

Scheduling Workflows with Budget Constraints^{*}

Eleni Tsiakkouri¹, Rizos Sakellariou², Henan Zhao², and Marios Dikaiakos¹

¹ Department of Computer Science, University of Cyprus, Cyprus

² School of Computer Science, University of Manchester, U.K.

Abstract. Grids are emerging as a promising solution for resource and computation demanding applications. However, the heterogeneity of resources in Grid computing, complicates resource management and scheduling of applications. In addition, the commercialization of the Grid requires policies that can take into account user requirements, and budget considerations in particular. This paper considers a basic model for workflow applications modelled as Directed Acyclic Graphs (DAGs) and investigates heuristics that allow to schedule the nodes of the DAG (or tasks of a workflow) onto resources in a way that satisfies a budget constraint and is still optimized for overall time. Two different approaches are implemented, evaluated and presented using four different types of basic DAGs.

1 Introduction

In the context of Grid computing, a wide range of applications can be represented as workflows many of which can be modelled as Directed Acyclic Graphs (DAGs) [7, 9, 1, 5]. In this model, each node in the DAG represents an executable task (it could be an application component of the workflow). Each directed edge represents a precedence constraint between two tasks (data or control dependence). DAGs represent a model that helps build a schedule of the tasks onto resources in a way that precedence constraints are respected and the schedule is optimized. Virtually all existing work in the literature [6, 1, 8] aims to minimize the total execution time (length or makespan) of the schedule.

Although the minimization of an application's execution time might be an important user requirement, managing a Grid environment is a more complex task which may require policies that strike a balance between different (and often conflicting) requirements of users and resources. Existing Grid resource management systems are mainly driven by system-centric policies, which aim to optimize system-wide metrics of performance. However, it is envisaged that future fully deployed Grid environments will need to guarantee a certain level of service and employ user-centric policies driven by economic principles [2]. Of particular interest will be the resource access cost, since different resources, belonging to different organisations, may have different policies for charging.

^{*} This work was supported by the CoreGRID European Network of Excellence, part of the European Commission's IST programme #004265

Clearly, users would like to pay a price which is commensurate to the budget they have available.

There has been little work examining issues related to budget constraints in a Grid context. The most relevant work is available in [3, 4], where it is demonstrated, through Grid simulation, how a scheduling algorithm can allocate jobs to machines at the same time satisfying constraints of Deadline and Budget. In this simulation, each job is considered to be a set of independent Gridlets (objects that contain all the information related to a job and its execution management details such as job length in million instructions, disk I/O operations, input and output file sizes and the job originator) [3]. Workflow types of applications, where jobs have precedence constraints, are not considered.

In this paper, we consider workflow applications that are modelled as DAGs. Instead of focussing only on makespan optimisation, as most existing studies have done [6, 8, 1], we also take into account a budget constraint. Each job, when running on a machine, costs some money. Thus, the overall aim is to find the shortest schedule for a given DAG and a given set of resources *without* exceeding the budget available. In this model, our emphasis is placed on the heuristics rather than the accurate modelling of a Grid environment; thus, we adopt a fairly static methodology in defining execution costs of the tasks of the DAG. However, as indicated by studies on workflow scheduling [1, 5, 9], it appears that heuristics performing best in a static environment (e.g., [6]) have the highest potential to perform best in a more accurately modelled Grid environment.

In order to solve the problem of scheduling optimally under a budget constraint, we propose two basic families of heuristics, which are evaluated in the paper. The idea in both approaches is to start from an assignment which has good performance under one of the two optimization criteria considered (that is, makespan and budget) and swap tasks between machines trying to optimize as much as possible for the other criterion. The first approach starts with an assignment of tasks onto machines that is optimized for makespan (using a standard algorithm for DAG scheduling onto heterogeneous resources, such as HEFT [8] or HBMCT [6]). As long as the budget is exceeded, the idea is to keep swapping tasks between machines by choosing first those tasks where the largest savings in terms of money will result in the smallest loss in terms of schedule length. We call this approach as **LOSS**. Conversely, a second approach starts with the cheapest assignment of tasks onto resources (that is, the one that requires the least money). As long as there is budget available, the idea is to keep swapping tasks between machines by choosing first those tasks where the largest benefits in terms of minimizing the makespan will be obtained for the smallest expense. We call this approach **GAIN**. Variations in how tasks are chosen result in different heuristics, which we evaluate in the paper.

