SCHEDULING WORKFLOWS WITH BUDGET CONSTRAINTS*

Rizos Sakellariou and Henan Zhao
School of Computer Science
University of Manchester
U.K.
rizos@cs.man.ac.uk
hzhao@cs.man.ac.uk

Eleni Tsiakkouri and Marios D. Dikaiakos
Department of Computer Science
University of Cyprus
Cyprus
cstsiak@cs.ucy.ac.cy
mdd@cs.ucy.ac.cy

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.

Keywords: Workflows, Scheduling, Budget Constrained Scheduling, DAG Scheduling.

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

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) [9, 12, 2, 7]. 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). A DAG represents 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 [1, 8, 10, 11] 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 [3, 6]. Of particular interest will be the resource access cost, since different resources, belonging to different organisations, may have different policies for charging. 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 [4–5], where it is demonstrated, through Grid simulation, how a scheduling algorithm can allocate jobs to machines in a way that satisfies constraints of Deadline and Budget at the same time. 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) [4]. Workflow types of applications, where jobs have precedence constraints, are not considered.

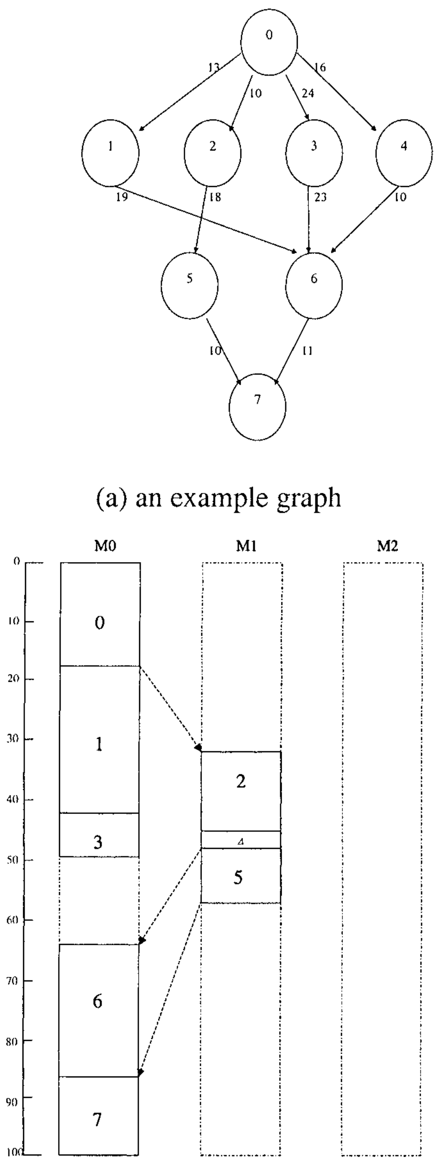
In this paper, we consider workflow applications that are modelled as DAGs. Instead of focussing only on makespan optimisation, as most existing studies have done [2, 8, 10], we also consider that a budget constraint needs to be satisfied. Each job, when running on a machine, costs some money. Thus, the overall aim is to find the schedule that gives the shortest makespan 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 [2, 7, 12], it appears that heuristics performing best in a static environment (e.g., HBMCT [8]) 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 [10] or HBMCT [8]). 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, the 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 [2, 12, 9], 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 time on each different machine available is given. In addition, the time to communicate data between machines is given. Using this input, traditional studies from the literature 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 [10], is shown in Figure 1. A number of other heuristics could



(a) an example graph

(d) the schedule derived using the HEFT algorithm

task	m0	m1	m2
0	17	28	17
1	26	11	14
2	30	13	27
3	6	25	3
4	12	2	12
5	7	8	23
6	23	16	29
7	12	14	11

(b) the computation cost of nodes on three different machines

machines	time for a data unit
m0 - m1	1.607
m1 - m2	0.9
m0 - m2	3.0

(c) communication cost between the machines

node	start time	finish time
0	0	17
1	17	43
2	33.07	46.07
3	43	49
4	46.07	48.07
5	48.07	56.07
6	64.14	87.14
7	87.14	99.14

(e) the start time and finish time of each node in (d)

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

be used too (see [8], for example). It is noted that in the example in the figure no task is ever assigned to machine M2. This is primarily due to the high communication; 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 with one extra condition: the usage of each machine available costs some money. As a result, an additional constraint needs to be satisfied when scheduling the DAG, namely, that the overall financial cost of the schedule does not exceed a certain budget. We define the overall (total) cost as the sum of the costs of executing each task in the DAG onto a machine, that is,

$$TotalCost = \sum C_{ij}, \tag{1}$$

where C_{ij} is the cost of executing task i onto machine j and 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, that is,

$$C_{ij} = MachineCost_j \times ExecutionTime_{ij}, \tag{2}$$

where $MachineCost_j$, is the cost (in money units) per unit of time to run something on machine j and $ExecutionTime_{ij}$ is the time task i takes to execute on machine j. Throughout this paper, we assume that the value of $MachineCost_j$, for all machines, is given.

