadline is equal to the deadline of its task partition. However, if a task is a simple task of a branch, its sub-deadline is assigned by dividing the deadline of its partition based on its processing time. Let P_i be the set of parent tasks of T_i and S_i is the set of resources that are capable to execute T_i. t_i^j is the sum of input data transmission time and execution time of executing T_i on S_i. The sub-deadline of task in partition is defined by:

$$dl[T_i] = eet[T_i] + rt[V] \tag{5.6}$$

where

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

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

Once each task has its own sub-deadline, a local optimal schedule can be generated for each task. If each local schedule guarantees that their task execution can be completed within their sub-deadline, 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 sub-deadline at low execution cost.

5.4.2 Budget constrained scheduling

As the QoS guaranteed resources charges access cost, users would like to execute workflows based on the budget they available. Budget constrained scheduling intends to minimize workflow execution time while meeting users' specified budgets. Tsiakkouri et al. [52] present budget constrained scheduling called *LOSS* and *GAIN*.

LOSS and GAIN

LOSS and GAIN scheduling approach adjusts a schedule which is generated by a time optimized heuristic and a cost optimized heuristic to meet users' budget constraints, respectively. A time optimized heuristic attempts to minimize execution time while a cost optimization attempts to minimize execution cost.

If the total execution cost generated by time optimized schedule is not greater than the budget, the schedule can be used as the final assignment; otherwise, the LOSS approach is applied. The idea behinds LOSS is to gain a minimum loss in execution time for the maximum money savings, while amending the schedule to satisfy the budget. The algorithm repeats to re-assign the tasks with smallest values of the *LossWeight* until the budget constraint is satisfied. The LossWeight value for each task to each available resource is computed and it is defined by:

$$LossWeight(i,r) = \frac{T_{new} - T_{old}}{C_{old} - C_{new}} \tag{5.3}$$

where T_{old} and C_{old} are the execution time and corresponding cost of task T_i on the original resource assigned by the time optimized scheduling, T_{new} and C_{new} are the execution time of task Ti on resource r respectively. If Cold is not greater than C_{new}, the value of LossWeight is set to zero.

If the total execution cost generated by a cost optimized scheduler is less than the budget, the GAIN approach is applied to uses surplus to decrease the execution time. The idea behinds GAIN is to gain the maximum benefit in execution time for the minimum monetary cost, while amending the schedule. The algorithm repeats to re-assign the tasks with biggest value of the GainWeight until the cost exceeds the budget. The GainWeight value for each task to each available resource is computed and it is defined by:

$$GainWeight(i,r) = \frac{T_{old} - T_{new}}{C_{new} - C_{old}} \tag{5.3}$$

where T_{new}, T_{old}, C_{new} and C_{old} have the same meaning as in the LOSS approach. If T_{new} is greater than Told or C_{new} is equal to C_{old}, the value of GainWeight is set to zero.

5.4.3 Meta-heuristic based constrained workflow scheduling

A genetic algorithm [61] is also developed to solve the deadline and budget constrained scheduling problem. It defines a fitness function which consists of two components, *cost-fitness* and *time-fitness*. For the budget constrained scheduling, the cost-fitness component encourages the formation of the solutions that satisfy the budget constraint. For the deadline constrained scheduling, it encourages the genetic algorithm to choose individuals with less cost. The cost fitness function of an individual I is defined by:

$$F_{cost}(I) = \frac{c(I)}{B^{\alpha}(maxCost^{(1-\alpha)})}, \alpha = \{0, 1\} \tag{5.3}$$

where $c(I)$ is the sum of the task execution cost and data transmission cost of I, *maxCost* is the most expensive solution of the current population and B is the budget constraint. α is a binary variable and $\alpha = 1$ if users specify the budget constraint, otherwise $\alpha = 0$.

For the budget constrained scheduling, the time-fitness component is designed to encourage the genetic algorithm to choose individuals with earliest completion time from the current population. For the deadline constrained scheduling, it encourages the formation of individuals that satisfy the deadline constraint. The time fitness function of an individual I is defined by:

$$F_{time}(I) = \frac{t(I)}{D^{\beta}(maxTime^{(1-\beta)})}, \beta = \{0, 1\} \tag{5.3}$$

where $t(I)$ is the completion time of I, *maxTime* is the largest completion time of the current population and D is the deadline constraint. β is a binary variable and $\beta = 1$ if users specify the deadline constraint, otherwise $\beta = 0$.

