A Budget Constrained Scheduling of Workflow Applications on Utility Grids using Genetic Algorithms

Jia Yu and Rajkumar Buyya

Grid Computing and **D**istributed **S**ystems (GRIDS) Laboratory
Dept. of Computer Science and Software Engineering
The University of Melbourne, VIC 3010 Australia
{jiayu, raj}@csse.unimelb.edu.au

***Abstract*—Over the last few years, Grid technologies have progressed towards a service-oriented paradigm that enables a new way of service provisioning based on utility computing models. Users consume these services based on their QoS (Quality of Service) requirements. In such "pay-per-use" Grids, workflow execution cost must be considered during scheduling based on users' QoS constraints. In this paper, we propose a budget constraint based scheduling, which minimizes execution time while meeting a specified budget for delivering results. A new type of genetic algorithm is developed to solve the scheduling optimization problem and we test the scheduling algorithm in a simulated Grid testbed.**

I. INTRODUCTION

Utility computing [29] has emerged as a new service provisioning model [9] 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][12], that creates an infrastructure for enabling users to consume services transparently over a secure, shared, scalable, sustainable and standard world-wide network environment.

Table I 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 workflow management systems [8][10][16][18][20][22][28][31] 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.

Table I. 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

Given this motivation, we focus on developing workflow scheduling based on user's QoS constraints such as deadline and budget. In general, processing time and execution cost are two typical QoS constraints for the workflow execution on "pay-per-use" services. We have presented a cost based scheduling heuristic in [32], which minimizes workflow execution cost within a certain deadline. In this paper, we develop a genetic algorithm based scheduling heuristic to minimize execution time while meeting user's budget constraint.

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

The remainder of the paper is organized as follows. We introduce the problem overview in Section II including problem definition, general genetic algorithms and performance estimation approaches. Our proposed genetic-based workflow scheduling algorithm is presented in Section III. Experimental details and simulation results are presented in Section IV. We introduce related work and compare them with our work in Section V. Finally, we conclude the paper with directions for further work in Section VI.

II. PROBLEM OVERVIEW

A. Problem Description

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

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

The scheduling problem is to map every T_i onto a suitable S_i^j to minimize the execution time of the workflow and complete it within the budget B.

B. Genetic Algorithms

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. In Grid environments, scheduling decision must be produced in the shortest time possible, because there are many users compute for resources and desired time slots could be occupied by others at any time.

Genetic algorithms (GAs)[14] 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 consists 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 step 2 and 3 until the algorithm converges.

C. 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 reservation-enabled resource 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 [21], empirical and historical data [19][25]) to predict task execution time on every discovered resource service. Duration T_{reserv} required to be reserved on a resource service can be calculated by $T_{reserv} = T_{estimat} / \alpha$, where $T_{estimat}$ the execution time is generated by a prediction approach and α is the corresponding accuracy rate and $\alpha \le 1$.

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

III. PROPOSED APPROACH

A. Problem Encoding

For the workflow scheduling problem, a feasible solution is required to meet following conditions:

- A task can only be started after all its predecessors have completed.
- Every task appears once and only once in the schedule.
- 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, evolving 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 allocation for each task and the order of tasks allocated on each service. After crossover and mutations, a time slot assignment method is deployed to transfer an operation string to a feasible schedule.

A simple one-dimensional string shown in Fig. 1 is not suitable for representing a workflow schedule. In a workflow, the execution order of interdependent tasks is controlled by their dependencies, e.g. a task is always executed after its immediate parent tasks. However, many independent tasks, e.g. 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 independent tasks within the workflow may impact the performance of workflow execution significantly. Therefore, encoded strings are required to show the order of task assignments on each service. The simple string method shown Fig. 1 sets a constant position for each task and then assigns services to tasks. This only describes which service is allocated to each task, but ignores their execution order. For example, in individual 1, both T_3 and T_4 are assigned to S_3, but it is not clear which one is executed first if they are ready to be executed at the same time and only one time slot is available on S_3.