The rest of the paper is organized as follows. Section 2 gives some background information about DAGs. In Section 3 we present the core algorithm proposed along with a description of the two approaches developed and some variants. In Section 4, we present experimental results that evaluate the two approaches. Finally, Section 5 concludes the paper.

2 Background

Following similar studies [1, 9, 7], the DAG model we adopt makes the following assumptions. Without loss of generality, we consider that a DAG starts with a single entry node and has a single exit node. Each node connects to other nodes with edges, which represent the node dependencies. Edges are annotated with a value, which indicates the amount of data that need to be communicated from a parent node to a child node. For each node the execution cost on each different machine available is given. In addition, the communication cost between machines is given. Traditional studies aim to assign tasks onto machines in such a way that the overall schedule length is minimized and precedence constraints are met. An example of a DAG and the schedule length produced using a well-known heuristic, HEFT [8] is shown in Figure 1. A number of other heuristics could be used too (see [6], for example). It is noted that in the example in the figure no task is ever assigned to machine M2. This is due to the high communication cost and since HEFT assigns tasks onto the machine that provides the earliest finish time, no task ever satisfies this condition.

The contribution of this paper relates to the extension of the traditional DAG model by adding (financial) cost to the available processors. The cost is measured in units of money. As a result, an additional constraint needs to be satisfied, namely that the overall financial cost of the schedule does not exceed a certain budget. We define the total cost (see Equation 2) as the sum of the costs of executing each task in the DAG onto a machine. This cost is calculated as the product of the execution time required by the task on the machine that has been assigned to, times the cost of this machine. This is shown in Equation 1, where $C_{i,j}$ is the cost of executing task i to machine j , $MachineCost_j$ is the cost (in money units) per unit of time to run something on machine j and $ExecutionTime_{i,j}$ is the time task i takes to execute on machine j .

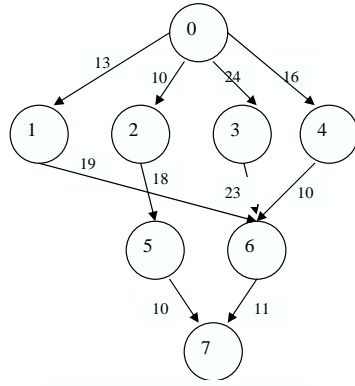
$$C_{i,j} = MachineCost_j \times ExecutionTime_{i,j} \quad (1)$$

$$TotalCost = \sum C_{i,j} \quad (2)$$

3 The Algorithm

3.1 Outline

The key idea of the algorithm proposed is to satisfy the budget constraint by finding the “best” affordable assignment possible. We define the “best assignment” as the assignment whose execution time is the minimum possible. The algorithm starts from a given assignment (schedule) and computes for each reassignment of each task to a different machine, a weight value associated with that particular change. A weight table is created for each task in the DAG and each machine. Two alternative approaches for computing the weight values are proposed depending on the two choices used for the starting assignment: either optimal for makespan (approach called LOSS), or cheapest (approach called GAIN); the two



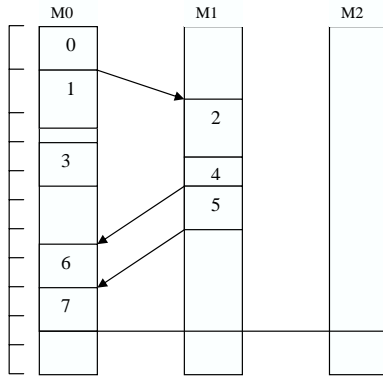
a) A DAG example

Task	M0	M1	M2
0	17.0	28.0	17.0
1	26.0	11.0	14.0
2	30.0	13.0	27.0
3	6.0	25.0	3.0
4	12.0	2.0	12.0
5	7.0	8.0	23.0
6	23.0	16.0	29.0
7	12.0	14.0	11.0