For the deadline constrained scheduling problem, the final fitness function combines two parts and it is expressed as:

$$F(I) = \begin{cases} F_{time}(I), \, if F_{time}(I) > 1 \\ F_{cost}(I), \quad otherwise \end{cases} \tag{5.3}$$

For the budget constrained scheduling problem, the final fitness function combines two parts and it is expressed as:

$$F(I) = \begin{cases} F_{cost}(I), \, if F_{cost}(I) > 1 \\ F_{time}(I), \quad otherwise \end{cases} \tag{5.3}$$

In order to applying mutation operators in Grid environment, it developed two types of mutation operations, *swapping mutation* and *replacing mutation*. Swapping mutation aims to change the execution order of tasks in an individual that compete for a same time slot. It randomly selects a resource and

swaps the positions of two randomly selected tasks on the resource. Replacing mutation re-allocates an alternative resource to a task in an individual. It randomly selects a task and replaces its current resource assignment with a resource randomly selected in the resources which are able to execute the task.

5.4.4 Comparison of QoS constrained scheduling algorithms

The overview of QoS constrained scheduling is presented in Table 5.11 5.12. Comparing two heuristics for the deadline constrained problem, the back-tracking approach is more naïve. It is like a constrained based myopic algorithm since it makes a greedy decision for each ready task without planning in the view of entire workflow. It is required to track back to the assigned tasks once it finds the deadline constraint cannot be satisfied by the current assignments. It is restricted to many situations such as data flow and the distribution of execution time and cost of workflow tasks. It may be required to go through many iterations to modify the assigned schedule in order to satisfy the deadline constraint. In contrast, the deadline distribution makes a scheduling decision for each tasks based on a planned sub-deadline according to the workflow dependencies and overall deadline. Therefore, it has a better plan while scheduling current tasks and does not require tracing back the assigned schedule. However, different deadline distribution strategies may affect the performance of the schedule produced from one workflow structure to another.

Table 5.11. Comparison of deadline constrained workflow scheduling algorithms.

Algorithm	Features
Back-tracking	It assigns ready tasks whose parent tasks have been mapped to the least expensive computing resources and back-tracks to previous assignment if the current aggregative execution time exceeds the deadline.
Deadline distribution	It distributes the deadline over task partitions in workflows and optimizes execution cost for each task partition while meeting their sub-deadlines.
Genetic algorithms	It uses genetic algorithms to search a solution which has minimum execution cost within the deadline.

To date, the LOSS and GAIN approach is the only heuristic that addresses the budget constrained scheduling problem for Grid workflow applications. It takes advantage of heuristics designed for a single criteria optimization problem such as time optimization and cost optimization scheduling problem to

Table 5.12. Comparison of budget constrained workflow scheduling algorithms.

Algorithm	Features
LOSS and GAIN	It iteratively adjusts a schedule which is generated by a time optimized heuristic or a cost optimized heuristic based on its corresponding LOSS or GAIN weight rate of each task-resource pair, until the total execution cost meets users' budget constraint.
Genetic algorithms	It uses genetic algorithms to search a solution which has minimum execution time within the budget.

solve a multi-criteria optimization problem. It amends the schedule optimized for one factor to satisfy the other factor in the way that it can gain maximum benefit or minimum loss. Even though the original heuristics are targeted at the budget-constrained scheduling problem, such concept is easy to apply to other constrained scheduling. However, there exist some limitations. It relies on the results generated by an optimization heuristics for a single objective. Even though time optimization based heuristics have been developed over two decades, there is a lack of workflow optimization heuristics for other factors such as monitory cost based on different workflow application scenarios. In addition, large scheduling computation time could occur for data-intensive applications due to the weight re-computation for each pair of task and resource after amending a task assignment.

Unlike best-effort scheduling in which only one single objective (either optimizing time or system utilization) is considered, QoS constrained scheduling needs to consider more factors such as monetary cost and reliability. It needs to optimize multiple objectives among which some objectives are conflicting. However, with the increase of the number of factors and objectives required to be considered, it becomes infeasible to develop a heuristic to solve QoS constrained scheduling optimization problems. For this reason, we can believe that metahueristics based scheduling approach such as genetic algorithms will play more important role for the multi-objective and multi-constraint based workflow scheduling.

5.5 Simulation Results

In this section, we show an example of experimental comparisons for workflow scheduling algorithms. Basically, we compares deadline constrained scheduling heuristics which are presented in previous section.