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. We define "affordable assignment" as the assignment whose cost does not exceed the budget available. We also assume that, on the basis of the input given, the budget available is higher than the cost of the cheapest assignment (that is, the assignment where tasks are allocated onto the machine where it costs the least to execute them); this guarantees that there is at least one solution within the budget available. We also assume that the budget available is less than the cost of the schedule that can be obtained using a DAG scheduling algorithm that aims to minimize the makespan, such as HEFT or HBMCT. Without the latter assumption, there would be no need for further investigation: since the cost of the schedule produced by the DAG scheduling would be within the budget available, it would be reasonable to use this schedule.

The algorithm starts with an initial assignment of the tasks onto machines (schedule) and computes for each reassignment of each task to a different machine, a weight value associated with that particular change. Those weight values are tabulated; thus, 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 initial assignment: either optimal for makespan (approach called LOSS — in this case, the initial assignment would be produced by an efficient DAG scheduling heuristic [10, 8]), or cheapest (approach called GAIN — in this case, the initial assignment would be produced by allocating tasks to the machines where it costs the least in terms of money; we call this as the cheapest assignment); the two approaches are described in more detail below. Using the weight table, tasks are repeatedly considered for possible reassignment to a machine, as long as the cost of the current schedule exceeds the budget (in the case that LOSS is followed), or, until all possible reassignments would exceed the budget (in the case of GAIN). In either case, the algorithm will try to reassign any given pair of tasks only once, so when no reassignment is possible the algorithm will terminate. We illustrate the key steps of the algorithm in Figure 2.

3.2 The LOSS Approach

The LOSS approach uses as an initial assignment the output assignment of either HEFT [10] or HBMCT [8] 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 and no further action is needed. In all the other cases that the budget is less than the cost required for the initial assignment, the LOSS approach is invoked. The aim of this approach is to make a change in the schedule (assignment) obtained through HEFT or HBMCT, 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 original assignment would require but with less cost. In order to come up with such a re-assignment, the LOSS weight values for each task to each machine are computed as follows:

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

where T_{old} is the time to execute task i on the machine assigned by HEFT or HBMCT, T_{new} is the time to execute Task i on machine m. Also, C_{old} is the cost of executing task i on the machine given by the HEFT or HBMCT 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 4 of the algorithm in Figure 2).

```
Input: A DAG (workflow) G with task execution time and communication
       A set of machines with cost of executing jobs
       A DAG scheduling algorithm H
       Available Budget B

Algorithm: (two options: LOSS and GAIN)
1) If LOSS
     then generate schedule S using algorithm H
     else generate schedule S by mapping each task onto the cheapest machine
2) Build an array A[number_of_tasks][number_of_machines]
3) for each Task in G
     for each Machine
       if, according to Schedule S, Task is assigned to Machine
         then A[Task][Machine] $\leftarrow$ 0
         else Compute the Weight for A[Task][Machine]
     endfor
   endfor
4) if LOSS
     then condition $\leftarrow$ (Cost of schedule S $>$ B)
     else condition $\leftarrow$ (Cost of schedule S $\leq$ B)
   While (condition and not all possible reassignments have been tried)
     if LOSS
       then find the smallest non-zero value from A, A[i][j]
       else find the biggest non-zero value from A, A[i][j]
     Re-assign Task i to Machine j in S and calculate new cost of S.
     if (GAIN and cost of S $>$ B)
       then invalidate previous reassignment of Task i to Machine j.
   endwhile
5) if (cost of schedule S $>$ B)
     then use cheapest assignment for S.
6) Return S
```

Figure 2. The Basic Steps of the Proposed Algorithm

3.3 The GAIN Approach

The GAIN approach uses as a starting assignment the assignment that requires the least 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.

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

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 Equations 3 and 4 refer to the benefit in terms of the overall makespan and the overall cost for the schedule and not the benefit associated with the individual tasks being considered for reassignment.

