Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms

Jia Yu[*] and Rajkumar Buyya
Grid Computing and Distributed Systems (GRIDS) Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, VIC 3010 Australia
E-mail: {jiayu, raj}@csse.unimelb.edu.au

Abstract. Grid technologies have progressed towards a service-oriented paradigm that enables a new way of service provisioning based on utility computing models, which are capable of supporting diverse computing services. It facilitates scientific applications to take advantage of computing resources distributed world wide to enhance the capability and performance. Many scientific applications in areas such as bioinformatics and astronomy require workflow processing in which tasks are executed based on their control or data dependencies. Scheduling such interdependent tasks on utility Grid environments need to consider users' QoS requirements. In this paper, we present a genetic algorithm approach to address scheduling optimization problems in workflow applications, based on two QoS constraints, deadline and budget.

Keywords: Grid workflow, workflow scheduling, utility Grids, deadline constrained scheduling, budget constrained scheduling

1. Introduction

Utility computing [28] has emerged as a new service provisioning model [7] and is capable of supporting diverse computing services such as servers, storage, network and applications for e-Business and e-Science over a global network. For utility computing based services, users consume the services when they need to, and pay only for what they use. With economy incentive, utility computing encourages organizations to offer their specialized applications and other computing utilities as services so that other individuals/organizations can access these resources remotely. Therefore, it facilitates individuals/organizations to develop their own core activities without maintaining and developing fundamental infrastructure. In the recent past, providing utility computing services has been reinforced by service-oriented Grid computing [2,10], that creates an infrastructure for enabling users to consume services transparently over a secure, shared, scalable, sustainable and standard world-wide network environment.

Table 1 shows some differences between community Grids and utility Grids in terms of availability, Quality of Services (QoS) and pricing. In utility Grids, users can make a reservation with a service provider in advance to ensure the service availability, and users can also negotiate with service providers on service level agreements for required QoS. Compared with utility Grids, service availability and QoS in community Grids may not be guaranteed. However, community Grids provide free access, whereas users need to pay for service access in utility Grids. In general, the service pricing is based on the QoS level and current market supply and demand.

Many Grid applications in areas such as bioinformatics and astronomy require workflow processing in which tasks are executed based on their control or data dependencies. As a result, a number of Grid

[*]Corresponding author.

Table 1
Community Grids vs. Utility Grids

	Community Grids	Utility Grids
Availability	Best effort	Advanced reservation
QoS	Best effort	Contract/SLA
Pricing	Not considered or free access	Usage, QoS level, Market supply and demand

workflow management systems [6,8,14,16,19,21,26, 30] with scheduling algorithms have been developed. They facilitate the execution of workflow applications and minimize their execution time on Grids. However, to impose a workflow paradigm on utility Grids, execution cost must also be considered when scheduling tasks on resources. The price of a utility service is mainly determined by its QoS level such as the processing speed of the service. Typically, service providers charge higher prices for higher QoS. Users may not always need to complete workflows earlier than they require. They sometimes may prefer to use cheaper services with a lower QoS that is sufficient to meet their requirements.

Given this motivation, we focus on developing workflow scheduling based on user's QoS constraints. Unlike the time optimization scheduling problem in which only execution time needs to be considered, constrained workflow execution optimization problems are required to consider many factors such as time, monetary cost, reliability and security. It may not be feasible to develop a simple heuristic to solve such complex problems. Therefore, we investigate metaheuristics capable of being applied to complex domains. In this paper, we propose a genetic algorithm based scheduling heuristic to solve performance optimization problems based on two typical QoS constraints, deadline and budget, for the workflow execution on "pay-per-use" services.

The remainder of the paper is organized as follows. We introduce the problem overview in Section 2 including problem definition and performance estimation approaches. Our proposed genetic algorithm based workflow scheduling approach is presented in Section 3. Experimental details and simulation results are presented in Section 4. We introduce related work in Section 5. Finally, we conclude the paper with directions for further work in Section 6.

2. Problem overview

2.1. Problem description

In our approach, we model a workflow application as a Directed Acyclic Graph (DAG). Let Γ be the finite set of tasks $T_i (1 \leqslant i \leqslant n)$. Let Λ be the set of directed arcs of the form (T_i, T_j) where T_i is called a parent task of T_j, and T_j the child task of T_i. We assume that a child task cannot be executed until all of its parent tasks have been completed.

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

Let B be the cost constraint (budget) and D be the time constraint (deadline) specified by the users for workflow execution. The budget constrained scheduling problem is to map every T_i onto a suitable S_i^j to minimize the execution time of the workflow and complete it within B. The deadline constrained scheduling problem is to map every T_i onto a suitable S_i^j to minimize the execution cost of the workflow and complete it within D.

2.2. Performance estimation

Performance estimation is crucial to generate an accurate schedule for advance reservations. Different performance estimation approaches can be applied to different types of utility service. We classify existing utility services as either *resource services* or *application services*.

Resource services provide hardware resources such as computing processors, network resources, storage and memory, as a service for remote clients. To submit tasks to resource services, the scheduler needs to determine the number of resources and duration required to run tasks on the discovered services. The performance estimation for resource services can be achieved by using existing performance estimation techniques (e.g. analytical modeling [20], empirical and historical data [18,24]) to predict task execution time on every discovered resource service.

Application services allow remote clients to use their specialized applications. Unlike resource services, an application service is capable of providing estimated service times based on the metadata of users' service requests [1]. As a result, the task execution time can be obtained by the application providers.

3. Proposed scheduling approaches

Workflow scheduling focuses on mapping and managing the execution of inter-dependent tasks on diverse utility services. In general, the problem of mapping tasks on distributed services belongs to a class of problems known as "NP hard problem". For such problems, no known algorithms are able to generate the optimal solution within polynomial time. Although the workflow scheduling problem can be solved by using exhaustive search, the complexity of the methods for solving it is very large.

