A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems

Rizos Sakellariou and Henan Zhao
Department of Computer Science
University of Manchester
Oxford Road
Manchester M13 9PL, U.K.
{rizos,hzhao}@cs.man.ac.uk

Abstract

This paper is motivated by the observation that different methods to compute the weights of nodes and edges when scheduling DAGs onto heterogeneous machines may lead to significant variations in the generated schedule. To minimize such variations, the paper presents a novel heuristic for DAG scheduling, which is based upon solving a series of independent task scheduling problems. A novel heuristic for the latter problem is also included in the paper. Both heuristics compare favourably with other related heuristics.

1. Introduction

Task scheduling for heterogeneous systems is a well studied problem, a consequence of its significance on application performance. Applications are typically represented by means of a *directed acyclic graph* (DAG) and a number of heuristics have been proposed to schedule the nodes (or tasks; the terms are used interchangeably throughout the paper) of the DAG onto the heterogeneous machines (see, for instance, [11, 14] for an extensive list of references). Heuristics based on list scheduling are among those that provide good quality schedules at a reasonable cost.

In a recent study [16], it was observed that the performance of a commonly cited list scheduling heuristic, HEFT [14], is affected significantly by the approach followed to assign weights to the nodes and edges of the graph. In an extreme case, the makespan that HEFT returned for a certain graph was 47.2% worse than the makespan that would be obtained if a different approach to compute the weights of the nodes and edges of the graph was chosen [16]. Note that the heterogeneous setting allows for a variety of different approaches to compute the weights; for instance, the weight of a node could be obtained as the average of its corresponding execution time across all machines, or the median, or the smallest value, etc.

In this paper, carrying this study further, we observed that other DAG scheduling heuristics exhibit a similar behaviour too; some results are included here in Section 2. Although it has long been known that the approach followed to rank the nodes of a graph may affect the quality of the schedule produced by list scheduling, we believe that this is exacerbated in heterogeneous environments as a result of the heterogeneity. The resulting variations in the makespan may be so significant that it can be difficult to determine the baseline behavior of a heuristic. More importantly, the sensitivity that the heuristics exhibit, with respect to the approach used to compute the weights, indicates that there is scope for improvement in their design.

These observations motivated the work presented in this paper. The main contributions are:

- A novel heuristic for DAG scheduling on heterogeneous machines, which compares favourably with other related heuristics and shows less sensitivity to different approaches for weighting nodes/edges. We call this heuristic *hybrid*, because it uses list scheduling to break the whole problem to subproblems where heuristics for scheduling independent tasks can be applied.
- A novel heuristic for scheduling independent tasks on heterogeneous machines, which outperforms other known heuristics.

The remainder of the paper is organized as follows. Section 2 provides some background, describes related work and presents some results on the impact of different approaches to compute the weights of nodes, a motivation for the work that follows. Section 3 forms the main body of the paper, describing two novel heuristics, for DAG scheduling and for independent task scheduling, and illustrating them with an example. The two heuristics are evaluated in Section 4. Finally, Section 5 concludes the paper.

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

2. Background

The application model we consider in this paper is a directed acyclic graph (DAG). Each node in the graph represents an executable task. Each directed edge represents a precedence constraint (or simply a dependence) between two tasks; the sink node cannot start execution until the source node has finished and the transmission of the required amount of data from the source node to the sink node has been completed. We assume that the DAG has always a single entry node (i.e., a node with no parents) and a single exit node (i.e., a node with no children). The target environment consists of a set of heterogeneous machines, which are fully connected; a data transfer cost is given for each pair of machines. A task can execute on any available machine; the execution cost of each task on each machine is also given. The task scheduling problem is to allocate tasks for execution onto machines in such a way that precedence constraints are respected and the overall execution time (makespan) is minimized. It is assumed that only one task can execute on a machine at a time and once a task has started execution on a machine it cannot be preempted.

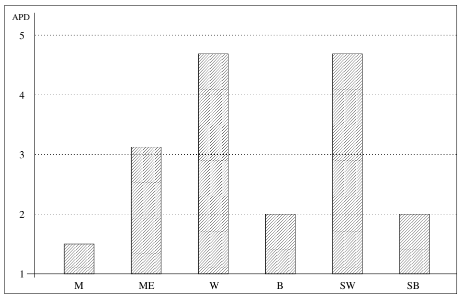
There is a vast amount of literature related to heuristics for DAG scheduling on heterogeneous systems [4, 5, 12, 15]. An important family of heuristics is based on list scheduling (e.g., [13, 14]). In list scheduling, a weight is assigned to each node and edge of the graph; these weights are used to prioritize the nodes, which are subsequently assigned in this order to machines. Other families of heuristics are based on grouping nodes into clusters which are then considered for assignment to machines (e.g., [5]), task duplication (e.g., [12]), scheduling the critical path first (e.g., [14]), or scheduling by dividing the DAG into levels (e.g., [6]). In our study, we focus on five heuristics with good performance characteristics: Dynamic Level Scheduling (DLS) [13]; Heterogeneous Earliest Finish Time (HEFT) [14]; Critical Path On a Processor (CPOP) [14]; Fastest Critical Path (FCP) [9]; and Levelized-Min Time (LMT) [6].