- LOSS3 and GAIN3: in this case, the weights, computed as shown by 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 [8–9]. In order to evaluate each version of both approaches we run the algorithm proposed in this paper with four different types of DAGs used in the relevant literature [8–9]: FFT, Fork-Join (denoted by FRJ), Laplace (denoted by LPL) and Random DAGs, generated as indicated in [13, 8]. All DAGs contain about 100 nodes each and they are scheduled on 3 different machines. We run the algorithm 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:

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

where C_{DAG} is the total cost of the assignment produced by the DAG scheduling heuristic used for the initial assignment (that is, HEFT or HBMCT) when the LOSS approach is considered 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 the schedule generated by HEFT or HBMCT. 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 and/or HBMCT can provide a solution within the budget.

4.2 Results

Average Normalized Difference metric: In order to compare the quality of the schedule produced by the algorithm 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:

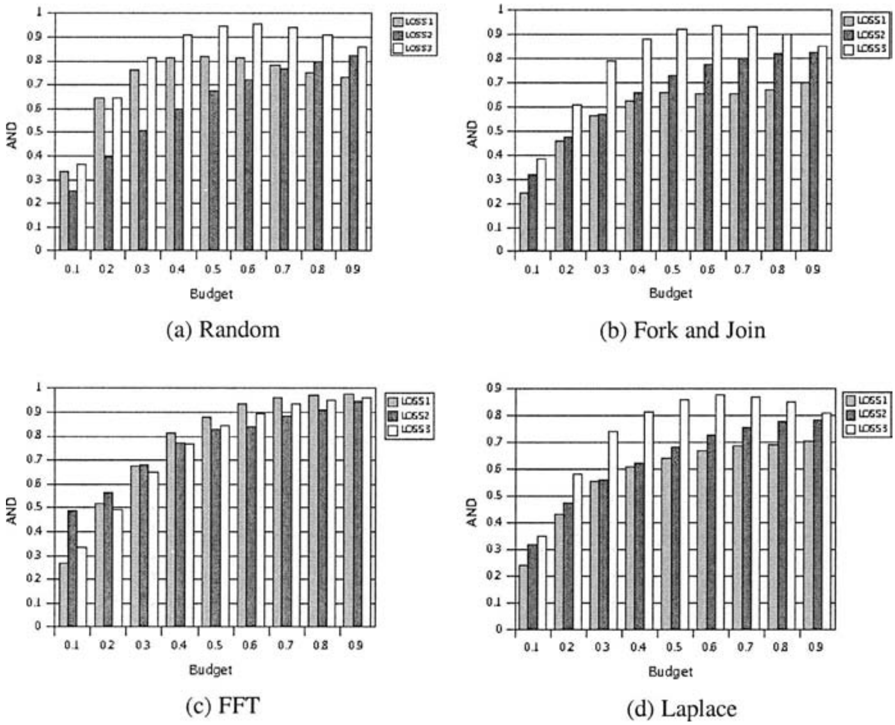
$$\frac{T_{value} - T_{cheapest}}{T_{DAG} - T_{cheapest}}, \tag{6}$$

where T_{value} is the makespan returned by our algorithm, $T_{cheapest}$ is the makespan of the cheapest assignment and T_{DAG} is the makespan of HEFT or HBMCT. As a general rule, the makespan of the cheapest assignment, $T_{cheapest}$, is expected to be the worst (longest), and the makespan of HEFT or HBMCT, T_{DAG}, the best (shortest). As a result, the formula above is expected to return a value between 0 and 1 indicating how close the algorithm was to each of the two bounds (note that since HEFT or HBMCT are greedy heuristcs, occasional values which are better than the values obtained by those two heuristics may occur). Hence, for comparison purposes, larger values in Equation 6 indicate a shorter makespan. Since for each case we take 100 runs, the average value of the quantity above produces the *Average Normalized Difference* (AND) from the worst and the best, that is,

$$AND = \frac{1}{100}\sum_{i=1}^{100}\left(\frac{T^i_{value} - T^i_{cheapest}}{T^i_{DAG} - T^i_{cheapest}}\right), \tag{7}$$

where the superscript i denotes the i-th run.