Genetic algorithms (GAs) [12] 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. A genetic algorithm combines the 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 (chromosomes). 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 consists of the following steps: (1) create an initial population consisting of randomly generated solutions. (2) generate new offspring by applying genetic operators, namely selection, crossover and mutation, one after the other. (3) evaluate the fitness value of each individual in the population. (4) repeat steps 2 and 3 until the algorithm converges.

In order to using genetic algorithms concept to solve the workflow scheduling problem, we need to determine the representation of individual in the population, the fitness function and genetic operations. The details of our approach are presented in following subsections.

3.1. Problem representation

For the workflow scheduling problem, a feasible solution is required to meet the following conditions: (1) A task can only be started after all its predecessors have completed. (2) Every task appears once and only once in the schedule. (3) Each task must be allocated to one available time slot of a service capable of executing the task.

Each individual in the population represents a feasible solution to the problem, and consists of a vector of task assignments. Each task assignment includes four elements: *taskID, serviceID, startTime,* and *endTime*. *taskID* and *serviceID* identify to which service each task is assigned. *startTime* and *endTime* indicate the time frame allocated on the service for the task execution. However, involving time frames during the genetic operation may lead to a very complicated situation, because any change made to a task could require adjusting the values of *startTime* and *endTime* of its successive tasks. Therefore, we simplify the operation strings used for genetic manipulation by ignoring the time frames. The operation strings encode only the service allocation for each task and the order of tasks allocated on each service. After crossover and mutations, a time slot assignment method is applied to transfer an operation string to a feasible schedule.

In a workflow, the execution order of interdependent tasks is controlled by their dependencies, meaning that a task is always executed after its immediate parent tasks. However, many independent tasks, for instance T_3 and T_4 in the example workflow shown in Fig. 1 may compete for the same time slot on a service. Different execution priorities of such parallel tasks within the workflow may impact the performance of workflow execution significantly. For this reason, the solution representation strings are required to show the order of task assignments on each service in addition to service allocation of each task. We use a 2D string to represent a schedule as illustrated in Fig. 1. One dimension represents the numbers of services while the other dimension shows the order of tasks on each service. Two-dimensional strings are then converted into a one-dimensional string for genetic manipulations. The number in brackets in the one-dimensional string represents the identity number of the service on which the task is allocated.

3.2. Fitness function

A fitness function is used to measure the quality of the individuals in the population according to the given optimization objective. As the goal of the scheduling is to minimize the performance based on two factors, time and monetary cost, the fitness function separates evaluation into two parts: *cost-fitness* and *time-fitness*. Both functions use two binary variables, α and β. If

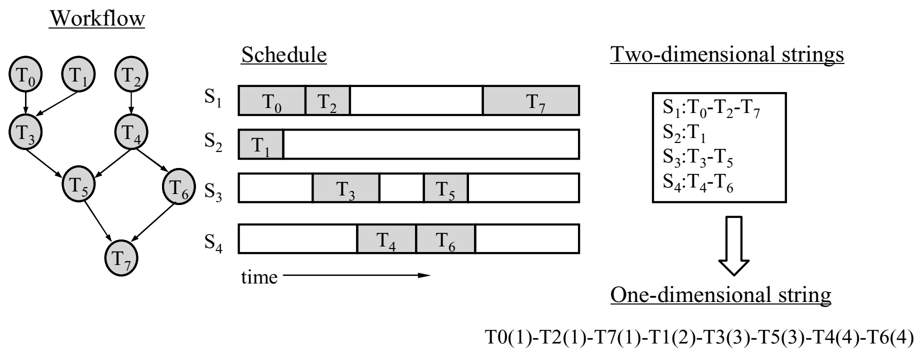


Fig. 1. Illustration of problem encoding.

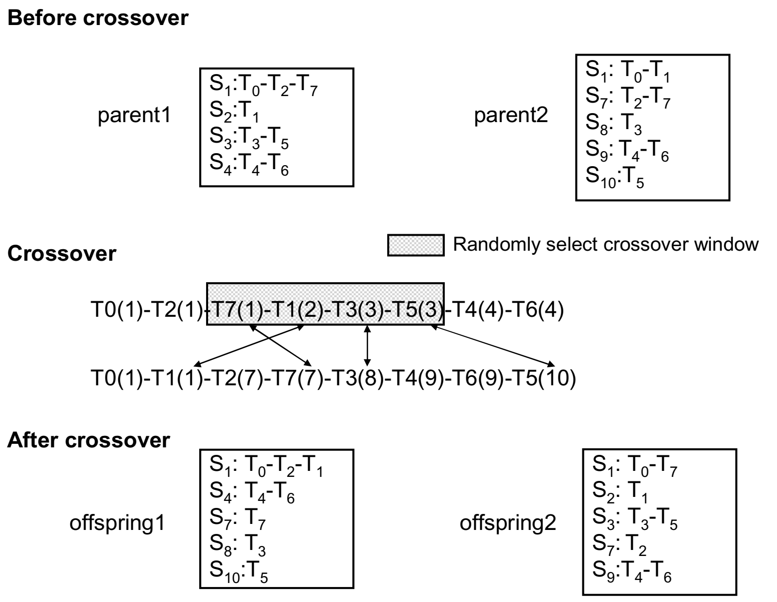


Fig. 2. Illustration of crossover operation.

users specify a budget constraint, then $\alpha = 1$ and $\beta = 0$. If users specify a deadline, then $\alpha = 0$ and $\beta = 1$.

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:

$$F_{\cos t}(I) = \frac{c(I)}{B^{\alpha} \times \mathit{maxCost}^{(1-\alpha)}},$$

where $c(I)$ is the sum of the task execution cost and data transmission cost of I and $c(I) = \sum_{T_i \in I} c_i^k$, $1 \leqslant k \leqslant m_i$, *maxCost* is the most expensive solution of the current population, and B is the budget of the workflow.