5.5.1 Workflow Applications

Given that different workflow applications may have a different impact on the performance of the scheduling algorithms, a task graph generator is developed

to automatically generate a workflow based on the specified workflow structure, and 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 services. 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.

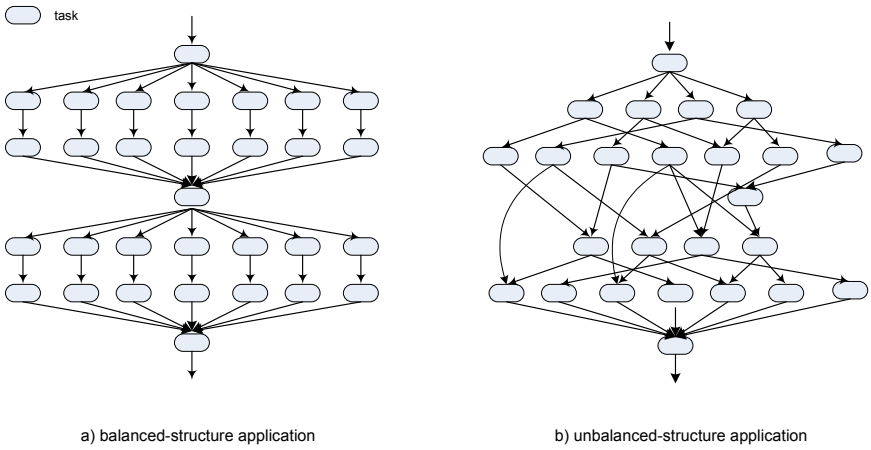


Fig. 5.9. Small portion of workflow applications.

According to many Grid workflow projects [11, 35, 55], workflow application structures can be categorized as either *balanced structure* or *unbalanced structure*. Examples of balanced structure include Neuro-Science application workflows [63] and EMAN refinement workflows [35], while the examples of unbalanced structure include protein annotation workflows [40] and Montage workflows [11]. Fig. 5.9 shows two workflow structures, a *balanced-structure application* and an *unbalanced-structure application*, used in our experiments. As shown in Fig. 5.9a, the balanced-structure application consists of several parallel pipelines, which require the same types of services but process different data sets. In Fig. 5.9b, 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 services, and their workload and I/O data varies significantly.

5.5.2 Experiment Setting

GridSim [48] is used to simulate a Grid environment for experiments. Fig. 5.10 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.

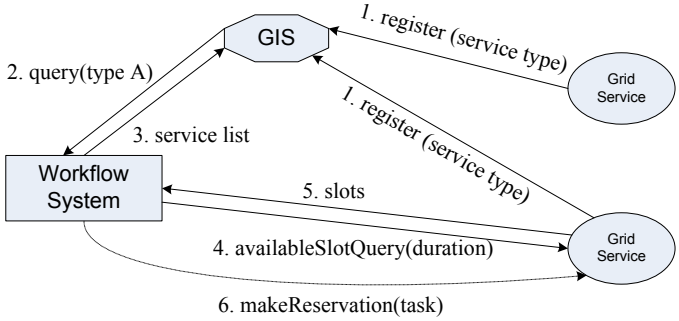


Fig. 5.10. Simulation environment.

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 capability. The topology of the system is such that all services are connected to one another, and the available network bandwidths between services are 100Mbps, 200Mbps, 512Mbps and 1024Mbps.

Table 5.13. Service speed and corresponding price for executing a task.

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

Table 5.14. Transmission bandwidth and corresponding price.

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

For the experiments, the cost that a user needs to pay for a workflow execution comprises of two parts: processing cost and data transmission cost. Table 5.13 shows an example of processing cost, while Table 5.14 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 a reasonable deadline constraint we also implemented a time optimization algorithm, HEFT, and a cost optimiza-

tion algorithm, *Greedy Cost*(GC). The HEFT algorithm is a list scheduling algorithm which 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 used for the experiments are based on the results of these two algorithms. Let T_{max} and T_{min} be the total execution time produced by GC and HEFT respectively. Deadline D is defined by:

$$D = T_{min} + k(T_{max} - T_{min}) \tag{5.3}$$

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.