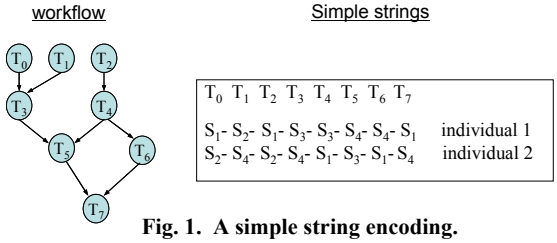


Fig. 1. A simple string encoding.

In order to solve this problem, we use a 2D string to represent a schedule as illustrated in Fig. 2. 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.

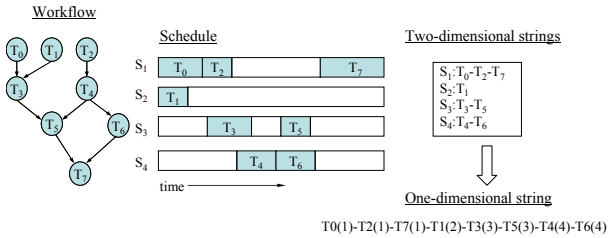


Fig. 2. Illustration of problem encoding.

B. Initial Population

Each individual of the initial population is generated through a random heuristic. For each individual, the task to be scheduled is determined by the following rules:

1. Choose a *ready* task T_i, that has already had all of its parent tasks scheduled.
2. Compute the ready time of T_i by: $readyTime(T_i)= \max_{T_j \in P_i} endTime(T_j)$, where P_i is the set of parent tasks of T_i.
3. Randomly select a service S_i , from those that are able to run T_i.
4. Compute the transmission time, $transTime(T_i)$, of I/O data transfer between services that execute parent tasks of T_i and S_i.
5. Query available time slots after $startTime(T_i)$ on S_i, where $startTime(T_i) = readyTime(T_i)+transTime(T_i)$.
6. Allocate a free time slot for T_i at random.

C. 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 execution time while still meeting the user's specified budget, the fitness function separates evaluation into two parts: *cost-fitness* and *time-fitness*.

The cost-fitness component encourages the formation of the solutions that achieve the budget constraint. The cost fitness function of an individual I is defined by:

$$F_{cost}(I) = \frac{c(I)}{B},$$

where $c(I)$ is the sum of the task execution cost and data transmission cost of I and $c(I)= \sum_{T_i \in I} c_i^k$, $1 \le k \le m_i$, and B is the budget of the workflow.

The time-fitness component is designed to encourage the genetic algorithm to choose individuals with earliest completion time in the current population. The time fitness function is defined by:

$$F_{time}(I) = \frac{t(I)}{maxTime},$$

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

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

$$F(I) = \begin{cases} F_{cost}(I)+1, & if\ F_{cost}(I) > 1 \\ F_{time}(I), & \text{otherwise} \end{cases}$$

D. Genetic operators

1) Selection

After the fitness evaluation process, the new individuals are compared with the previous generation. All individuals from both generations are ranked based on their fitness values. An individual with a small value of fitness is better than the one with a large value of fitness. The fittest individuals are retained in the population as successive generations evolve.

2) 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 [15]. As illustrated in Fig.3, 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*.

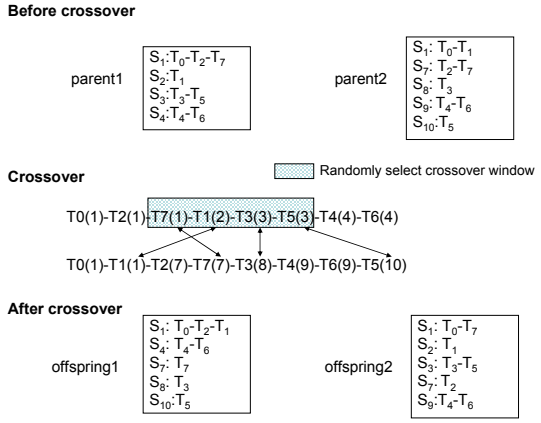


Fig. 3. Illustration of crossover operation.

3) 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. 4. After the mutation, the time slot initially assigned to T_0 is occupied by T_1.