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:

$$F_{\text{time}}(I) = \frac{t(I)}{D^{\beta} \times \mathit{maxTime}^{(1-\beta)}},$$

where $t(I)$ is the completion time of I, maxTime is the largest completion time of the current population, and D is the deadline of the workflow.

The final fitness function combines two parts and it is expressed as:

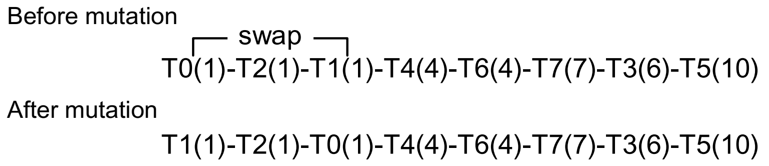

Before mutation
swap
T0(1)-T2(1)-T1(1)-T4(4)-T6(4)-T7(7)-T3(6)-T5(10)
After mutation
T1(1)-T2(1)-T0(1)-T4(4)-T6(4)-T7(7)-T3(6)-T5(10)

Fig. 3. Illustration of swapping mutation operation.

Task	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7
Service Type	A	A	A	B	C	B	C	A

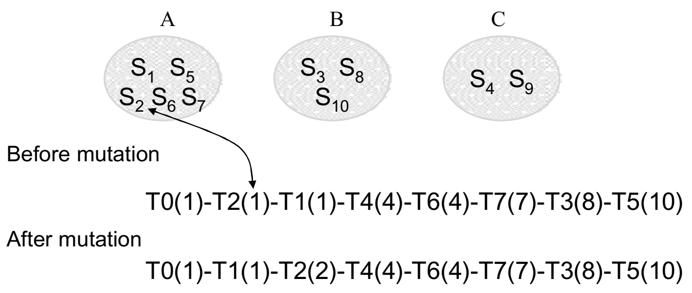

A
B
C
S1 S5 S2 S6 S7
S3 S8 S10
S4 S9
Before mutation
T0(1)-T2(1)-T1(1)-T4(4)-T6(4)-T7(7)-T3(8)-T5(10)
After mutation
T0(1)-T1(1)-T2(2)-T4(4)-T6(4)-T7(7)-T3(8)-T5(10)

Fig. 4. Illustration of replacing mutation operation.

$$F(I) = \begin{cases} \alpha \times F_{\cos t}(I) + \beta \times F_{\text{time}}(I), \\ \quad \text{if } F_{\cos t}(I) > 1 \text{ or } \mathrm{F}_{\text{time}}(I) > 1 \\ (F_{\cos t}(I)^{\beta} \times F_{\text{time}}(I)^{\alpha}, \\ \quad \text{otherwise} \end{cases}$$

3.3. *Genetic operators*

Genetic operations manipulate individuals in the current population and generate new individuals. We develop two genetic operators, crossover and mutation, for the scheduling problems.

3.3.1. *Crossover*

Crossovers are used to create new individuals on the current population by combining of 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 individuals [13]. As illustrated in Fig. 2, the crossover operator is implemented as follows: (1) Two parents are chosen at random in the current population. (2) Two random points are selected from the schedule order of the first parent. (3) All tasks between these two points are chosen as successive crossover points. (4) The locations of all tasks of the crossover points between *parent1* and *parent2* are exchanged. (5) Two new offspring are generated by combining task assignments taken from two parents. In this example, *offspring1* inherits task assignments of T_0, T_2, T_4 and T_6 from *parent1*, and the task assignments of the rest tasks are taken from *parent2*.

3.3.2. *Mutation*

In genetic algorithms, mutations occasionally occur in order to allow a certain children to obtain features that are not possessed by either parent. It helps a genetic algorithm to explore a new and better genetic material than previously considered. We have developed two types of mutation, namely *swapping mutation* and *replacing mutation*, in order to promote further exploration of the search space. The mutation operators are applied to the chosen individuals with a certain probability.

Swapping mutation aims to change the execution order of tasks in an individual that compete for a same time slot. It is implemented as follows: (1) A service in the individual is randomly selected. (2) The positions of two randomly selected independent tasks on the service are swapped. An example of swapping mutation is shown in Fig. 3. After the mutation, the time slot initially assigned to T_0 is occupied by T_1.

Replacing mutation aims to re-allocate an alternative service to a task in an individual. It is implemented as follows: (1) A task is randomly selected in the individual. (2) An alternative service which is capable of executing the task is randomly selected to replace the current task allocation.

An example of replacing mutation is shown in Fig. 4. Given the heterogeneous nature of execution environments required by workflow tasks, we classify processing services into groups. Each service group provides a

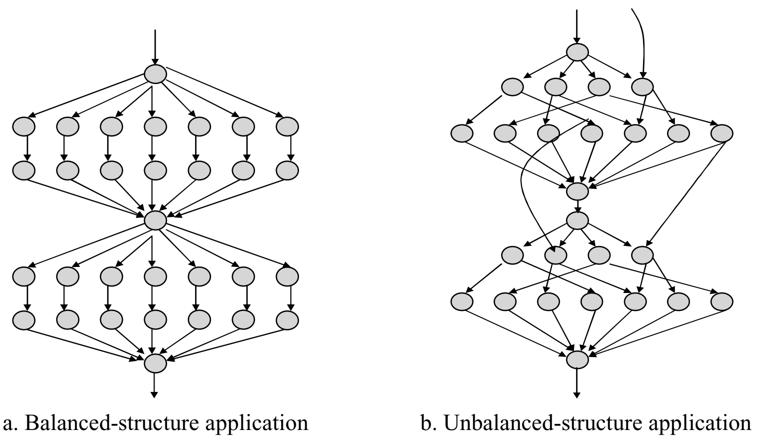

a. Balanced-structure application
b. Unbalanced-structure application

Fig. 5. Small portion of workflow applications.