5.5.3 Backtracing(BT) vs. Deadline/Time Distribution (TD)

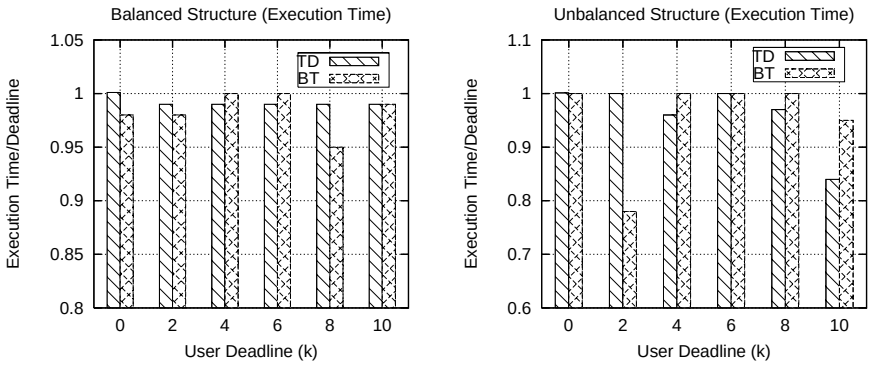
In this section, TD is compared with BackTracking denoted as BT on the two workflow applications, balanced and unbalanced. In order to show the results more clearly, we normalize the execution time and cost. Let C_{value} and T_{value} be the execution time and the monetary cost generated by the algorithms in the experiments respectively. The execution time is normalized by using T_{value}/D, and the execution cost by using C_{value}/C_{min}, where C_{min} is the minimum cost achieved Greedy Cost. The normalized values of the execution time should be no greater than one, if the algorithms meet their deadline constraints.

A comparison of the execution time and cost results of the two deadline constrained scheduling methods for the balanced-structure application and unbalanced-structure application is shown in Fig. 5.11 and Fig. 5.12 respectively. From Fig. 5.11, we can see that TD slightly exceeds deadline at $k = 0$, while BT can satisfy deadlines each time. For execution cost required by the two approaches shown in Fig. 5.12, TD significantly outperforms BT. TD saves almost 50execution cost when deadlines are relatively low. However, the two approaches produce similar results when deadline is greatly relaxed.

Fig. 5.13 shows the comparison of scheduling running time for two approaches. The scheduling time required by TD is 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 back-track for more iterations to adjust previous task assignments in order to meet tight deadlines.

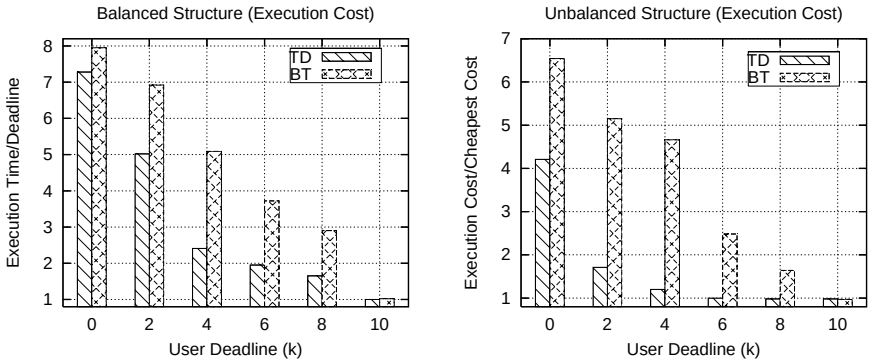
5.5.4 TD vs. Genetic Algorithms

In this section, the deadline constrained genetic algorithm is compared with the non-GA heuristics (i.e. TD) on the two workflow structures, balanced and unbalanced workflows.



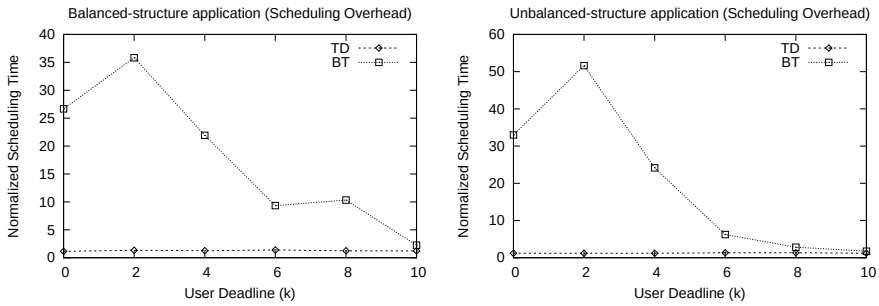
(a) balanced structure (b) unbalanced structure

Fig. 5.11. Execution time for scheduling balanced- and unbalanced-structure applications.