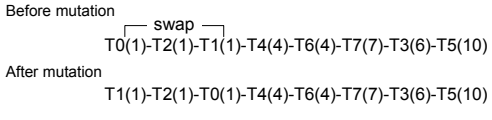


Fig. 4. Illustration of swapping mutation operation.

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. 5. Given the heterogeneous nature of execution environments required by workflow tasks, we classify processing services into groups. Each service group provides a certain type of service that satisfies the execution condition of a task in the workflow. In the example, all services are grouped together to support service type A, B, and C and different tasks in the workflow require different types of services. 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.

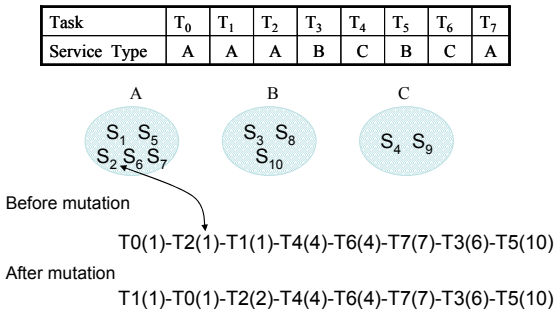


Fig. 5. Illustration of replacing mutation operation.

E. Time Slot Assignment

The string representing offspring produced by crossover and mutation operators are not a real schedule, since we ignore the time frames during the operations. We develop a time slot assignment process in order to transfer an offspring string to a feasible solution. As illustrated in Fig. 6, it queries available time slots from services based on the information of resource allocations and task execution orders in the offspring, and assign a time slot to each task. The produced schedule satisfies the conditions of a feasible solution defined in the previous sub-section. Algorithm 1 shows the pseudo-code of the reorder algorithm.

Algorithm 1. time slot assignment algorithm
Input: *A workflow graph* Ω *, two-dimensional strings* ***2D***
Output: *A feasible schedule*

```
ready ← get first level tasks in the workflow Ω
while ready ≠ Φ repeat
    for all S_i ∈ 2D do
        T← remove first task allocated on S_i
        if T ∈ ready then
            compute the ready time of T
            query and assign a free slot on S_i for T
            remove T from ready
            CT ← get ready child tasks of T
            for each ct_i ∈ CT do
                if ct_i ∉ ready then
                    ready ← put ct_i
```

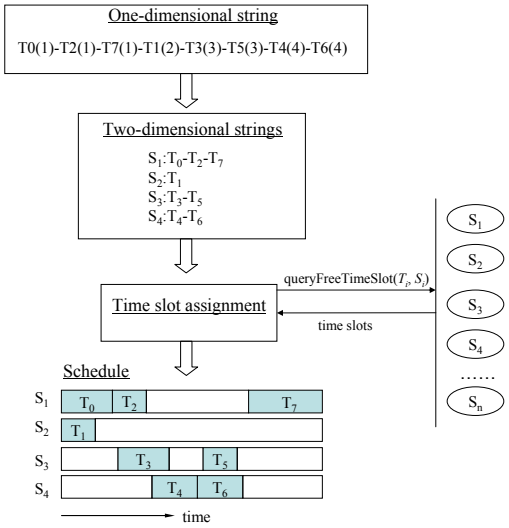


Fig. 6. Illustration of problem decoding.

F. Schedule Refinement

Although the solution generated by the genetic operations and the time slot assignment process is feasible, it may not be very efficient. Fig. 7 shows a sample of the time assignments of a schedule. Note that the end time of T_4 is significantly larger than that of T_5. Scheduling T_3 and T_5 on faster services does not contribute to the entire workflow execution, since T_6 cannot start until T_4 is completed. However, it may incur unnecessary execution cost, since faster services charge higher prices. Therefore, we develop a refinement method to refine the schedule generated by genetic operations. Instead of waiting for other paths to be completed, a path capable of being completed earlier is rescheduled on slower but cheaper services through a schedule refinement process. Applying a refinement process after the genetic operations may help the genetic algorithm to converge faster, especially when the budget is very low. However, similar to mutation operations, it also disrupts the genetic algorithm evolution. Therefore, the frequency of refinement occurrence should be controlled by the refinement rate whose value is determined experimentally.