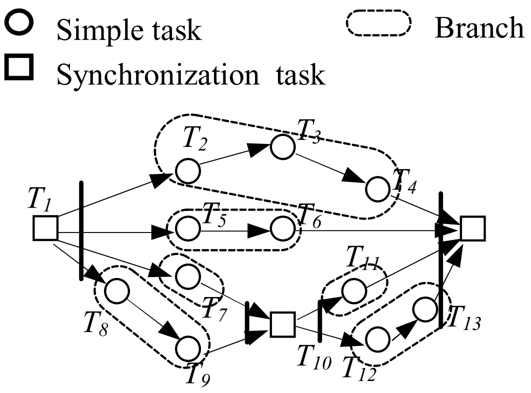


Fig. 6. Workflow task partitioning.

certain type of service that satisfies the execution condition of a task in the workflow. In the example, different tasks in the workflow require different types of services and all services are grouped together to support service type A, B, and C. For example, T_0, T_3 and T_4 require services of type A, B and C respectively. In the example, task T_2 is selected for mutation and T_2 is supported by services of type A. The mutation process randomly selects S_2 in the service group of type A and re-allocates it to T_2.

4. Experiments

4.1. Methodology

In order to evaluate the proposed approach, we implemented the algorithm described in Section 3 and compared it with a set of non-GA heuristics for two different types of workflow applications on a simulated Grid testbed. The details of the workflow applications, non-GA heuristics, simulation environment and experimental setting are presented in the following subsections.

4.1.1. Workflow applications

Given that different workflow applications may have different impact on the performance of the scheduling algorithms, we have developed a task graph generator which can automatically generate a workflow based on the specified workflow structures, the range of task workload and the I/O data. Since the execution requirements for tasks in scientific workflows are heterogeneous, we use the service type attribute 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 different sizes of workflows.

According to several Grid workflow projects [15, 17,32], workflow application structures can be categorized as either *balanced structure* or *unbalanced structure*. Examples of balanced structure are neuro-science workflows [34] and EMAN refinement workflows [15], while the examples of unbalanced structure are protein annotation workflows [4] and Montage workflows [17]. Figure 5 shows two workflow structures, *balanced-structure application* and *unbalanced-structure application*, used in our experiments. As shown in Fig. 5(a), the balanced-structure application consists of several parallel pipelines, which require the same types of services but process different data sets. As shown in Fig. 5(b), 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.

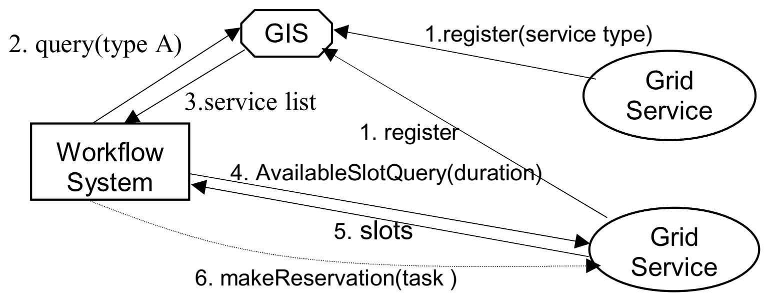


Fig. 7. Simulation environnent.

4.1.2. Non-GA heuristics

In order to evaluate the genetic algorithm (GA) we also implemented two other non-GA heuristics, namely *Greedy Cost – Time Distribution* (TD) and *Greedy Time - Cost Distribution* (CD). The CD approach is aimed at solving the budget constrained problem while the TD is designed to solve the deadline constrained problem.

Greedy Time-Cost Distribution (*CD*)

The CD heuristic distributes portions of the overall budget to each task in the workflow based on its average estimated execution cost. During the workflow execution, CD attempts to allocate a fastest service to each task among the services, which are able to complete the task execution within its planned budget. The actual costs of allocated tasks and their planned costs are also computed successively at runtime. If the aggregated actual cost is less than the aggregated planned cost, the scheduler uses the unspent aggregated budget to schedule the current task.

Greedy Cost-Time Distribution (*TD*)

The TD heuristic distributes the overall deadline over single workflow tasks. The deadline assignment is based on our previous work [31]. In order to produce an efficient schedule, TD partitions workflow tasks into *branches* and *synchronization tasks* as shown in Fig. 6. A synchronization task is a task with more than one parent task or child task, while a branch is a set of interdependent simple tasks that are executed sequentially between two synchronization tasks. Firstly sub-deadlines are assigned to task partitions. The overall deadline is divided over task partitions in proportion to their approximate transmission time and processing time. The cumulative assigned sub-deadlines of any independent path between two synchronization tasks must be same. For example, the deadline assigned to $\{T_8, T_9\}$ is the same as $\{T_7\}$ in Fig. 6. Similarly, sub-deadlines assigned to $\{T_2, T_3, T_4\}$, $\{T_5, T_6\}$, and $\{\{T_7\}, \{T_{10}\}, \{T_{12}, T_{13}\}\}$ are same. The sub-deadline of each task partition is then divided into their tasks based on its approximate execution time and transition time. At the runtime, a task is scheduled on a service, which is able to complete it within its assigned sub-deadline at the lowest cost.

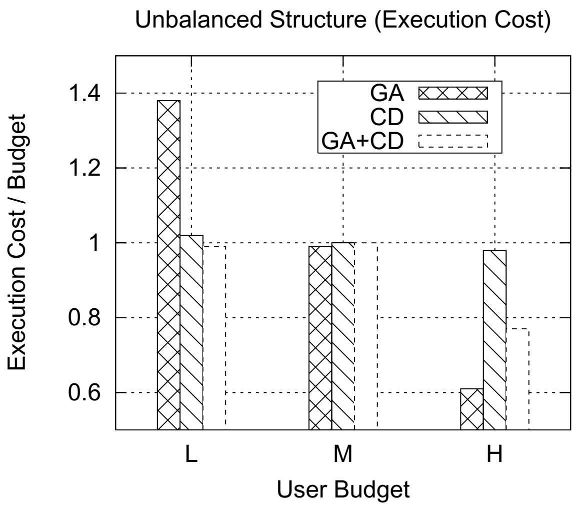
4.1.3. Simulation environment

We use GridSim [25] to simulate a Grid environment for our experiments. Figure 7 shows the simulation environment, in which simulated services are discovered by querying the GridSim Index Service (GIS). Every service is able to handle a free slot query, reservation request and commitment.