(a) balanced structure (b) unbalanced structure

Fig. 5.12. Execution cost for scheduling balanced- and unbalanced-structure applications.



(a) balanced structure (b) unbalanced structure

Fig. 5.13. Scheduling overhead for deadline constrained scheduling.

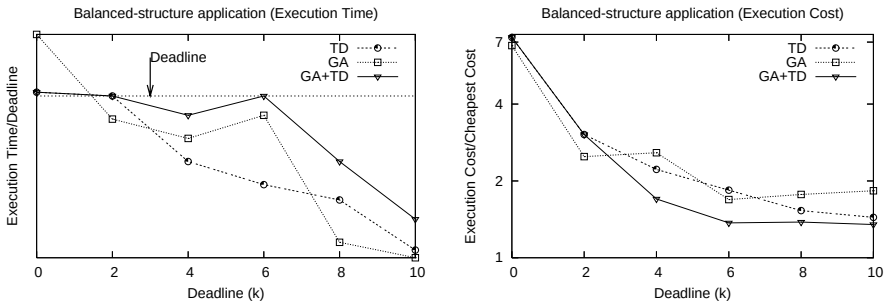
Table 5.15. Default settings.

Parameter	Value/Type
Population size	10
Maximum generation	100
Crossover probability	0.9
Reordering mutation probability	0.5
Replacing mutation probability	0.5
Selection scheme	elitism-rank selection
Initial individuals	randomly generated

The genetic algorithm is investigated by starting with two different initial populations. One initial population consists of randomly generated solutions, while the other initial population consists of a solution produced by TD together with other randomly generated solutions. In the result presentation, the results generated by GA with a completely random initial population is denoted by GA, while the results generated by GA which include an initial individual produced by the TD heuristic are denoted as GA+TD. The parameter settings used as the default configuration for the proposed genetic algorithm are listed in Table 5.15.

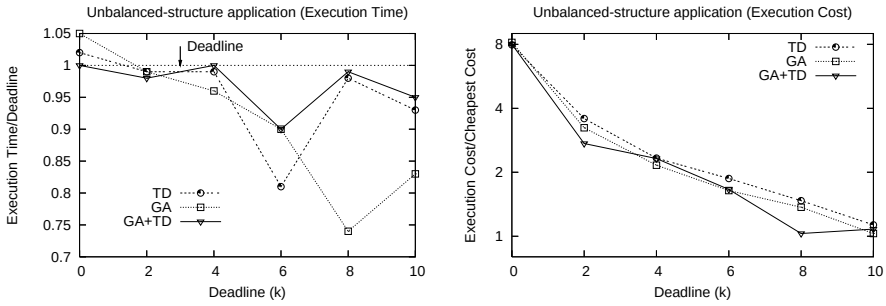
Fig. 5.14 and Fig. 5.15 compare the execution time and cost of using three scheduling approaches for scheduling the balanced-structure application and unbalanced-structure application with various deadlines respectively.

We can see that it is hard for both GA and TD to successfully meet the low deadline individually. As shown in Fig. 5.14a and 5.15a, the normalized execution times produced by TD and GA exceed 1 at tight deadline ($k = 0$), and GA performs worse than TD since its values is higher than TD, especially for balanced-structure application. However, the results are improved when incorporating GA and TD together by putting the solution produced by TD



(a) execution time (b) execution cost

Fig. 5.14. Normalized Execution Time and Cost for Scheduling Balanced-structure Application.



(a) execution time (b) execution cost

Fig. 5.15. Normalized Execution Time and Cost for Scheduling Unbalanced-structure Application.

into the initial population of GA. As shown in Fig. 5.15a, the value of GA+TD is much lower than that of GA and TD at the tight deadline.

As the deadline increases, both GA and TD can meet the deadline (see Fig. 5.14a and 5.15a) and GA can outperform TD. For example, execution time (see Fig. 5.14a) and cost (see Fig. 5.14b) generated by GA at $k = 2$ are lower than that of TD. However, as shown in Fig. 5.14b) the performance of GA is reduced and TD can perform better, when the deadline becomes very large ($k = 8$ and 10). In general, GA+TD performs best. This shows that the genetic algorithm can improve the results returned by other simple heuristics by employing these heuristic results as individuals in its initial population.