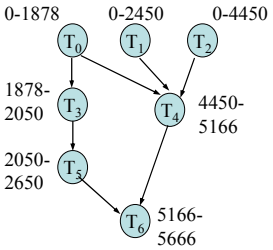
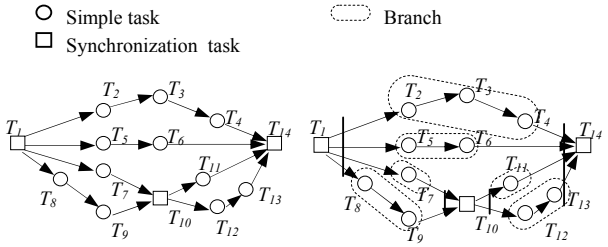


Fig. 7. The time assignments of workflow tasks. The number attached to each task is the time slot of the form of [start time]-[end time]. Network transmission time is ignored in this example, so the start time of a task is equal to the ready time of the task.

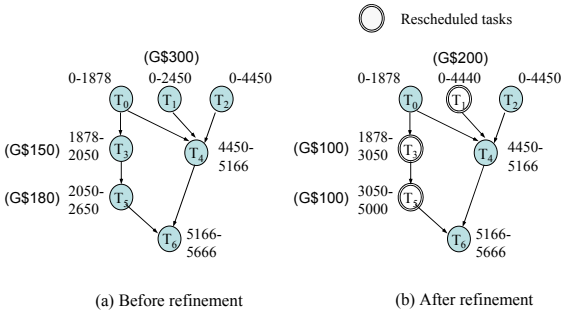
In order to refine a workflow schedule, we group tasks in a workflow into branches and synchronization tasks as shown in Fig. 8. A synchronization task is a task which has more than one child task or parent task, whereas a branch consists of a set of interdependent simple tasks that are executed sequentially between two synchronization tasks. The refinement process goes through all branches in the workflow. It reschedules tasks of a branch whose end time is much less than the ready time of its child synchronization task. The ready time of a synchronization task is the maximum end time of its parent branches.



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

Fig. 8. Workflow task partition.

The refinement process is used to solve the scheduling optimization problem of branch tasks. A new schedule generated by the refinement process should be the optimal schedule that minimizes the execution cost while completing the branch execution by the time its child synchronization task starts. For a branch with only one task, the optimal decision is simple. The optimal service is the cheapest service that can process the task on time. For a branch with multiple tasks, we model its scheduling decision problem as a Markov Decision Process (MDP) [27]. We set the ready time of its child synchronization task as the deadline of the branch. The details of the MDP definition can be found in [32]. We use value iteration, a standard dynamic programming method, to compute optimal policy for each MDP state and thus obtain the optimal schedule of the branch. Fig. 9 shows the time and cost of task assignments before and after refinement.



(a) Before refinement (b) After refinement

Fig. 9. Time and cost of task assignments.

IV. Performance Evaluation

We use GridSim [6][26] to simulate a Grid environment for our experiments. Fig. 10 shows the simulation environment in which simulated services are discovered by querying the GridSim Index Service (GIS) and every service is able to handle a free slot query, reservation request and commitment.

We compare our proposed scheduling algorithm denoted as *Genetic Algorithm* (GA) with a scheduling approach

derived from existing market based workflow scheduling [5] [13] denoted as *Greedy Time* (GT). The greedy time approach assigns a planed budget to each task in the workflow based on the average estimated execution costs of tasks and the total budget of the workflow. 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 current task. During the workflow execution, the greedy time approach attempts to allocate a fastest service to each task among the services, which are able to complete the task execution within its planned budget.

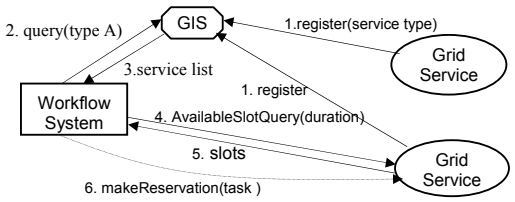


Fig. 10. Simulation environment.