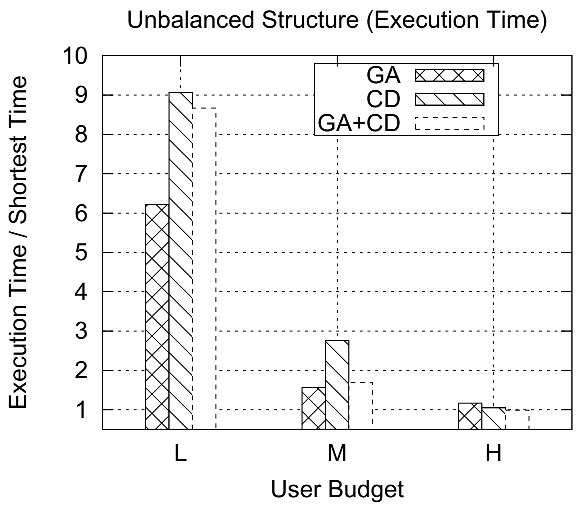
In our experiments, we simulated 15 types of services with various price rates, each of which was supported by 10 service providers with various processing capability. The topology of the system is such that all services are connected to one another, and the available network bandwidths between services are 100 Mbps, 200 Mbps, 512 Mbps and 1024 Mbps. The processing cost and transmission cost are inversely proportional to the processing time and transmission time respectively.

4.1.4. Experimental setting

In order to evaluate algorithms on reasonable budget and deadline constraints we also implemented a time optimization algorithm, *Heterogeneous-Earliest-Finish Time* (HEFT) [27], and a cost optimization algorithm, *Greedy Cost* (GC). The HEFT algorithm is a list scheduling algorithm which attempts to schedule interdependent tasks at minimum execution time on a heterogeneous environment. The GC approach is to minimize workflow execution cost by assigning tasks to services of lowest cost. The deadline and budget we used for the experiments are based on the results of these two algorithms. Let C_{GC} and C_{HEFT} be the total monetary cost produced by GC and HEFT re-



a. Execution cost of three budget constrained approaches.



b. Execution time of three budget constrained approaches.

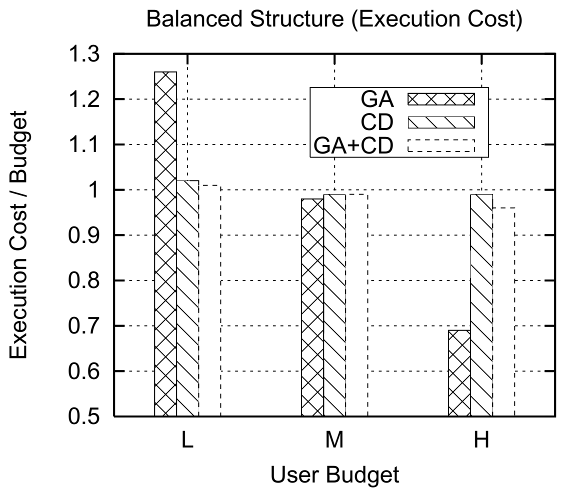
Fig. 8. Execution cost and time using three approaches for scheduling the unbalanced-structure application.

spectively, and T_{GC} and T_{HEFT} be their corresponding total execution time. Deadline D is defined by $D = T_{\mathrm{HEFT}} + k \times (T_{\mathrm{GC}} - T_{\mathrm{HEFT}})$ and budget B is defined $B = G_{\mathrm{GC}} + k \times (C_{\mathrm{HEFT}} - G_{\mathrm{GC}})$. The value of k varies between 0, 0.5 and 1 to evaluate the algorithm performance at tight/low, medium and high constraints.

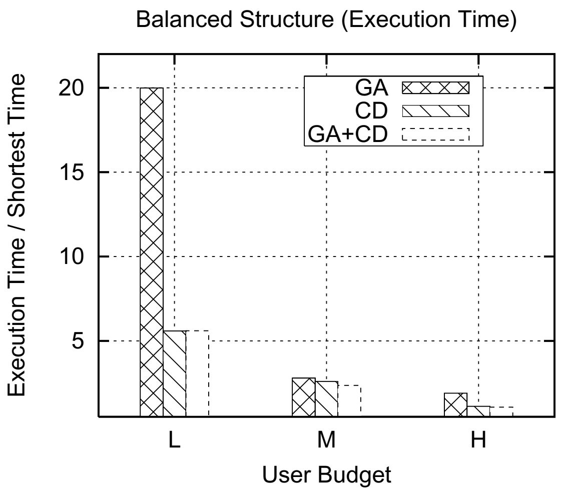
The following parameter settings are the default configuration used for producing results of the genetic algorithm: population size of 10, swapping mutation and replacing mutation probability of 0.5, a generation limit of 100.

4.2. Results

We compare the genetic algorithms with the CD and TD heuristics on the two workflow applications, balanced and unbalanced. We run the genetic algorithm starting with an initial population consisting of randomly generated solutions. We also investigate the af-



a. Execution cost of three budget constrained approaches.



b. Execution time of three budget constrained approaches.

Fig. 9. Execution cost and time using three approaches for scheduling the balanced-structure application.

fect of running the genetic algorithm by starting with an initial population consisting of a solution produced by one of the simple heuristics together with other randomly generated solutions. The results generated by the CD and TD heuristics are denoted as CD and TD respectively, and the results generated by the 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 CD and TD heuristics are denoted as GA+CD and GA+TD respectively.