Results showing the AND for each different type of DAG, variant, and budget available (shown in terms of the value of k — see Equation 5) are presented in Figures 3, 4 and 5. Each figure groups the results of a different approach: LOSS starting with HEFT, LOSS starting with HBMCT, and GAIN (in the latter case, a DAG scheduling heuristic would not make any difference, since the initial schedule is built on the basis of assigning tasks to the machine with the least cost). The graphs show the difference of the two approaches. The LOSS variants have a generally better makespan than the GAIN variants and they are capable of



(a) Random

(b) Fork and Join

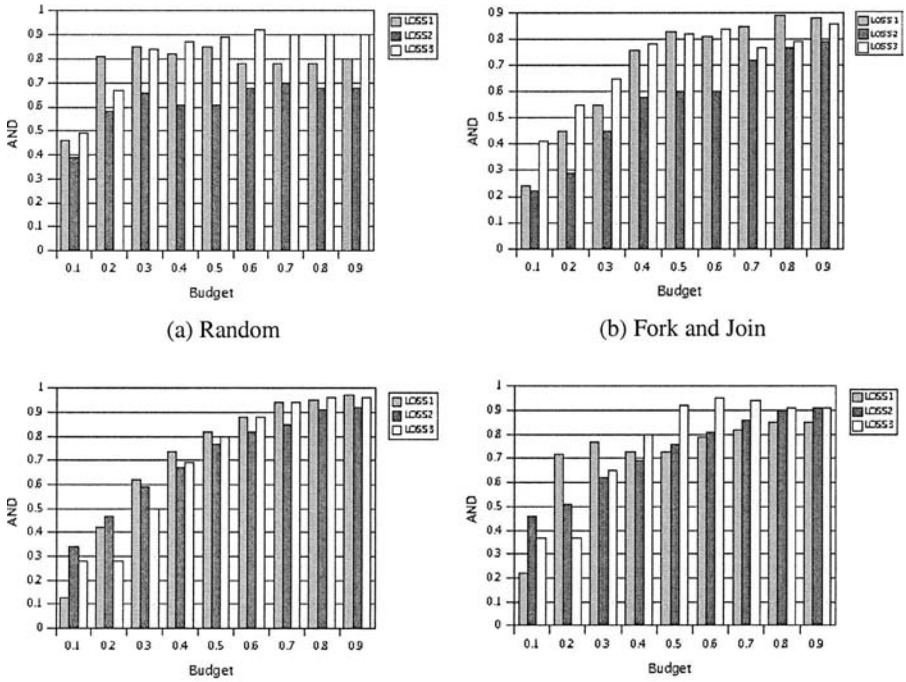
(c) FFT

(d) Laplace

Figure 3. Average normalized difference for the three variants of LOSS when HEFT is used to generate the initial schedule.

performing close to the baseline performance of HEFT or HBMCT (that is, the value 1 in Figures 3 and 4) for different values of the budget. This is due to the fact that the starting basis of the LOSS approach is a DAG scheduling heuristic, which already produces a short makespan. Instead, the GAIN variants starts from the Cheapest Assignment whose makespan is typically long. However, from the experimental results we notice that in a few, limited, cases where the budget is close to the cheapest budget, the AND of the first variant of the GAIN approach is higher than the AND of the LOSS approaches.

Running Time for the Algorithm: To evaluate the performance of each version of the algorithm, using both the LOSS and GAIN approaches, we extracted from the experiments we carried out before, the running time of the algorithm. It appears that the results have little difference between different types of DAGs, so we include here only the results obtained for FFT graphs. Two graphs are presented in Figure 6; one graph assumes that the starting point for LOSS is HEFT and the other graph assumes that the starting point for LOSS is HBMCT. Same as before, the execution time is the average value from 100 runs. It can be