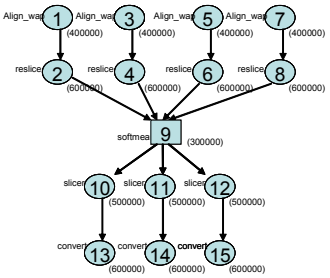
We simulate two common workflow structures in scientific workflow applications for our experiments: parallel and hybrid. A parallel application (see Fig. 11a) requires multiple pipelines to be executed in parallel. A pipeline executes a number of tasks in a single sequential order. For example, in Fig. 11a, there are 4 pipelines (1-2, 3-4, 5-6 and 7-8) before task 9. A hybrid structure application (see Fig. 11b) is a complex combination of parallel and sequential execution. In our experiments, we used a neuro-science workflow [36] for our parallel application and a protein annotation workflow [4] developed by London e-Science Centre for our hybrid workflow structure application.

Since the execution requirements for tasks in scientific workflows are heterogeneous, we use the service type attribute to represent the different types of services. Every task in our experimental workflow applications requires a certain type of service. For example, task 1, 3, 5, 7 in a parallel application require service type *Align_wap* and task 2, 4, 6 and 8 require *reslice.* In the simulation, we use MI (million instructions) to represent the length of tasks and use MIPS (Million Instructions per Second) to represent the processing capability of services. We simulate 15 types of services, each supported by 10 service providers with various processing capability. The values of MIPS for services range from 100 to 5000 and the value of MI for each task is indicated in brackets next to the task in Fig. 11.

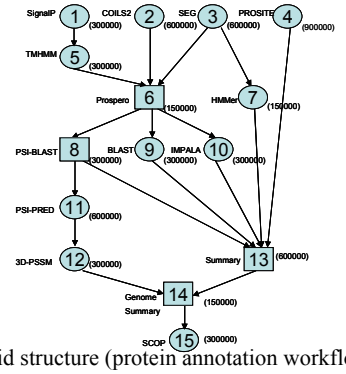
In our experiments, every task in the workflows generates output data required by its child tasks as inputs. The data needs to be staged out from the task processing node and staged into the processing node of its child tasks. The I/O data of the workflows ranges from 10MB to 1024 MB. The available network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

For our experiments, the cost that a user needs to pay for a workflow execution comprises of two parts: processing cost and data transmission cost. Table II shows an example of processing cost, while Table III 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.

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



a. Parallel application (fMRI workflow [36])



b. Hybrid structure (protein annotation workflow [4])

Fig. 11. Workflow applications. The label on the left of a task denotes the required service type. The number in brackets represents the length of the task in MI.

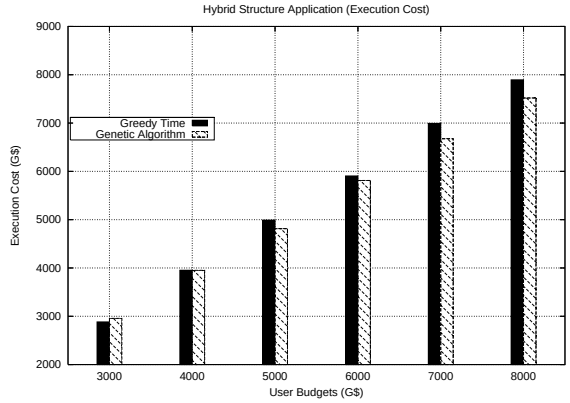
Table II. Service speed and corresponding price for executing a task.

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

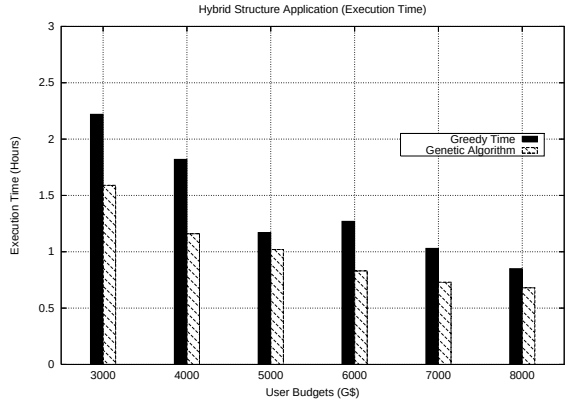
Table III. Transmission bandwidth and corresponding price.