In order to show the results more clearly, we normalize the execution time and cost. Let C_{value} and T_{value} be the execution time and the monetary cost generated by the algorithms in the experiments respectively. For the case of budget constrained problems, we normalize the execution cost by using C_{value}/B, and the execution time by using $T_{\text{value}}/T_{\text{HEFT}}$. After normalization, the values of the execution cost should be no greater than one, if the algorithms meet their budget constraints. Therefore, we can easily recognize whether the algorithms achieve the budget constraints. By using the normalized execution time value, we can also easily recognize whether the algorithms produce an optimal solution when the budget is high. In the same way, we also normalized execution time and the execution cost for the deadline constraint case by using T_{value}/D and $C_{\mathrm{value}}/C_{\mathrm{GC}}$ respectively.

4.2.1. Cost optimization within a set deadline

A comparison of the execution time and cost results of the three scheduling methods for scheduling the unbalanced-structure application and balanced-structure application with low, medium and high budget constraints respectively is shown in Figs 8 and 9. We can see that both GA and CD approaches cannot satisfy the low budget constraint, and GA produces the worst results. However, the results are improved if we combine GA and CD together by putting the solution produced by CD into the initial population of the GA. At the medium budget constraint, the GA performs better than CD for the unbalanced structure application, whereas CD performs better for the balanced structure application. This is because the decision of the task assignment for CD is based only on its local budget constraint and does not consider task dependencies. Tasks in the unbalanced-structure application are highly heterogeneous, have different workload and I/O data, and many are required to be executed in parallel. These parallel tasks are also required to run on various services with various price rates. Many tasks could be completed at earliest time using more expensive services based on their local budget, but its child tasks cannot start execution until other parallel tasks have been completed. Therefore, the schedule generated by CD is not very efficient for a complex unbalanced-structure application. This also shows that it is important to consider other parallel task dependencies when assigning a local budget to a task. For the balanced-structure application, parallel tasks are similar and hence obtain same local budgets which allow them to be completed at the same speed. Therefore, CD can perform better for the balanced-structure application than the unbalanced-structure application. However, its budget constraint distribution problem for the unbalanced-structure application can be released when the budget is very high. At the high budget value, CD performs better than the GA. Moreover, by combining the two approaches, GA+CD can achieve the same time optimization result as produced by the HEFT algorithm, but it can produce a solution with a lower cost.

4.2.2. Time optimization within a set budget

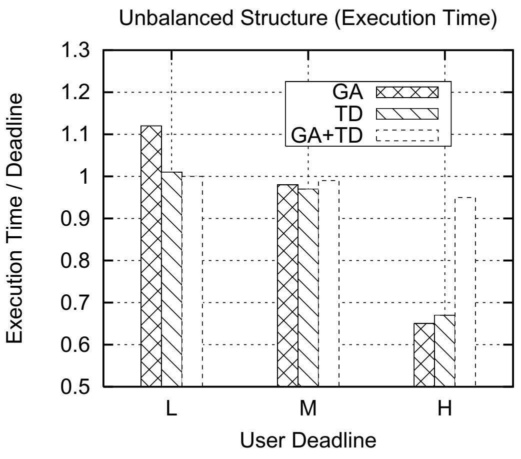
Figures 10 and 11 compare the execution time and cost of using three scheduling approaches for scheduling the unbalanced-structure application and balanced structure application with low, medium and high deadline constraints respectively. We can see that it is hard for both GA and TD to successfully meet the low deadline individually. As same as shown in the budget constraint case, GA+TD can improve the results. Unlike CD, TD performs better than GA for the unbalanced structure application as the deadline increases, since it distributes the overall deadline between tasks based on both task workload and parallel task dependencies. For the balanced- structure application, the results produced by GA and TD with a medium deadline are similar. At high deadline, TD performs slightly better than the GA, but the results are much improved for the unbalanced-structure application by using GA to continue search the better solution based on that of TD. With a high deadline, the execution costs of GA+TD are closed to the cheapest costs returned by the Greedy Cost approach, but it can produce faster solution for the unbalanced structure application.

4.2.3. Effect of the number of generations

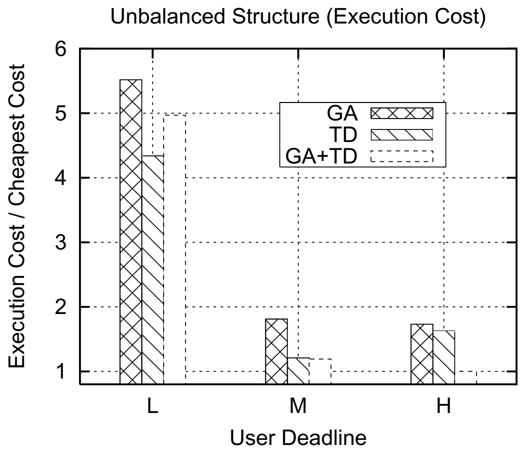
We also observe the performance of the GA when the number of generation cycles is altered. Figure 12(a) shows that the execution cost is significantly reduced to the specified budget as the number of generations is increased from 1 to 5. Consequently, the execution time shown in Fig. 12(b) increases during these generation cycles; this is because individuals which process slower are selected in order to decrease the execution cost. However, once the GA has found the individuals which are able to complete the execution within the budget, it starts to improve the performance, and execution time is decreased for successive generations.

5. Related work

Many heuristics have been investigated by several projects for scheduling workflows on Grids. The heuristics can be classified as either *task level* or *workflow level*. Task level heuristics make scheduling decisions based only on the information about a task or a set of independent tasks, while workflow level heuristics take into account the information of the entire workflow. *Min-Min*, *Max-Min* and *Sufferage* are three major task level heuristics employed for scheduling workflows on Grids. They have been used by



a. Execution time of three deadline constrained approaches.