In [16], it was observed that different methods for computing the weights of the nodes and edges of the DAG may have a significant impact on the schedule produced by HEFT. Note that in a homogeneous system an appealing approach to assign weights to nodes and edges is to use task computation and communication costs, respectively. However, in a heterogeneous setting these costs may vary across different machines and one could use these costs in different ways to compute a weight; for instance, one could take the average value, the smallest value, etc... Extending the study in [16] we noticed that different methods for computing the weight make a significant impact on the schedule produced by other heuristics too. The different methods we used in our study are those also used in [16], that is:

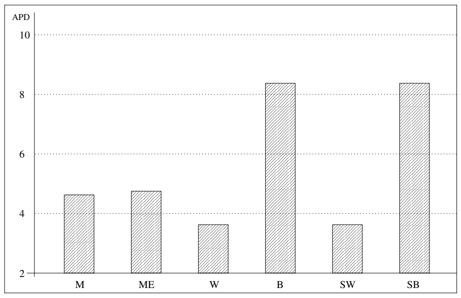
- *mean value* (denoted by M) is based on the average computation cost of each task across all machines (to assign a weight to a node) and the average communication cost between two tasks (to assign a weight to an edge).
- *median value* (denoted by ME) is based on the median value respectively.
- *worst value* (denoted by W) is based on the worst value of the execution cost, that is, maximum computation cost, to assign the weight of a node. The weight of an edge is based on the communication cost that corresponds to the two machines on which each of the two communicating tasks has its highest computation cost.
- *best value* (denoted by B) is based on the best value of the execution cost, that is, minimum computation cost, while the weight of an edge is based on the communication cost that is determined by the procedure described previously.
- *simple worst value* (denoted by SW) is based on the worst value for both computation and communication (that is, maximum computation and maximum communication).
- *simple best value* (denoted by SB) is based on the best value for both computation and communication (that is, minimum computation and minimum communication).

Using one thousand randomly generated DAGs (generated as explained in [16] and further in Section 4.2 of this paper) and for each of the five heuristics mentioned earlier, we compared the makespan produced by each different method to compute the weights. The comparison is carried out using the following metrics:

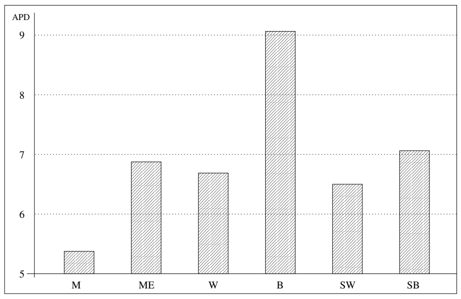
- The *average percentage degradation* (APD) [8] is the average (over all DAGs) of the percentage of degradation of the makespan generated by a particular method from the best makespan (produced by any of the six different methods).
- The *number of best solutions* (denoted by NB) is the number of time a particular method to compute the weights was the *only one* that produced the shortest makespan. This metric is complemented with the *number of best solutions equal with another method* (denoted by NEB), which counts those cases where a particular method produced the shortest makespan but at least another one method also achieved the same makespan.



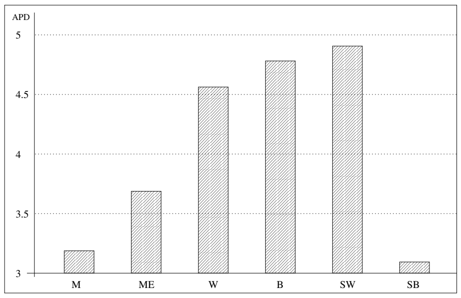
(a) DLS



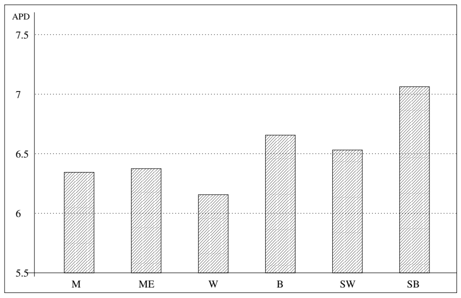
(b) LMT



(c) CPOP



(d) HEFT



(e) FCP

Figure 1. Average Percentage Degradation (APD) from the best schedule for each of six different methods to compute weights.

	DLS		
	NB	*NEB*	*WPD*
M	396	7	16.4
ME	147	6	16.0
W	0	83	19.6
B	0	371	15.8
SW	0	83	19.6
SB	0	371	15.8

	LMT		
	NB	*NEB*	*WPD*
M	276	1	27.0
ME	270	1	38.7
W	0	359	30.4
B	0	95	39.6
SW	0	359	30.4
SB	0	95	39.6

	CPOP		
	NB	*NEB*	*WPD*
M	214	2	53.0
ME	144	1	41.1
W	198	1	47.4
B	103	0	50.0
SW	170	0	41.0
SB	169	1	41.1

	HEFT		
	NB	*NEB*	*WPD*
M	230	4	17.0
ME	186	2	19.0
W	126	1	22.9
B	110	3	24.9
SW	95	1	19.2
SB	248	4	19.0

	FCP		
	NB	*NEB*	*WPD*
M	150	11	31.9
ME	173	9	36.4
W	177	4	43.2
B	177	1	35.8
SW	170	4	34.3
SB	137	3	40.5

Figure 2. Three additional metrics for the experiments in Figure 1: Number of times a method gives the best schedule *NB*; number of times a method gives the best, but another one is also giving the best too *NEB*; and worst percentage degradation *WPD*, that is the maximum percentage degradation over all cases.

- Finally, as an indication of what is the absolutely worst that a method performed over the 1000 cases, we consider the *worst percentage degradation* (WPD), which is the maximum value of the percentage degradation of the makespan of a given method from the makespan of the best method for a particular case.

The average percentage degradation (APD) for each of the five heuristics and the six different methods is shown in Figure 1. The APD tends to be higher for certain heuristics, however, we regard it as significant in all cases. Furthermore, no particular method to compute the weights seems to give consistently better results. These observations can be enhanced by the additional metrics presented in Figure 2. In an extreme case, for CPOP, the mean value method performs 53% worse than the best method for the same DAG.

The significant variations observed in the makespan highlight the sensitivity of the heuristics to different methods to compute the weights. This sensitivity appears to be largely due to the difficulty of the heuristics to assess meaningfully the relative importance of independent tasks in the DAG (and rank them appropriately). In turn, this is a consequence of the variations inherently present in the heterogeneity. To minimize the effect of all this, we focussed on the development of a heuristic that would make use of a more robust approach for scheduling the independent tasks of the DAG. This is presented in the next section.