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

The following parameter settings are the default configuration for producing results of the genetic algorithm: population size of 10, swapping mutation and replacing mutation probability of 0.5, a generation limit of 30, refinement probability of 0.5. Since the genetic algorithm is a stochastic search algorithm, each of the experiments was repeated ten times and average values are used to report the results.



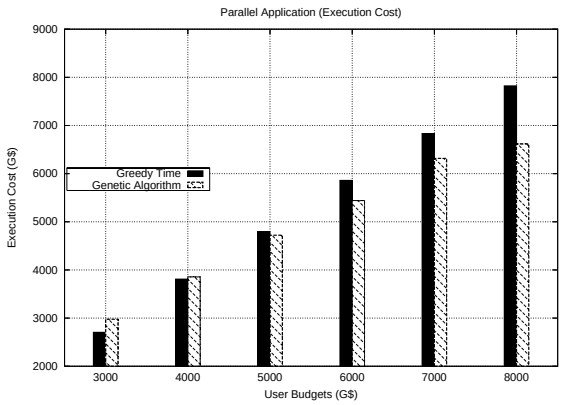
a . Execution cost of two approaches.



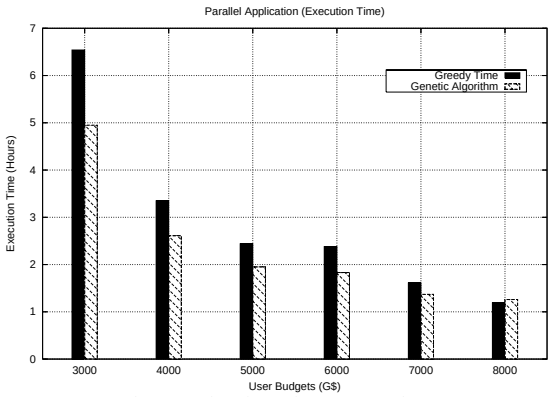
b. Execution time of two approaches.

Fig. 12. Execution time and cost using two approaches for scheduling the hybrid structure application.

Fig. 12 and Fig. 13 compare the execution time and cost of using the GA and the GT for scheduling parallel and hybrid structure applications with budget G$3000, G$4000, G$5000, G$6000, G$7000 and G$8000 respectively. It can be seen that the GT takes much longer to complete even though it incurs a higher execution cost. This is because the decision making of the GT based only on the information of the current task. It may produce the best schedule for the current task but it could consequently reduce the entire workflow performance. However, as the user's budget increases, the results of the two approaches are closer. Compared these two application structures, the performance of the parallel application produced by the greedy time is better than that of the hybrid structure application.

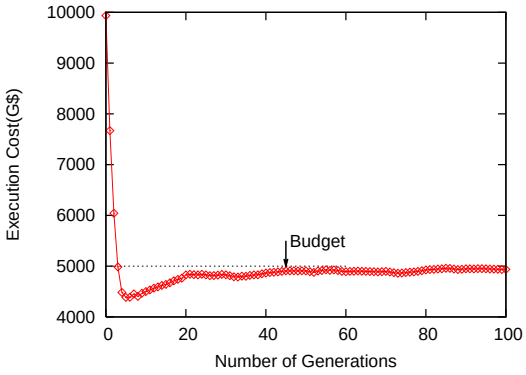


a. Execution cost of two approaches.

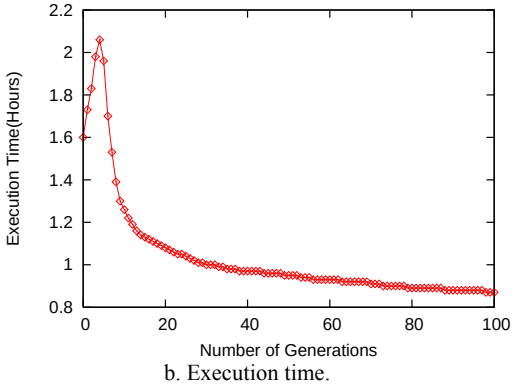


b. Execution time of two approaches.