b) The computation cost of tasks on machines

Machines	Time for a data unit
M0-M1	1.607
M1-M2	0.9
M0-M2	3

c) Communication costs between the machines



d) Schedule derived by the HEFT algorithm

Task	Start Time	Finish Time
0	0.0	17.0
1	17.0	43.0
2	33.07	46.07
3	43.0	49.0
4	46.07	48.07
5	48.07	56.07
6	58.74	69.74
7	69.74	81.74

e) Start Time and Finish Time of each task in Schedule

Fig. 1. An Example of HEFT scheduling in a DAG workflow

approaches are described in more detail below. Using the weight table, tasks are repeatedly considered for possible reassignment to a machine, until the budget is exceeded. We illustrate the key steps of the algorithm in Table 1.

3.2 The LOSS Approach

The LOSS approach uses as a starting assignment the output assignment of either HEFT [8] or HBMCT [6] DAG scheduling algorithms. If the available budget is bigger or equal to the money cost required for this assignment then this assignment can be used straightaway. In all the other cases that the budget is less than the cost required for the starting assignment, the LOSS approach is invoked. The main aim of this approach is to make a change in the schedule (assignment)

```

Input: An application graph G and a schedule S produced by a scheduling algorithm H,
      Budget B
Cost-Time Scheduling Algorithm:
1) Get schedule S [] after running algorithm H.
2) Build an array A[task_number x machine_number]
3) Place a ZEROs at cells A[T,M] where task T is assigned to machine M as indicated in S []
4) For each Task in G
   for each Machine in G
     Compute the Weight value placed in A[Task,Machine]
   endfor
   endfor
5) While Cost of assignment < B
   Find the a smallest or bigger (depending on approach used) value from A.
   Value = A [i,j]
   Re-assign Task i to machine j in S[] and calculate new cost of assignment.
   endwhile
6) Return S

```

Table 1. The Basic Steps of the Proposed Scheduling Algorithm

given from, say HEFT, so that it will result in the minimum loss in execution time for the largest money savings. This means that the new schedule has an execution time close to the time the HEFT assignment would require but with less cost. In order to come up with such a re-assignment, the *LossWeight* values for each task to each machine are computed as follows:

$$LossWeight(i, m) = \frac{T_{new} - T_{old}}{C_{old} - C_{new}} \quad (3)$$

where T_{old} is the time to execute task i on the machine assigned by HEFT, T_{new} is the time to execute Task i on machine m . C_{old} , respectively, is the cost of executing task i on the machine given by the HEFT assignment and C_{new} is the cost of executing task i on machine m . If C_{old} is less than or equal to C_{new} the value of *LossWeight* is considered zero. The algorithm keeps trying re-assignments by considering the smallest values of the *LossWeight* for all tasks and machines (step 5) of the algorithm in Table 1.

3.3 The GAIN Approach

The GAIN approach uses as a starting assignment the assignment that requires the less money. Each task is initially assigned to the machine that executes the task with the smallest cost. This assignment is called the Cheapest Assignment. In this variation of the algorithm, the idea is to change the Cheapest Assignment by keeping re-assigning tasks to the machine where there is going to be the biggest benefit in makespan for the smallest money cost. This is repeated until there is no more money available (budget exceeded). In a way similar to

Equation 3, weight values are computed as follows. It is noted that tasks are considered for reassignment starting with those that have the largest *GainWeight* value.

$$GainWeight(i, m) = \frac{T_{old} - T_{new}}{C_{new} - C_{old}} \quad (4)$$

where T_{old} , T_{new} , C_{new} , C_{old} have exactly the same meaning as in the LOSS approach. Furthermore, if T_{new} is greater than T_{old} or C_{new} is equal to C_{old} we assign a weight value of zero.

3.4 Variants

For each of the two approaches above, we consider three different variants which relate to the way that the weights in Equations 3 and 4 are computed; these modifications result in slightly different versions of the heuristics. The three variants are:

- LOSS1 and GAIN1: in this case, the weights are computed exactly as described above.
- LOSS2 and GAIN2: in this case, the values of T_{old} , T_{new} , and C_{new} , C_{old} in Eqs 3 and 4 refer to the overall makespan and the overall cost, respectively, of the schedule and not the values associated with the individual tasks.
- LOSS3 and GAIN3: in this case, the weights in Equations 3 and 4 are recomputed each time a reassignment is made by the algorithm.