3. A Hybrid Heuristic

3.1. Overview/Outline

The key idea of the hybrid heuristic is to use a standard list scheduling approach to rank the nodes of the DAG and then use this ranking to assign tasks to groups of tasks that can be subsequently scheduled independently. The input of the heuristic is a DAG and two arrays: one array gives the execution cost of each node on each machine, and another array gives the communication cost between two nodes connected by an edge on all combinations of different machines where these nodes may run (we assume that this cost is zero if two tasks connected by an edge are executed by the same machine).

The heuristic consists of three phases: *ranking*, *group creation*, *scheduling independent tasks within each group*. In the first phase, a weight is assigned to each node and edge of the graph; this is based on averaging all possible values for the cost of each node (or edge, respectively) on each machine (or combination of machines, respectively). Using this weight, upward ranking is computed and a rank value is assigned to each node. The rank value, r_i, of a node i is recursively defined as follows:

$$r_i = w_i + \max_{\forall j \in S_i}(c_{ij} + r_j),$$

where w_i is the weight of node i, S_i is the set of immediate successors of node i and c_{ij} is the weight of the edge connecting nodes i and j.

In the second phase, nodes are sorted in descending order of their rank value; using this order, they are considered for assignment to groups as follows. The first node (i.e., the node with the highest rank value) is added to a group numbered 0. Successive nodes, always in descending order of their rank value, are placed in the same group as long as they are independent with all the nodes already assigned to the group (i.e., there is no dependence between them in the DAG). If a dependence is found, then the node with the smallest rank value (i.e., the sink of the dependence) is made the member of a new group; the new group's number is the current group's number increased by one. Again, subsequent tasks, in terms of their rank value, will be added to this group as long as they are not dependent to any other node which is a member of this group; if they are, a new group will be created and so on. The outcome from this process is a set of ordered groups, each of which consists of a number of *independent* tasks, and has a predetermined priority (based on the original ranking of the nodes; a smaller group number indicates higher priority).

In the third phase, a schedule of the DAG can be obtained by considering each group in ascending order of its number, and using *any* heuristic for scheduling the independent tasks within each group. It is noted that the input of the latter heuristic will be a set of (independent) tasks; a set of machines; the array giving the execution cost of each node on any machine; and, another array giving the earliest time that each task may start execution on each machine. The latter, which we call EST, has to be computed once before the tasks of a group are to be considered for scheduling. The value of an entry EST_{ij} of this array, referring to task i and machine j, will be given by $EST_{ij} = \max(FT_j, ETT_{ij})$, where FT_j is the time that machine j finishes the execution of all tasks of the previous group(s) (hence, the time when work can be scheduled on it), and ETT_{ij} is the time that all the data needed to execute task i on machine j is available (this is computed by considering all immediate ancestors of task i, the time they finish, and any time needed to transfer data from the machine where they run on to machine j — recall that, by definition, all ancestors will belong to a group of higher priority and as a result they have already been scheduled).

An outline of the heuristic is given in Figure 3. Although the hybrid heuristic may seem similar to level-based heuristics [2, 6], there is a fundamental difference. Level-based heuristics attempt to split nodes into groups on the basis of the DAG structure. Our approach takes into account both the DAG structure and the computation and communication costs. Thus, weighting and ranking take place first, and it is on that basis that partitioning of nodes happens.

```
(1)  Assign a weight to each node as the average computation cost across all machines
     Assign a weight to each edge as the average communication cost across all combinations of machines.
     Use upward ranking to compute a rank value for each node.
(2)  Sort nodes in descending order of their rank values.
     $G_0 = \{\}; i = 0$
     Scan nodes in descending order of their rank values
         if current node has a dependence with a node in $G_i$
             then $i++$;   $G_i = \{\}$
         add node to $G_i$
     keep scanning until there are no more nodes
(3)  For all groups, $G_i$, in ascending order of $i$
         compute EST array (earliest time each task may start execution on each machine)
         schedule independent tasks in $G_i$, taking account of EST
     endfor
```

Figure 3. A hybrid heuristic for scheduling DAGs on heterogeneous machines.

3.2. Scheduling Independent Tasks

Although there exist many heuristics to schedule a number of independent tasks onto heterogeneous machines [3] — which could be used for the corresponding phase of the hybrid heuristic presented above — we propose here a novel heuristic. As will be shown next, this outperforms other, similar heuristics. We refer to this heuristic as *Balanced Minimum Completion Time* (BMCT), to reflect the fact that after an initial allocation of tasks to machines which minimize the execution time, there is a phase that tries to minimize the overall time by swapping tasks between machines in an attempt to 'balance' their work. As already indicated, the input of our heuristic is: a set of independent tasks; a set of machines; an array, W, giving the cost of executing each task on each machine; and an array, EST, showing, for each task, the earliest time it can be scheduled on each machine. The objective is to complete the execution of all tasks as early as possible (i.e., to minimize the overall makespan).

The algorithm consists of two phases. In the first phase, an *initial allocation* of tasks onto machines takes place; in the second, *optimization* phase, selected tasks are moved between machines in order to minimize the overall schedule length. The initial allocation phase assigns each task to the machine that gives the fastest execution time. Once all tasks are assigned to a machine, then the tasks of every machine are sorted in ascending order of their EST value for this particular machine. This sorted order determines the order in which the tasks of a given machine need to be executed so that a minimal execution time for this particular machine is achieved. The completion (or finish) time of a given task, i, on a machine, m, is given by

$$ft_{im} = W_{im} + \max(EST_{im}, ft_{i'm}),$$

where $ft_{i'm}$ is the time that the task, i', executing before task i on that machine, m, will finish. The finish time of the first task, say i_1, scheduled on that machine, m, will be $ft_{i_1m} = W_{i_1m} + EST_{i_1m}$. We use the notation FT_m to denote the time that all tasks allocated to machine m have completed execution (that is, the finish time of the last task scheduled on machine m).