Fig. 13. Execution time and cost using two approaches for scheduling the parallel application.



a. Execution cost.

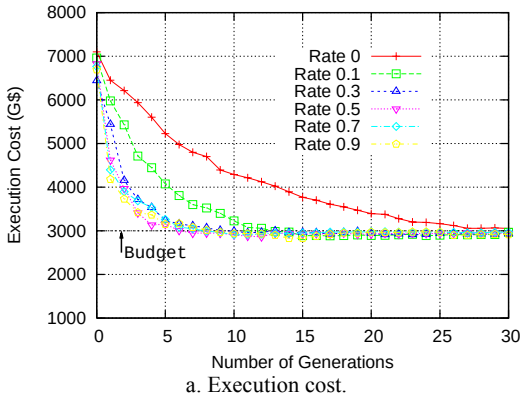


b. Execution time.

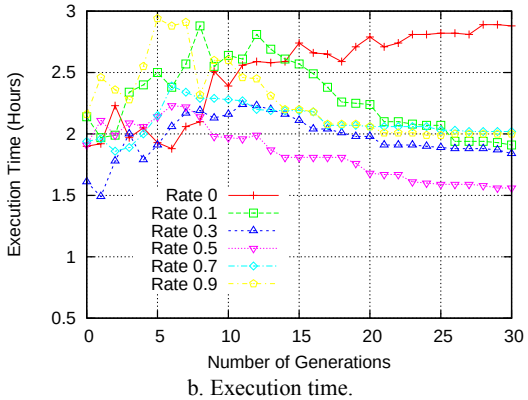
Fig. 14. Evolution of execution time and cost during 100 generations.

We observe the performance of the GA when the number of generation cycles is altered. Fig. 14a shows that the execution cost is significantly reduced to the specified budget as the number of generations is increased from 1 to 5. Consequently, as shown in Fig. 14b the execution time increases during these generation cycles; this is because individuals which take longer to process are selected in order to reduce 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 reduced for successive generations.

Fig. 15 shows the results generated by various refinement rates for scheduling the hybrid structure application when the budget is G$3000. It is observed from Fig. 15a that the GA cannot meet the budget within 30 generation cycles without the refinement process. We also observe from Fig. 15b that there is no significant performance improvement, when the rate is relatively low, i.e. 0.1 and 0.3 in this case, or relatively high, i.e. 0.7 and 0.9. This is because the GA converges slower as less of the refinement process is involved. The refinement process helps the GA evolve faster from high cost solutions to the solutions that meet budget constraints, since it replaces the higher cost task assignments of the selected individuals with cheaper task assignments. However, overly applying the refinement process can cause the parents to lose task assignments with shorter execution time, and thus result in the children with longer execution time.

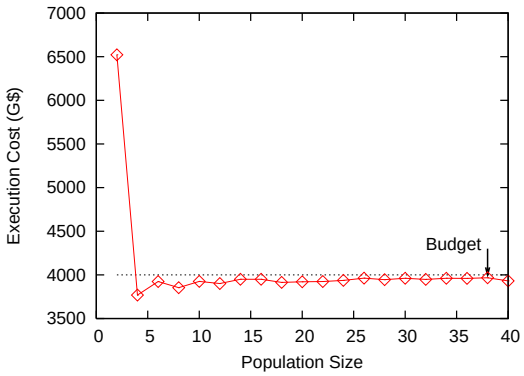


a. Execution cost.