5.6 Conclusions

In this chapter, we have presented a survey of workflow scheduling algorithms for Grid computing. We have categorized current existing Grid workflow scheduling algorithms as either best-effort based scheduling or QoS constraint based scheduling.

Best-effort scheduling algorithms target on community Grids in which resource providers provide free access. Several heuristics and metahueristics based algorithms which intend to optimize workflow execution times on community Grids have been presented. The comparison of these algorithms in terms of computing time, applications and resources scenarios has also been examined in detail. Since service provisioning model of the community Grids is based on best effort, the quality of service and service availability cannot be guaranteed. Therefore, we have also discussed several techniques on how to employ the scheduling algorithms in dynamic Grid environments.

QoS constraint based scheduling algorithms target on utility Grids in which service level agreements are established between service providers and consumers. In general, users are charged for service access based on the usage and QoS levels. The objective functions of QoS constraint based scheduling algorithms are determined by QoS requirements of workflow applications. In this chapter, we have focused on examining scheduling algorithms which intend to solve performance optimization problems based on two typical QoS constraints, deadline and budget.

Acknowledgment

We would like to thank Hussein Gibbins and Chee Shin Yeo for their comments on this paper. This work is partially supported through Australian Research Council (ARC) Discovery Project grant.

References

1. J. Almond, D. Snelling, UNICORE: Uniform Access to Supercomputing as an Element of Electronic Commerce, Future Generation Computer Systems 15:539-548, NH-Elsevier, 1999.
2. The Austrian Grid Consortium. http://www.austrangrid.at.
3. R. Bajaj and D. P. Agrawal, Improving Scheduling of Tasks in a Heterogeneous Environment, *IEEE Transactions on Parallel and Distributed Systems*, 15:107-118, 2004.
4. F. Berman et al., New Grid Scheduling and Rescheduling Methods in the GrADS Project, *International Journal of Parallel Programming (IJPP)*, 33(2-3):209-229, 2005.

5. G. B. Berriman et al., Montage : a Grid Enabled Image Mosaic Service for the National Virtual Observatory. *ADASS XIII, ASP Conference Series*, 2003.
6. G. Berti et al., Medical Simulation Services via the Grid, *HealthGRID 2003 conference*, Lyon, France, January 16-17, 2003.
7. S. Benkner et al., VGE - A Service-Oriented Grid Environment for On-Demand Supercomputing, *The 5th IEEE/ACM International Workshop on Grid Computing (Grid 2004)*, Pittsburgh, PA, USA, November 2004.
8. S. Binato et al., A GRASP for job shop scheduling. *Essays and surveys on meta-heuristics*, pp.59-79, Kluwer Academic Publishers, 2001.
9. L. S. Blackford et al., ScaLAPACK: a linear algebra library for message-passing computers. *the Eighth SLAM Conference on Parallel Processing for Scientific Computing(Minneapolis, MN, 1997)*, pp.15, Philadelphia, PA, USA, 1997.
10. P. Blaha et al., WIEN2k: An Augmented Plane Wave plus Local Orbitals Program for Calculating Crystal Properties. Institute of Physical and Theoretical Chemistry, Vienna University of Technology, 2001.
11. J. Blythe et al., Task Scheduling Strategies for Workflow-based Applications in Grids, *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005)*, 2005
12. T. D. Braun, H. J. Siegel, and N. Beck, A Comparison of Eleven static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems, *Journal of Parallel and Distributed Computing*, 61:801-837, 2001.
13. R. Buyya and S. Venugopal, The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An overview and Status Report, *The 1st IEEE International Workshop on Grid Economics and Business Models, GECON 2004,* Seoul, Korea, April 23, 2004.
14. H. Casanova et al., Heuristics for Scheduling Parameter Sweep Applications in Grid Environments, *The 9th Heterogeneous Computing Workshop (HCW'00),* April. 2000.
15. K. Cooper et al., New Grid Scheduling and Rescheduling Methods in the GrADS Project, *NSF Next Generation Software Workshop, International Parallel and Distributed Processing Symposium*, Santa Fe, April 2004.
16. A. DOĞAN and F. Özgüner, Genetic Algorithm Based Scheduling of Meta-Tasks with Stochastic Execution Times in Heterogeneous Computing Systems, *Cluster Computing*, 7:177-190, Kluwer Academic Publishers, Netherlands, 2004.
17. E. Deelman et al., Pegasus: Mapping scientific workflows onto the grid, *European Across Grids Conference*, pp. 11-20, 2004.
18. 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.
19. T. A. Feo and M. G. C. Resende, Greedy Randomized Adaptive Search Procedures, *Journal of Global Optimization*, 6:109-133, 1995.
20. S. Fitzgerald et al., A Directory Service for Configuring High-Performance Distributed Computations, *The 6th IEEE Symposium on High-Performance Distributed Computing*, Portland State University, Portland, Oregon August 5-8, 1997.
21. I. Foster and C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, *International Journal of Supercomputer Applications*, 11(2): 115-128, 1997.
22. I. Foster and C. Kesselman (Eds.). *The Grid: Blueprint for a Future Computing Infrastructure,* Morgan Kaufmann Publishers, USA, 1999.