4 Experimental Results

4.1 Experiment Setup

The algorithm described in the previous section was incorporated in a tool developed at the University of Manchester, for the evaluation of different DAG scheduling algorithms [6, 7]. In order to evaluate each version of both approaches we run the heuristic proposed in this paper with four different types of DAGs used in the relevant literature [6, 7]: FFT, Fork-Join (denoted by FRJ), Laplace (denoted by LPL) and Random DAGs, generated as indicated in [10, 6], with around 100 nodes each. We run the heuristic proposed in the paper 100 times for each type of DAG and both approaches and their variants, and we considered the average values. In each case, we considered nine values for the possible budget, B , as follows:

$$B = C_{cheapest} + k \times (C_{HEFT} - C_{cheapest}), \quad (5)$$

where C_{HEFT} is the total cost of the assignment produced by HEFT and $C_{cheapest}$ is the cost of the cheapest assignment. The value of k varies between 0.1 and 0.9. Essentially, this approach allows us to consider values of budget that lie in ten equally distanced points between the money cost for the cheapest assignment and the money cost for HEFT. Clearly, values for budget outside those two ends are trivial to handle since they indicate that either there is no solution satisfying the given budget, or HEFT can provide a solution within the budget.

4.2 Metrics

Average Percentage Difference metric: In order to compare the quality of the schedule produced by the heuristic for each of the six variants and each type of DAG, and since 100 experiments are considered in each case, we normalize the schedule length (makespan) using the following formula:

$$\frac{T_{value} - T_{cheapest}}{T_{HEFT} - T_{cheapest}}, \quad (6)$$

where T_{value} is the makespan returned by the heuristic, $T_{cheapest}$ is the makespan of the cheapest assignment and T_{HEFT} is the makespan of HEFT. As a general rule, the makespan of $T_{cheapest}$ is expected to be the worst, hence, for comparison purposes, larger values in (6) indicate a shorter makespan. Since for each case we take 100 runs, the average value of the quantity above produces the *Average Percentage Difference* (APD) from the cheapest, that is,

$$APD = \frac{1}{100} \sum_{i=1}^{100} \left(\frac{T_{value}^i - T_{cheapest}^i}{T_{HEFT}^i - T_{cheapest}^i} \right), \quad (7)$$

where the superscript i denotes the i -th run.

Results showing the APD for each different type of DAG, variant, and budget available (shown in terms of the value of k — see Equation 5) are presented in Figure 2. The graphs show the difference of the two approaches. The LOSS variants are generally better than the GAIN variants. This might be due to the fact that the starting basis of the LOSS approach is HEFT which produces a short makespan. Instead, the GAIN approach starts from the Cheapest Assignment whose makespan is typically long. However, from the experimental results we notice that in cases where the budget is close to the cheapest budget, the APD of the GAIN approaches becomes bigger than the APD of the LOSS approaches. This can be seen in Figure 2 for Random and FRJ DAGs.

Execution Time for each Heuristic: To evaluate the performance of each version in both approaches we extracted for the experiments we carried out before, the execution time of the algorithm. The results are shown in Figure 3. Same as before, the execution time is the average value from 100 runs.

Observations: The above experiments indicate that the algorithm proposed in this paper is able to find affordable assignments with better makespan when the LOSS approach is applied, instead with the GAIN approach. The LOSS approach applies re-assignment to an assignment that is given by a good DAG scheduling heuristic, whereas in the GAIN approach the cheapest assignment is used which may have the worst makespan. However, in cases where the available budget is close to the cheapest budget, GAIN1 gives better makespan than LOSS1 or LOSS2. This can contribute to the optimisation in the performance of the heuristic.