The optimization phase considers if moving any of the tasks from the machine giving the maximal finish time (i.e., the machine m for which FT_m is maximum) to some other machine might result in a smaller makespan overall; if such a task exists, then it is reallocated to that other machine that minimizes the makespan. This is an iterative procedure, which is repeated until there is no task, from those allocated to the machine giving the maximal finish time, whose reallocation to any other machine would make the overall makespan smaller. Three actions need to take place during any single iteration: (i) select a task from the machine giving the maximal finish time; (ii) select a machine to move this task to; and, (iii) assuming such a machine is found, insert this task to the list of tasks of that particular machine. Different options exist for making a selection in (i) and (ii) above. We experimented with some and we opted for a combination of simplicity and good overall efficiency. Thus, the selection of a task to be considered for possible moving to another machine is based on a statically pre-computed order for all the tasks. This order is the result of sorting in ascending order the average earliest finish time of each task across all machines. The average earliest finish time of a task i is computed as

$$\frac{1}{M}\sum_{j=1}^{M}(EST_{ij} + W_{ij}),$$

where M is the total number of machines. Note that this value is computed *only* to determine the order that tasks,

```
(1)  Assign each task to the machine that gives the fastest execution time
     For all machines
        create a task execution order list containing all tasks assigned to a given machine
        sort the list in ascending order of the earliest start time of each task on that machine
     endfor
(2)  Avg_FT[] ← vector of all tasks in ascending order of their average earliest finish time across all machines
     Repeat
        $m$ ← machine giving the maximal finish time, $MFT$
        moved_task ← false
        mark all tasks of machine $m$ in Avg_FT[] as unchecked and remaining tasks as checked
        While (there are unchecked tasks AND moved_task is false)
           $t$ ← next unchecked task in Avg_FT[]
           for all machines $i$ except machine $m$
              compute finish time, $FT'_i$, of each machine if task $t$ was to be inserted to the machine's list
           endfor
           $i$ ← machine with the smallest value of $FT'_i$
           if ($FT'_i < MFT$) then move $t$ to list of machine $i$; moved_task ← true
                else mark $t$ as checked
        endwhile
     until moved_task is false
```

Figure 4. The BMCT heuristic for scheduling independent tasks on heterogeneous machines.

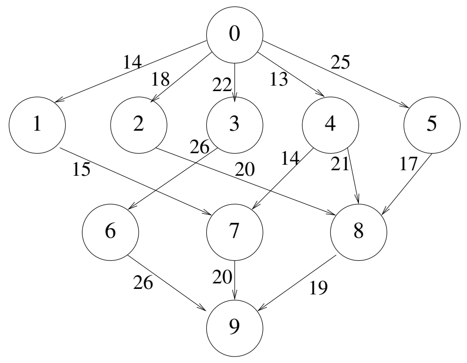
which are candidates for moving to another machine, will be considered and is not used for any other purpose. The selection of a machine to move this task to is based on an evaluation of the finish time obtained from all machines if this task was to run on each of them. The task is moved to the machine for which the machine's finish time becomes smallest, as long as the overall makespan is not increased (i.e., the finish time of this machine is not greater than the makespan obtained before moving the task to that machine). In other words, if FT_i is the time that all tasks of machine i are finished, assume that the maximal finish time, that is $\max(FT_1, FT_2, ..., FT_M)$, is given by machine 1, i.e., it is FT_1. A task is selected from machine 1 for possible relocation to another machine. To select what other machine to move the task to (if any), the finish time of each machine, if this task was to be added to it, is computed. If these values are given by FT'_i, then the task is moved to the machine that gives the minimum value (i.e., machine i for which FT'_i becomes minimum), as long as this value is smaller than the maximal finish time before, i.e., FT_1. Finally, since there are benefits in the makespan if the tasks of every machine are considered in ascending order of their EST value on that machine, a task that is reallocated to a machine will be inserted to the list of tasks of that machine at a point that will maintain a sorted order on the EST values. The algorithm terminates when none of the tasks of the machine giving the maximal finish time can be moved to any other machine.

The algorithm is outlined in Figure 4.

3.3. An Example

In order to illustrate both the above heuristics (hybrid and BMCT), consider the DAG shown in Figure 5(a). The cost to execute each of the 10 tasks in the graph on each of three different machines is given in Figure 5(b). The number next to each edge of the graph corresponds to the amount of data that needs to be passed from a task to an immediate successor. Figure 5(c) shows the cost to transfer a data unit for any given combination of machines; thus, the cost to transfer, for instance, the data needed from task 0 to task 1 would be 14×0.9 if one of the tasks was executed by machine 0 and the other by machine 1. Recall that we assume that the cost to transfer the data between two tasks that are executed on the same machine is zero.

The first phase involves the assignment of weights to the nodes and edges of the graph (using the average of all possible values) and then the computation of upward ranking for the nodes. The results are shown in Figure 6(a). The nodes in descending order of their ranking value are $\{0, 1, 4, 5, 7, 2, 3, 6, 8, 9\}$. The second phase involves partitioning of nodes into ordered groups, considering them in descending order of their ranking value. Node 0 is assigned to group 0. Node 1 cannot be in the same group as node 0 (since it depends on node 1), and, as a result, a new group (group 1) is created. Nodes 4 and 5 can also be in the same



(a) an example graph

node	m0	m1	m2	node	m0	m1	m2
0	17	19	21	5	30	27	18
1	22	27	23	6	17	16	15
2	15	15	9	7	49	49	46
3	4	8	9	8	25	22	16
4	17	14	20	9	23	27	19

(b) the computation cost for each node on three heterogeneous machines.

machines	time for a data unit
m0 - m1	0.9
m1 - m2	1.0
m0 - m2	1.4

(c) the communication cost table for interconnected machines.