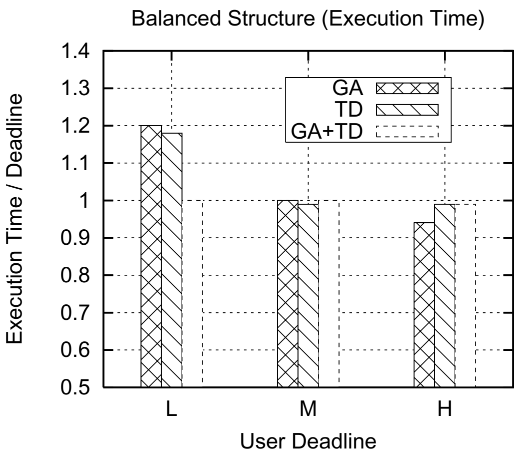


b. Execution cost of three deadline constrained approaches.

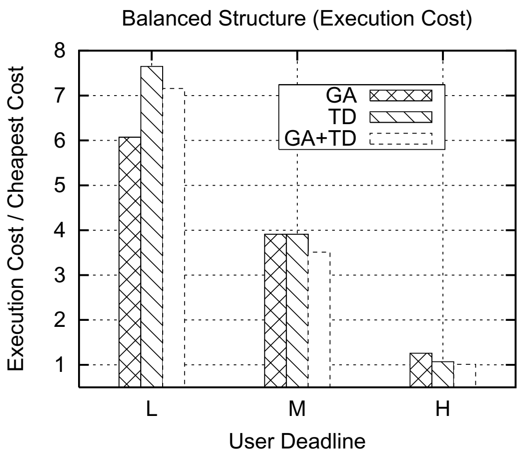
Fig. 10. Execution cost and time using three approaches for scheduling the unbalanced-structure application.

Mandal et al. [15] to schedule EMAN bio-imaging applications. Blythe et al. [3] developed a workflow level scheduling algorithm based on *Greedy Randomized Adaptive Search Procedure* (GRASP) [9] and compared it with Min-Min in compute- and data-intensive scenarios. Another two workflow level heuristics have been employed by the ASKALON project [22,32]. One is based on *Genetic Algorithms* and the other is a *Heterogeneous-Earliest-Finish-Time* (HEFT) algorithm [27]. Sakellariou and Zhao [23] developed a low-cost rescheduling policy. It intends to reduce the overhead produced by rescheduling by conducting rescheduling only when the delay of a task execution impacts on the entire workflow execution. However, these works only attempt to minimize workflow execution time and do not consider users' budget constraints.

Several works have been proposed to address scheduling problems based on users' budget constraints. Nimrod-G [5] schedules independent tasks for parameter-sweep applications to meet users' budget. A market-based workflow management system [11] locates an optimal bid based on the budget of the current task in the workflow. More recently, Tsiakkouri et al. [29] developed scheduling approaches, *LOSS* and *GAIN*, to adjust a schedule which is generated by a time optimized heuristic and a cost optimized heuristic to meet users' budget constraints respectively. In contrast, we focus on using genetic algorithms to solve the



a. Execution time of three deadline constrained approaches.



b. Execution cost of three deadline constrained approaches.

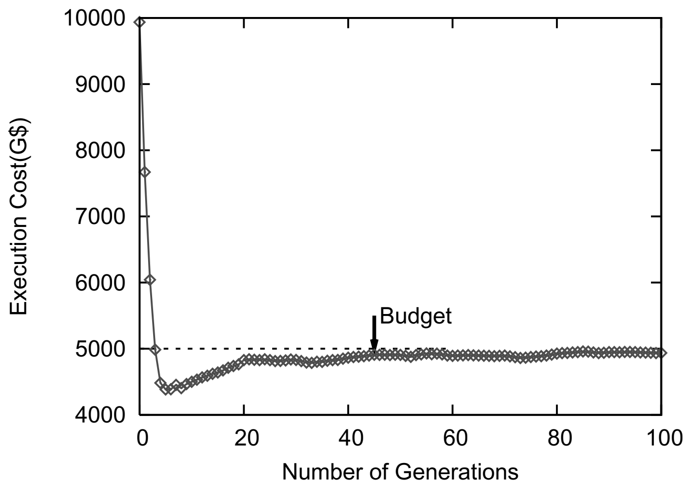
Fig. 11. Execution cost and time using three approaches for scheduling the balanced-structure application.

problems of scheduling inter-dependent tasks based on the budget and deadline of entire workflow.

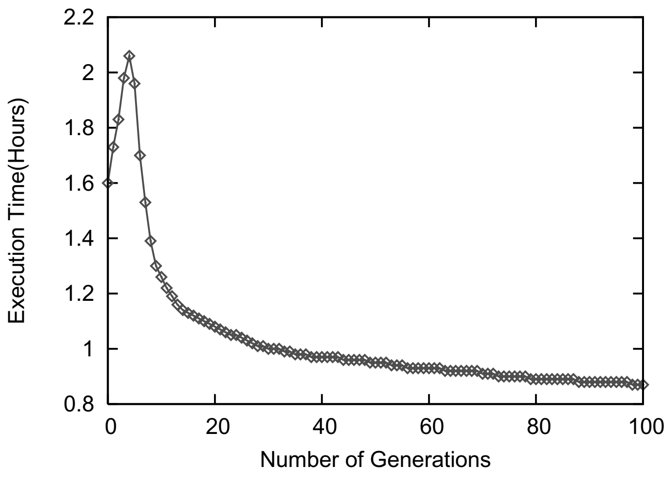
Using the genetic algorithm approach to schedule tasks in homogenous multiprocessor systems has been presented in many literature such as [13,33,35,36]. The proposed approach in this paper intends to introduce a new type of genetic algorithm for large heterogeneous environments for which the existing genetic operations algorithms cannot be directly applied.