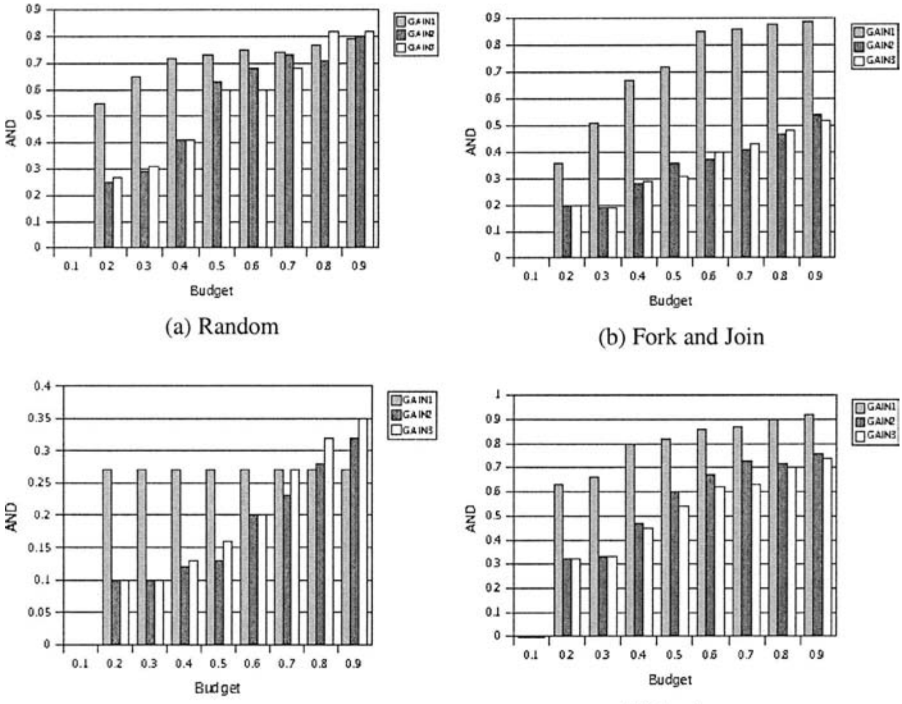
(a) Random

(b) Fork and Join

(c) FFT

(d) Laplace

Figure 4. Average normalized difference for the three variants of LOSS when HBMCT is used to generate the initial schedule.



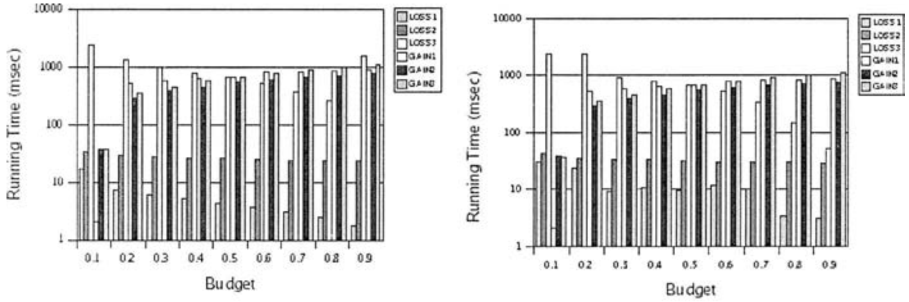
(a) Random

(b) Fork and Join

(c) FFT

(d) Laplace

Figure 5. Average normalized difference for the three variants of GAIN.



(a) HEFT

(b) HBMCT

Figure 6. Average running time for each variant of the algorithm, using FFT DAGs.

seen that the GAIN approaches, generally, take longer than the LOSS approaches (the exception seems to arise in cases where the budget is close to the cheapest assignment and the GAIN approaches are quick in identifying a solution). Also, as expected, the third variant of LOSS, which involves re-computation of the weights after each reassignment of tasks, takes longer than the other two.

Summary of 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 to build the schedule; this 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 observation can contribute to the optimization in the performance of the algorithm.

Regarding the running 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. The main conclusion is that starting from an optimized schedule, in terms of its makespan, pays off when trying to satisfy the budget constraint. As for future work: (i) other types of DAGs that correspond to workflows of interest in the Grid community could be considered (e.g., [2, 12]); (ii) more sophisticated models to charge for machine time could be incorporated (although relevant research in the context of the Grid is still in its infancy); and, (iii) more dynamic scenarios and environments for the execution of the DAGs and the modelling of the machine time could be considered (e.g., [9]).

References

[1] O. Beaumont, V. Boudet, and Y. Robert. A realistic model and an efficient heuristic for scheduling with heterogeneous processors. In *11th Heterogeneous Computing Workshop*, 2002.

[2] 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).*

[3] R. Buyya, D. Abramson, and S. Venugopal. The Grid Economy. In *Proceedings of the IEEE*, volume 93(3), pages 698–714, March 2005.

[4] 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.

[5] 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 Fransisco, 2001.

[6] C. Ernemann, V. Hamscher and R. Yahyapour. Economic Scheduling in Grid Computing. In *Proceedings of the 8th Workshop on Job Scheduling Strategies for Parallel Processing*, Vol. 2537 of Lecture Notes in Computer Science, Springer, pages 128–152, 2002.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski. Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. *Journal of Parallel and Distributed Computing*, 47:8–22, 1997.

[12] M. Wieczorek, R. Prodan and T. Fahringer. Scheduling of Scientific Workflows in the ASKALON Grid Environment. In *SIGMOD Record*, volume 34(3), September 2005.

[13] 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.