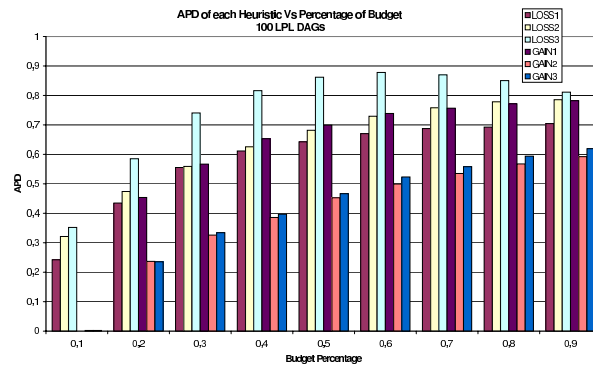
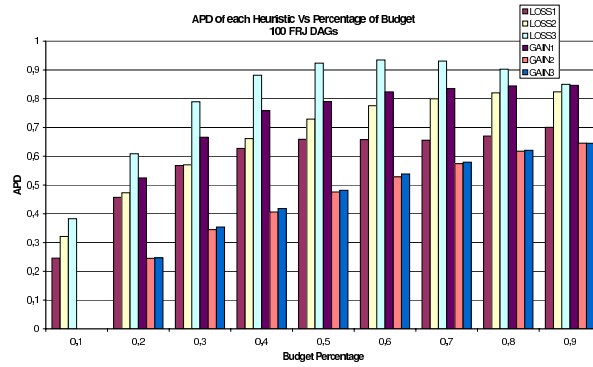
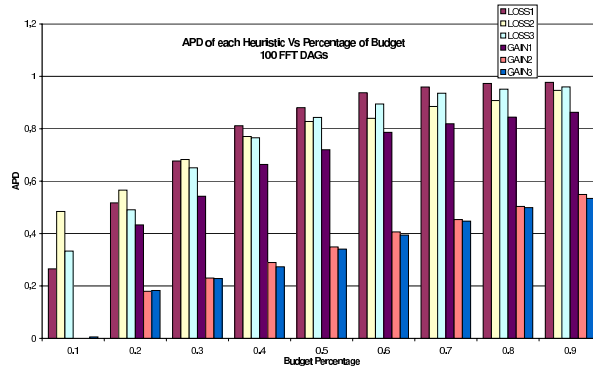
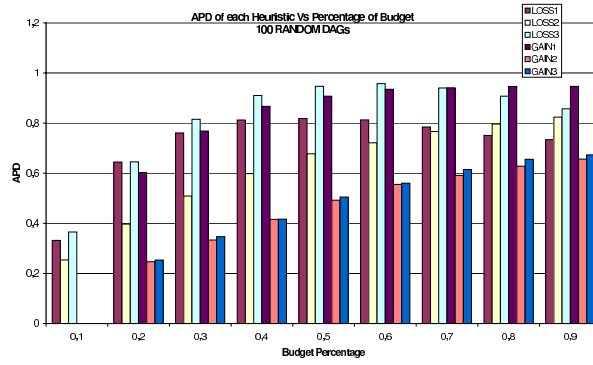