23. I. Foster et al., Chimera: A Virtual Data System for Representing, Querying and Automating Data Derivation, *The 14th Conference on Scientific and Statistical Database Management*, Edinburgh, Scotland, July 2002.
24. I. Foster et al., The Physiology of the Grid, Open Grid Service Infrastructure WG, *Global Grid Forum*, 2002.
25. D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.
26. D. E. Goldberg and K. Deb, A comparative analysis of selection schemes used in genetic algorithms, *Foundations of Genetic Algorithms*, pp.69-93, Morgan Kaufmann, 1991.
27. A. Grimshaw and W. Wulf, The Legion vision of a worldwide virtual computer, *Communications of the ACM,* 40(1):39-45, 1997.
28. X. He, X. Sun, and G. von Laszewski, QoS Guided Min-Min Heuristic for Grid Task Scheduling. *Journal of Computer Science and Technology*, 18(4):442-451, 2003.
29. F. S. Hillier and G. J. Lieberman, *Introduction to Operations Research*, McGraw-Hill Science, 2005.
30. D. Hollinsworth. The Workflow Reference Model, *Workflow Management Coalition*, TC00-1003, 1994.
31. H. H. Hoos and T. Stützle, *Stochastic Local Search: Foundation and Applications*, Elsevier Science and Technology, 2004.
32. E. S. H. Hou, N. Ansari, and H. Ren, A Genetic Algorithm for Multiprocessor Scheduling, *IEEE Transactions on Parallel and Distributed Systems,* 5(2):113-120, Feb. 1994.
33. Y. K. Kwok and I. Ahmad, Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors, *ACM Computing Surveys,* 31(4):406-471, Dec. 1999.
34. S. Ludtke, P. Baldwin, and W. Chiu. EMAN: Semiautomated software for high-resolution single-particle reconstructions. *Journal of Structural Biology*, 128:82-97, 1999.
35. A. Mandal et al., Scheduling Strategies for Mapping Application Workflows onto the Grid, *IEEE International Symposium on High Performance Distributed Computing (HPDC 2005)*, 2005.
36. A. Mayer et al., Workflow Expression: Comparison of Spatial and Temporal Approaches. *Workflow in Grid Systems Workshop, GGF-10,* Berlin, March 9, 2004.
37. D. A. Menascè and E. Casalicchio, A Framework for Resource Allocation in Grid Computing, *The 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04)*, Volendam, The Netherlands, Oct. 5-7 2004.
38. N. Metropolis et al., Equations of state calculations by fast computing machines. *Joural of Chemistry and Physics,* 21:1087-1091, 1953.
39. M. Maheswaran et al., Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computng Systems. *The 8th Heterogeneous Computing Workshop (HCW'99)*, San Juan, Puerto Rico, Apr. 12 1999.
40. A. O'Brien, S. Newhouse, and J. Darlington, Mapping of Scientific Workflow within the e-Protein project to Distributed Resources, *UK e-Science All Hands Meeting*, Nottingham, UK, 2004.
41. M. Obitko. Introduction to Genetic Algorithms. http://cs.felk.cvut.cz/ xobitko/ga/. [March 2006]