Figure 5. An example of a DAG with computation and communication values.

node	weight	rank	node	weight	rank
0	19	149.93	5	25	95.40
1	24	120.67	6	16	58.07
2	13	85.60	7	48	85.67
3	7	84.13	8	21	57.93
4	17	112.93	9	23	23.00

(a) Ranking of each node using mean values to compute weights.

group	tasks
0	$\{0\}$
1	$\{1, 4, 5\}$
2	$\{7, 2, 3\}$
3	$\{6, 8\}$
4	$\{9\}$

(b) Partitioning the nodes into groups according to their rank values.

Figure 6. Ranking and partitioning the nodes of the example graph in Figure 5.

group as node 1 (that is group 1, since all three nodes are independent), but node 7 depends on node 4, therefore a new group needs to be created, and so on. When this procedure is completed, nodes are grouped in 5 groups as shown in Figure 6(b).

The third phase involves the scheduling of the *independent* tasks within each group, using the BMCT heuristic (after computing, each time, the EST array); the steps followed are shown in Figure 7. In order to help readability we have added the prefix n to each graph node, and m to each machine. Initially, group 0 is considered for scheduling; the values of EST are zero. According to the BMCT heuristic, node 0 is assigned to machine 0, which gives the best execution time for this task. Clearly, no reallocation of the task to another machine would give an earlier finish time. Then, the nodes of group 1 are considered for scheduling. First, the EST array is computed for each task/node and machine; its values depend on the finish time of any parent nodes (from previous groups) and the time needed to complete the transfer of all necessary data. Thus, for node 1, for example, the values for each different machine are $EST_{10} = 17$, $EST_{11} = 29.6$, $EST_{12} = 36.6$. Then, an initial assignment of nodes is made, based on which machine gives the best execution time; this is machine 0 for node 1, machine 1 for node 4, and machine 2 for node 5. Based on this assignment, the machine with the maximal finish time (MFT) is m2, with a value of 70. Node 5 can move to machine 0, since this would give a smaller MFT (69). After this, m0 has the highest MFT; n5 cannot move to any other machine, since this would increase the MFT, but, n1 can move to machine m2 giving an MFT of 60. Now, the only node of machine m2 (machine giving the MFT), n1, cannot move to another machine without increasing the MFT, thus the heuristic terminates. In the same way, the BMCT heuristic proceeds with each of the remaining three groups.

Group	Initial Assignment	MFT machine	Selected Node	Moving to	Final Assignment
0	{(n0, m0)}	m0	n0	fail	{(n0, m0)}
1	{(n1, m0), (n4, m1), (n5, m2)}	m2	n5	m0	{(n1, m0), (n5, m0), (n4, m1)}
	{(n1, m0), (n5, m0), (n4, m1)}	m0	n5	fail	
		m0	n1	m2	{(n1, m2), (n5, m0), (n4,m1)}
	{(n1, m2), (n5, m0), (n4,m1)}	m2	n1	fail	{(n1, m2), (n5, m0), (n4,m1)}
2	{(n7, m2), (n2, m2), (n3, m0)}	m2	n2	m1	{(n7, m2), (n2, m1), (n3, m0)}
	{(n7, m2), (n2, m1), (n3, m0)}	m2	n7	fail	{(n7, m2), (n2, m1), (n3, m0)}
3	{(n6, m2), (n8, m2)}	m2	n6	m0	{(n6, m0), (n8, m2)}
	{(n6, m0), (n8, m2)}	m2	n8	m1	{(n6, m0), (n8, m1)}
	{(n6, m0), (n8, m1)}	m1	n8	fail	{(n6, m0), (n8,m1)}
4	{(n9, m2)}	m2	n9	fail	{(n9, m2)}

Figure 7. Scheduling steps for the example in Figure 5.

4. Experimental Evaluation

4.1. The Setting

We have evaluated, separately, both the hybrid heuristic for DAG scheduling and the BMCT heuristic for scheduling independent tasks. When generating the array giving the execution time of each task on each of the heterogeneous machines, we adopted the approach used in [1, 3] to model the heterogeneity. Thus, we refer to *consistent heterogeneity* when any task that runs on a machine, say i, faster than another machine, say j, implies that the execution time of every task on machine i is faster than the execution time on machine j. We refer to *partially consistent heterogeneity* when the previous "consistency" property is true for only half of the tasks. Finally, we refer to *inconsistent heterogeneity* when no consistency is enforced and the execution time of a given task on a given machine is randomly generated without enforcing any rule for consistency. In all cases, the execution time of a task is chosen using a random uniform distribution over the interval [10,100].

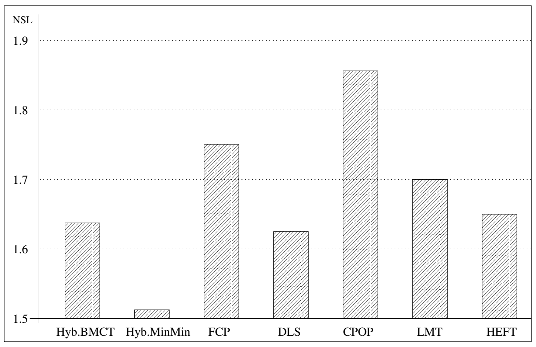
4.2. DAG Scheduling

In order to evaluate the performance of the hybrid heuristic for scheduling DAGs, proposed in the paper, we considered two possible implementations; these are based on the use of different heuristics for scheduling the independent tasks within each of the groups created. Thus, one implementation makes use of the BMCT heuristic; we refer to this implementation as Hyb.BMCT. The second implementation makes use of the Min-min heuristic [3] and we refer to it as Hyb.MinMin. We compared the performance of the two implementations with five other related heuristics for DAG scheduling: FCP [9], DLS [13], CPOP [14], LMT [6], HEFT [14]. The comparison was based on the *Normalized Schedule Length* (NSL) [8], defined as the ratio of the makespan divided by a fixed cost of the critical path. Four different types of DAGs were considered: Random Graphs, FFT [7, 8, 14], Fork-Join Graphs [8], and Laplace [10, 8]. Random graphs are generated using the procedure also explained in [16]. Thus, to generate a DAG with a number of nodes, we first generate a single entry and exit node; all other nodes are divided into levels, with each level having at least two nodes. Levels are created progressively; the number of nodes at each level is randomly selected up to half the number of the remaining to be generated nodes. Care is taken so that each node at a given level is connected to at least one node of the successor level and *vice versa*.