6. Conclusion and future work

Utility Grids enable users to consume utility services transparently over a secure, shared, scalable and standard world-wide network environment. Users are required to pay for access to services based on their usage and the level of QoS required for this network environment to be commercially sustainable. Therefore, workflow execution cost must be considered during scheduling. In this paper, we have proposed a genetic algorithm approach for scheduling workflow applications by either minimizing the monetary cost while meeting users' deadline constraint, or minimizing the execution time while meeting users' budget constraints. Compared with most existing genetic algorithms, the proposed approach targets heterogeneous and reservation based service-oriented environments for solving budget and deadline constrained optimization problems.



a. Execution cost.



b. Execution time.

Fig. 12. Evolution of execution time and cost during 100 generations.

We evaluate our approach by comparing it with two other heuristics, on both balanced and unbalanced workflow structures. The results show that the genetic algorithm is better for handling a complex workflow structure. The genetic algorithm can also significantly improve the results returned by other heuristics by employing these heuristic results as individuals in its initial population.

We will be further enhancing our scheduling algorithm by supporting different service negotiation models and dynamic data-driven workflow models. We will also study how the genetic algorithm approach can be applied for scheduling workflows based on other QoS constraints such as reliability and security.

Acknowledgments

We would like to thank Hussein Gibbins, Krishna Nadiminti and Chee Shin Yeo for their comments on this paper. We thank Anthony Sulistio for his support with the use of GridSim. This work is partially supported through an Australian Research Council (ARC) Discovery Project grant.

References

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

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

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

[4] A. O'Brien, S. Newhouse and J. Darlington, *Mapping of Scientific Workflow within the e-Protein project to Distributed Resources*, In UK e-Science All Hands Meeting, Nottingham, UK, Sep. 2004.

[5] R. Buyya, J. Giddy and D. Abramson, *An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications*, In 2nd Workshop on Active Middleware Services (AMS 2000), Kluwer Academic Press, August 1, 2000, Pittsburgh, USA.

[6] E. Deelman et al., Mapping abstract complex workflows onto grid environments, *Journal of Grid Computing* **1** (2003), 25–39.

[7] T. Eilam et al., A utility computing framework to develop utility systems, *IBM System Journal* **43**(1) (2004), 97–120.

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

[9] T.A. Feo and M.G.C. Resende, Greedy randomized adaptive search procedures, *Journal of Global Optimization* **6** (1995), 109–133.

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

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

[12] D. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[13] E.S.H. Hou, N. Ansari and H. Ren, A genetic algorithm for multiprocessor scheduling, *IEEE Transactions on Parallel and Distributed Systems* **5**(2) (February 1994), 113–120.

[14] B. Ludäscher et al., *Scientific Workflow Management and the KEPLER System*, Concurrency and Computation: Practice & Experience, Special Issue on Scientific Workflows, to appear, 2005.

[15] A. Mandal et al., *Scheduling Strategies for Mapping Application Workflows onto the Grid*, IEEE International Symposium on High Performance Distributed Computing (HPDC 2005), 2005.

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

[17] A. Mandal et al., *Scheduling Strategies for Mapping Application Workflows onto the Grid*, IEEE International Symposium on High Performance Distributed Computing (HPDC 2005), 2005.

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

[19] F. Neubauer, A. Hoheisel and J. Geiler, Workflow-based grid applications, *Future Generation Computer Systems* **22** (2006), 6–15.

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

[21] T. Oinn et al., Taverna: a tool for the composition and enactment of bioinformatics workflows, *Bioinformatics* **20**(17) (2004), 3045–3054, Oxford University Press, London, UK.

[22] R. Prodan and T. Fahringer, *Dynamic Scheduling of Scientific Workflow Applications on the Grid using a Modular Optimisation Tool: A Case Study*, In 20th Symposium of Applied Computing (SAC 2005), Santa Fe, New Mexico, USA, March 2005. ACM Press.

[23] R. Sakellariou and H. Zhao. A low-cost rescheduling policy for efficient mapping of workflows on grid systems, *Scientific Programming* **12**(4) (December 2004), 253–262.

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

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

[26] I. Taylor, M. Shields and I. Wang, *Resource Management of Triana P2P Services*, Grid Resources Management, Kluwer, Netherlands, June 2003.

[27] 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) (March 2002), 260–274.

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

[29] E. Tsiakkouri et al., Scheduling Workflows with Budget Constraints, in: *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.

[30] J. Yu and R. Buyya, A taxonomy of workflow management systems for grid computing, *Journal of Grid Computing, Springer* **3**(3–4) (Sept. 2005), 171–200, Spring Science+Business Media B.V., New York, USA,

[31] J. Yu, R. Buyya and C.K. Tham, *A Cost-based Scheduling of Scientific Workflow Applications on Utility Grids*, In 1st IEEE International Conference on e-Science and Grid Computing, Melbourne, Australia, Dec. 5–8, 2005.

[32] M. Wieczorek, R. Prodan and T. Fahringer, Scheduling of scientific workflows in the ASKALON grid environment, Special Issues on scientific workflows, *ACM SIDMOD Record* **34**(3) 56–62, ACM Press, 2005.

[33] A.S. Wu et al., An incremental genetic algorithm approach to multiprocessor scheduling, *IEEE Transactions on Parallel and Distributed Systems* **15**(9) (September 2004), 824–834.

[34] Y. Zhao et al., *Grid Middleware Services for Virtual Data Discovery, Composition, and Integration*, In 2nd Workshop on Middleware for Grid Computing, October 18, 2004, Toronto, Ontario, Canada.

[35] 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) (August 1999), 795–812.

[36] A.Y. Zomaya and Y.H. Teh, The, Observations on using genetic algorithms for dynamic load-balancing, *IEEE Transactions on Parallel and Distributed Systems* **12**(9) (September 2001), 899–911.