Fig. 2. Average Percentage Difference of 100 runs

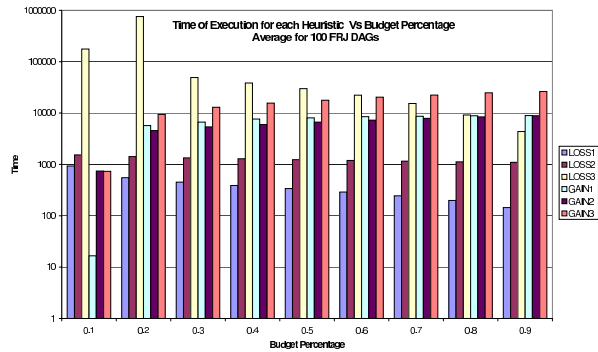
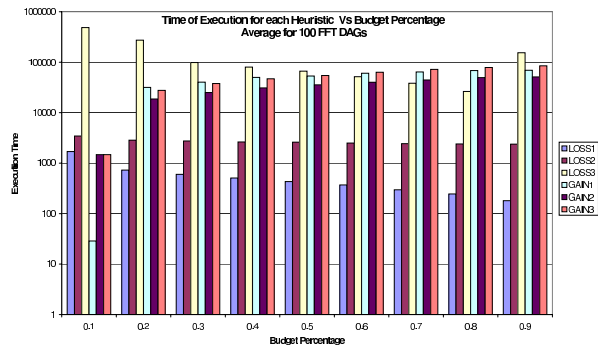
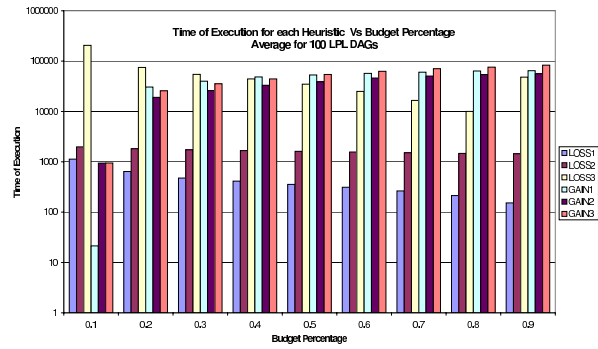
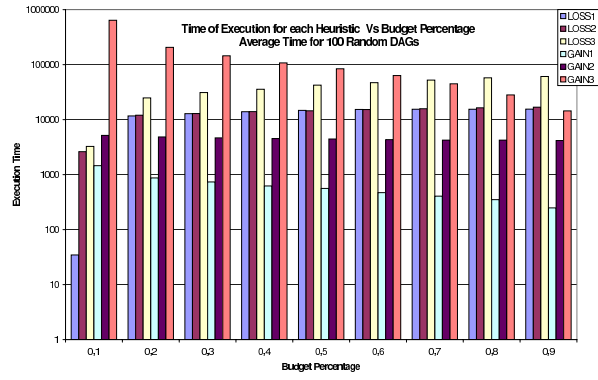


Fig. 3. Average Execution Time of 100 runs

Regarding the execution time, it appears that the LOSS approach takes more time as we move towards a budget close to the cost of the cheapest assignment; the opposite happens with the GAIN approach. This is correlated with the starting basis of each of the two approaches.

5 Conclusion

We have implemented an algorithm to schedule DAGs onto heterogeneous machines under budget constraints. Different variants of the algorithm were modelled and evaluated. Future work could consider other types of DAGs that correspond to workflows of interest in the Grid community (e.g., [1, 9]), could include more sophisticated models to charge for machine time (although relevant research in the context of the Grid is still in its infancy), and consider more dynamic scenarios and environments for the execution of the DAGs and the modelling of the machines.

References

1. J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Resource Allocation Strategies for Workflows in Grids In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005)*.
2. R. Buyya, D. Abramson, and S. Venugopal. The Grid Economy. In *Proceedings of the IEEE*, volume 93(3), pages 698–714, March 2005.
3. R. Buyya. *Economic-based Distributed Resource Management and Scheduling for Grid Computing*. PhD thesis, Monash University, Melbourne, Australia, <http://www.buyya.com/thesis>, April 12 2002.
4. R. Buyya, D. Abramson, and J. Giddy. An economy grid architecture for service-oriented grid computing. In *10th IEEE Heterogeneous Computing Workshop (HCW'01)*, San Francisco, 2001.
5. A. Mandal, K. Kennedy, C. Koelbel, G. Marin, J. Mellor-Crummey, B. Liu and L. Johnsson. Scheduling Strategies for Mapping Application Workflows onto the Grid. In *IEEE International Symposium on High Performance Distributed Computing (HPDC 2005)*, 2005.
6. R. Sakellariou and H. Zhao. A hybrid heuristic for DAG scheduling on heterogeneous systems. In *13th IEEE Heterogeneous Computing Workshop (HCW'04)*, Santa Fe, New Mexico, USA, April 2004.
7. R. Sakellariou and H. Zhao. A low-cost rescheduling policy for efficient mapping of workflows on grid systems. In *Scientific Programming*, volume 12(4), pages 253–262, December 2004.
8. H. Topcuoglu, S. Hariri, and M. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. In *IEEE Transactions on Parallel and Distributed Systems*, volume 13(3), pages 260–274, March 2002.
9. M. Wiczorek, R. Prodan and T. Fahringer. Scheduling of Scientific Workflows in the ASKALON Grid Environment. In *SIGMOD Record*, volume 34(3), September 2005.
10. H. Zhao and R. Sakellariou. An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm. In *Euro-Par 2003*. Springer-Verlag, LNCS 2790, 2003.