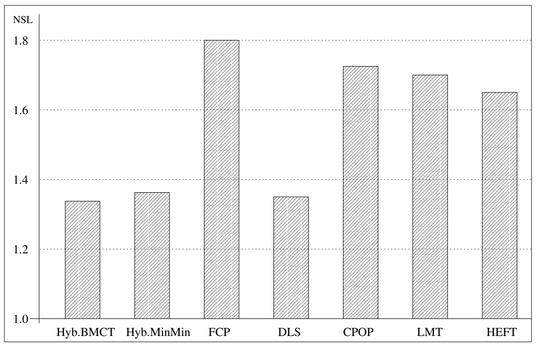
Taking into account the three different settings for the heterogeneity, a total of 12 comparisons are made; the results are shown in Figures 8, 9 and 10. In 8 cases, Hyb.BMCT returns the best NSL, in 2 of them Hyb.MinMin returns the best NSL, and, in the remaining two, DLS performs best. Neither the type of heterogeneity nor the application appear to make consistently a significant difference on the performance of the hybrid variants. Finally, from the five heuristics we used for comparison, DLS is the most competitive heuristic.

The average running time of all the heuristics, averaged over 100 runs using randomly generated DAGs, is shown in Figure 11. The results are in line with those from similar studies (see Fig.7 in [14]). Both Hyb.BMCT and Hyb.MinMin are faster than DLS (which is the slowest heuristic), with Hyb.BMCT providing better schedules too; FCP, HEFT, and CPOP are faster than Hyb.BMCT and Hyb.MinMin. It is also noted that Hyb.MinMin is only marginally faster than Hyb.BMCT even though the latter has higher complexity.

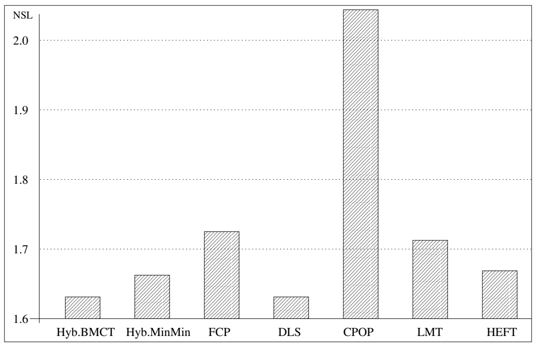
Finally, we ran both Hyb.BMCT and Hyb.MinMin using the six different methods to compute the weights men-