42. R. Prodan and T. Fahringer, Dynamic Scheduling of Scientific Workflow Applications on the Grid using a Modular Optimisation Tool: A Case Study, *The 20th Symposium of Applied Computing (SAC 2005)*, Santa Fe, New Mexico, USA, March 2005. ACM Press.
43. P. Rutschmann and D. Theiner, An inverse modelling approach for the estimation of hydrological model parameters. *Journal of Hydroinformatics*, 2005.
44. 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.
45. R. Sakellariou and H. Zhao, A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems, *The 13th Heterogeneous Computing Workshop (HCW 2004)*, Santa Fe, New Mexico, USA, April 26, 2004.
46. Z. Shi and J. J. Dongarra, Scheduling workflow applications on processors with different capabilities, *Future Generation Computer Systems*, 22:665-675, 2006.
47. D. P. Spooner et al., Performance-aware Workflow Management for Grid Computing, *The Computer Journal*, Oxford University Press, London, UK, 2004.
48. A. Sulistio and R. Buyya, A Grid Simulation Infrastructure Supporting Advance Reservation, *The 16th International Conference on Parallel and Distributed Computing and Systems (PDCS 2004)*, MIT Cambridge, Boston, USA, Nov. 9-11 2004.
49. T. Tannenbaum et al., Condor - A Distributed Job Scheduler, *Computing with Linux*, The MIT Press, MA, USA, 2002.
50. G. Thickins, Utility Computing: The Next New IT Model, *Darwin Magazine*, April 2003.
51. H. Topcuoglu, S. Hariri, and M. Y. Wu. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing, *IEEE Transactions on Parallel and Distributed Systems*, 13(3): 260-274, March 2002.
52. E. Tsiakkouri et al., Scheduling Workflows with Budget Constraints, *The CoreGRID Workshop on Integrated research in Grid Computing*, S. Gorlatch and M. Danelutto (Eds.), Technical Report TR-05-22, University of Pisa, Dipartimento Di Informatica, Pisa, Italy, Nov. 28-30, 2005, pp. 347-357.
53. J. D. Ullman, NP-complete Scheduling Problems, *Journal of Computer and System Sciences,* 10:384-393, 1975.
54. L. Wang et al., Task Mapping and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach, *Journal of Parallel and Distributed Computing,* 47:8-22, 1997.
55. M. Wieczorek, R. Prodan, and T. Fahringer, Scheduling of Scientific Workflows in the ASKALON Grid Enviornment, *ACM SIGMOD Record*, 34(3):56-62, Sept. 2005.
56. A. S. Wu, et al., An Incremental Genetic Algorithm Approach to Multiprocessor Scheduling, *IEEE Transactions on Parallel and Distributed Systems*, 15(9):824-834, September 2004.
57. A. YarKhan and J. J. Dongarra. Experiments with Scheduling Using Simulated Annealing in a Grid Environment. *The 3rd International Workshop on Grid Computing (Grid 2002)*, Baltimore, MD, USA, November 2002.
58. L. Young et al., Scheduling Architecture and Algorithms within the ICENI Grid Middleware, *UK e-Science All Hands Meeting*, IOP Publishing Ltd, Bristol, UK, Nottingham, UK, Sep. 2003, pp. 5-12.

59. J. Yu and R. Buyya, A Taxonomy of Workflow Management Systems for Grid Computing, *Journal of Grid Computing*, Springer, 3(3-4): 171-200, Spring Science+Business Media B.V., New York, USA, Sept. 2005.
60. J. Yu, R. Buyya, and C.K. Tham, A Cost-based Scheduling of Scientific Workflow Applications on Utility Grids, *The first IEEE International Conference on e-Science and Grid Computing*, Melbourne, Australia, Dec. 5-8, 2005.
61. J. Yu and R. Buyya, Scheduling Scientific Workflow Applications with Deadline and Budget Constraints using Genetic Algorithms, *Scientific Programming*, 14(3-4): 217 - 230, IOS Press, Amsterdam, The Netherlands, 2006.
62. H. Zhao and R. Sakellariou, An experimental investigation into the rank function of the heterogeneous earliest finish time shceulding algorithm, *Euro-Par 2003*, pp. 189-194.
63. Y. Zhao et al., Grid Middleware Services for Virtual Data Discovery, Composition, and Integration, *The Second Workshop on Middleware for Grid Computing,* Toronto, Ontario, Canada, 2004.
64. A. Y. Zomaya, C. Ward, and B. Macey, Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues, *IEEE Transactions on Parallel and Distributed Systems*, 10(8):795-812, Aug. 1999.
65. A. Y. Zomaya, Y. H. Teh, Observations on Using Genetic Algorithms for Dynamic Load-Balancing, *IEEE Transactions on Parallel and Distributed Systems*, 12(9):899-911, Sept. 2001.