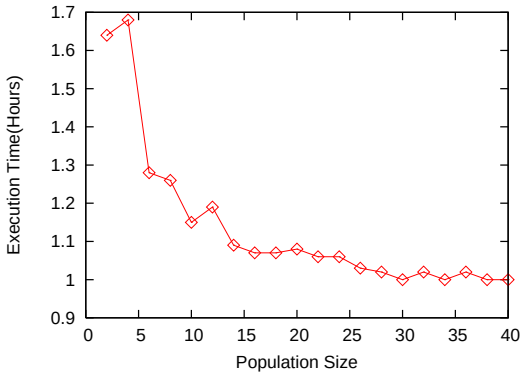


b. Execution time.

Fig.15. Evolution of execution time and cost in response to different refinement rate when budget is G$3000.



a. Execution cost vs. population size.



b. Execution time vs. population size.

Fig. 16. Execution cost and time for sizes of the population ranged from 2 to 40 when user's budget is G$4000.

Fig. 16 shows execution cost and time for sizes of the population ranging from 2 to 40. We observe from Fig. 16a that the solutions produced by the GA cannot even meet the specified budget when the size of population is very small. Once the population size is greater than 5, it is observed that increasing the population size does not affect execution cost significantly. However, large populations give the GA more opportunity to find faster solutions for the same execution costs.

V. 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 Mandal et al [17] to schedule EMAN bio-imaging applications. Blythe et al [3] developed a workflow level scheduling algorithm based on *Greedy Randomized Adaptive Search Procedure* (GRASP) [11] and compared it with Min-Min in compute- and data-intensive scenarios. Another two workflow level heuristics have been employed by the ASKALON project [23][34]. One is based on *Genetic Algorithms* and the other is a *Heterogeneous-Earliest-Finish-Time* (HEFT) algorithm. Sakellariou and

Zhao [24] 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 [13] locates an optimal bid based on the budget of the current task in the workflow. More recently, Tsiakkouri et al [30] 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. A time optimized heuristic attempts to minimize execution time while a cost optimization attempts to minimize execution cost. In contrast, we focus on using genetic algorithms to solve the problems of scheduling inter-dependent tasks based on the budget of entire workflow.

Using the genetic algorithm approach to schedule tasks in homogenous multiprocessor systems has been presented in many literature such as [15][35][37][38]. 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.

VI. Conclusion And Future Work

Utility Grids enable users to consume utility services transparently over a secure, shared, scalable and standard world-wide network environment. Users are required to pay for access 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 budget constraint based workflow scheduling approach that minimizes the execution time while meeting a specified budget. A new type of genetic algorithm has also been developed, as the crossover and mutation operations of existing genetic algorithms focused on homogenous and non reservation-enabled multiprocessor systems and therefore cannot be applied to the problem directly. The fitness function is developed to encourage the formation of the solutions to achieve the budget constraint and time minimization. We have also presented a workflow refinement approach using Markov decision processes to make the genetic algorithm converge faster when the budget is very low.

We will be further enhancing our scheduling algorithm by supporting different service negotiation models and runtime rescheduling, along with duplication of critical tasks to meet users QoS requirements even under failures. We will also study how the GA 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 and Krishna Nadiminti 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] R. Buyya and M. Murshed, "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing" *Concurrency and Computation: Practice and Experience,* 14(13-15):1175-1220, Wiley Press, USA, 2002.

[7] 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, IEEE CS Press, Los Alamitos, CA, USA, April 2004.

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

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

[10] 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.

[11] T. A. Feo and M. G. C. Resende, Greedy Randomized Adaptive Search Procedures, *Journal of Global Optimization*, 6:109-133, 1995.

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

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

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

[15] 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, February 1994.

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

[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] 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.

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

[20] F. Neubauer, A. Hoheisel and J. Geiler, "Workflow-based Grid Applications", *Future Generation Computer Systems*, 22:6-15, 2006.

[21] 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.

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

[23] 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.

[24] R. Sakellariou and H. Zhao. "A Low-Cost Rescheduling Policy for Efficient Mapping of Workflows on Grid Systems". Scientific Programming, 12(4), pages 253-262, December 2004.

[25] 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.

[26] 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.

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

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

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

[30] 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, pages 347-357 .

[31] 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.

[32] 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.

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

[34] 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.

[35] 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.

[36] 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.

[37] 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, August 1999.

[38] 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):899-911, September 2001.