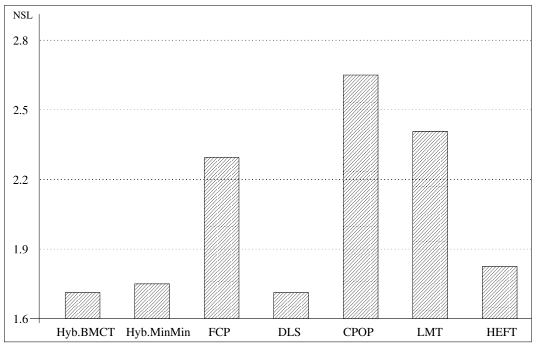
(a) Random graphs, 25-100 tasks



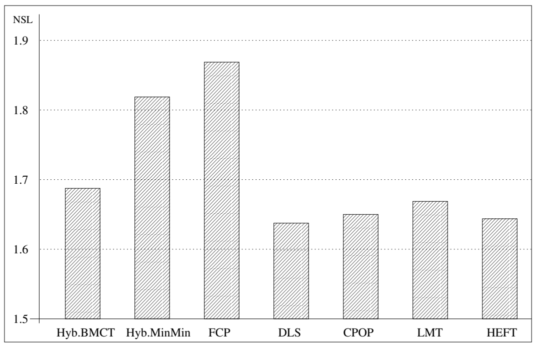
(a) Random graphs, 25-100 tasks



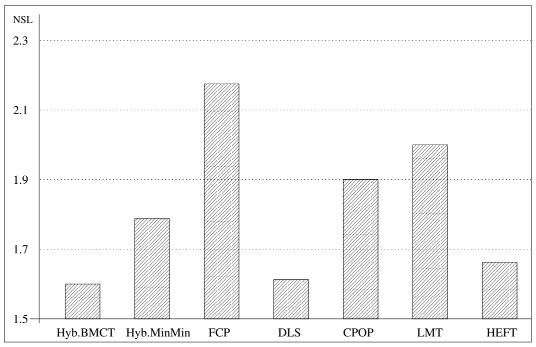
(b) FFT graphs, 15-223 tasks



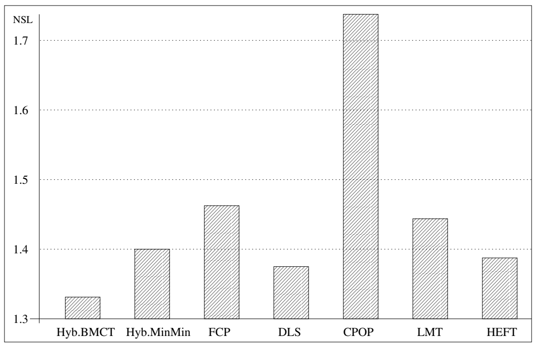
(b) FFT graphs, 15-223 tasks



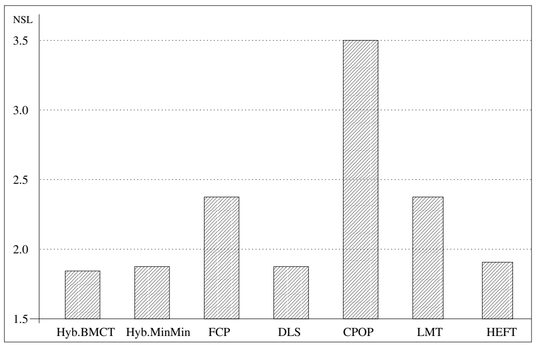
(c) Fork-join graphs, 7-229 tasks



(c) Fork-join graphs, 7-229 tasks



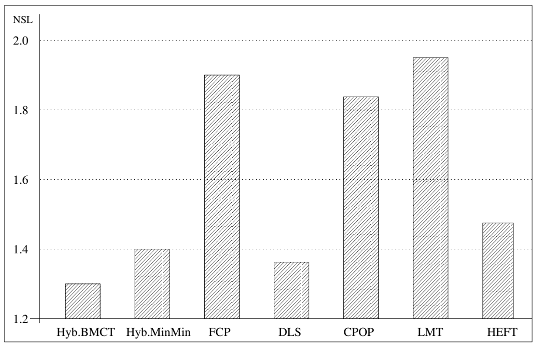
(d) Laplace graphs, 25-225 tasks



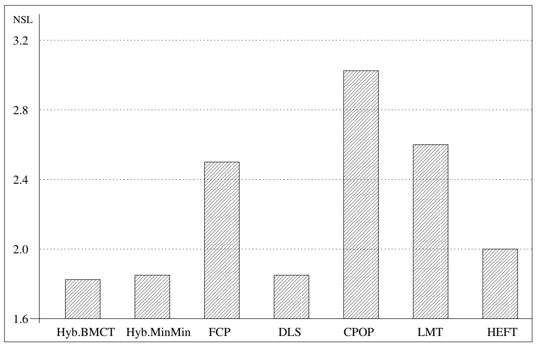
(d) Laplace graphs, 25-225 tasks

Figure 8. Average NSL of two versions of the Hybrid heuristic and 5 other DAG scheduling algorithms using 4 different application graphs, 3-8 machines and consistent heterogeneity.

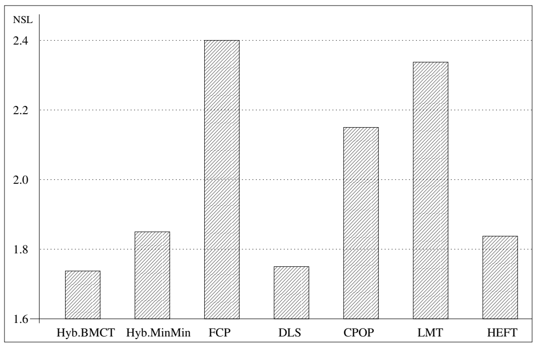
Figure 9. Average NSL of two versions of the Hybrid heuristic and 5 other DAG scheduling algorithms using 4 different application graphs, 3-8 machines and partially consistent heterogeneity.



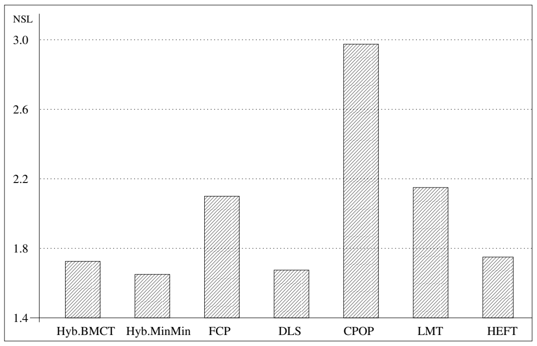
(a) Random graphs, 25-100 tasks



(b) FFT graphs, 15-223 tasks



(c) Fork-join graphs, 7-229 tasks



(d) Laplace graphs, 25-225 tasks

Figure 10. Average NSL of two versions of the Hybrid heuristic and 5 other DAG scheduling algorithms using 4 different application graphs, 3-8 machines and inconsistent heterogeneity.

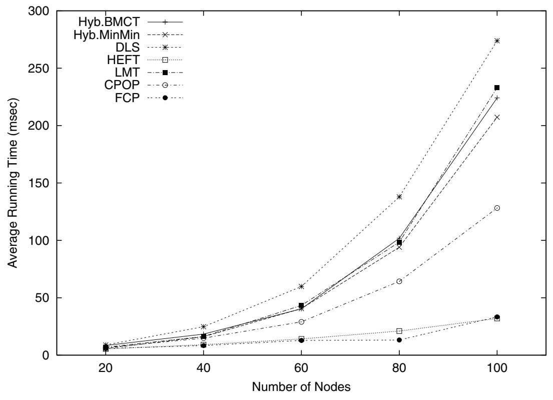
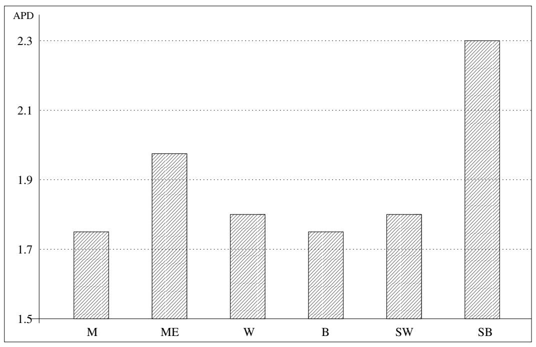
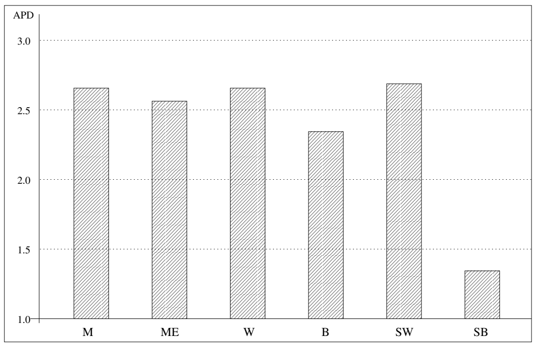


Figure 11. Average running time (over 100 runs on Randomly Generated DAGs) of Hyb.BMCT, Hyb.MinMin and the five heuristics considered in the evaluation in Figures 8, 9, and 10.



(a) Hyb.BMCT



(b) Hyb.MinMin

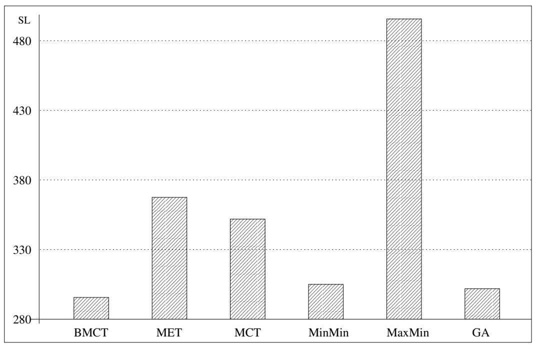
Figure 12. Average Percentage Degradation (APD) from the best schedule for each of six different methods to compute weights for Hyb.BMCT and Hyb.MinMin.

	Hybrid.BMCT		
	NB	*NEB*	*WPD*
M	169	177	9.7
ME	93	92	10.1
W	100	111	9.4
B	207	205	8.8
SW	91	109	9.3
SB	55	68	12.2

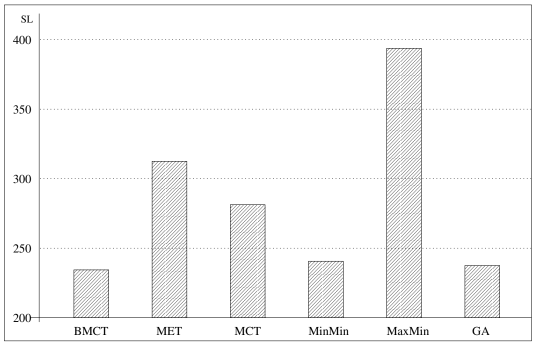
	Hybrid.MinMin		
	NB	*NEB*	*WPD*
M	20	238	10.7
ME	52	215	12.6
W	57	177	10.7
B	209	72	9.9
SW	22	213	11.9
SB	310	96	8.8

Figure 13. Three additional metrics for the experiments in Figure 12: Number of times a method gives the best schedule *NB*; number of times a method gives the best, but another one is also giving the best too *NEB*; and worst percentage degradation *WPD*, that is the maximum percentage degradation over all cases.

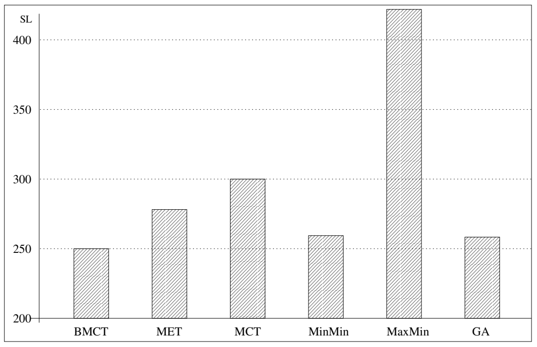
tioned in Section 2. We considered the Average Percentage Degradation (APD) [8, 16], over 1000 runs, of the makespan of each method from the best makespan (generated by any of the six different methods). The results are shown in Figure 12 and are directly comparable with the results of the other heuristics shown in Figure 1 (the same randomly generated DAGs were used). It can be seen that, in the worst case, the APD does not exceed 2.3 in the case of Hyb.BMCT and 2.62 in the case of Hyb.MinMin, whereas it was much higher for all the other DAG scheduling heuristics considered in Figure 1. This indicates that the hybrid heuristic is less sensitive to different approaches for computing the weights comparing to other heuristics. For completeness, the additional metrics shown in Figure 2 for the other heuristics, are also shown in Figure 13 for the two implementations of the hybrid heuristic. It is interesting to notice that the WPD was only 12.6% (comparing to much higher values for the heuristics considered in Figure 2); also, it appears that computing the weights using B or SB seems to result in slightly better schedules, but further investigation and more experiments would be needed before establishing anything here.



(a) Consistent heterogeneity, 50-200 tasks, 3-8 machines



(b) Partially Consistent heterogeneity, 50-200 tasks, 3-8 machines



(c) Inconsistent heterogeneity, 50-200 tasks, 3-8 machines

Figure 14. Comparison of the makespan achieved by the Balanced Minimum Completion Time (BMCT) heuristic and 5 other heuristics for independent task scheduling using consistent, partial-consistent and inconsistent heterogeneity.

4.3. Scheduling Independent Tasks

In order to evaluate the performance of the BMCT heuristic for scheduling independent tasks, we compared its performance with that of five other related heuristics for independent task scheduling [3]: Minimum Execution Time (MET), Minimum Completion Time (MCT), Min-min (MinMin), Max-min (MaxMin), and a Genetic Algorithm based heuristic (GA) (all five heuristics are described in [3]). The average schedule length (SL) over a 1000 different cases, using consistent, partially consistent and inconsistent heterogeneity is shown in Figure 14. A random uniform distribution is used to choose the number of tasks from the interval [50,200], the number of machines from the interval [3,8], and the execution time of each task from the interval [10,100]. In all cases, the BMCT heuristic performs better although it is noted that its success comes with an increased complexity (as a result of the phase that moves tasks across machines in an attempt to minimize the overall makespan) with respect to some of the other heuristics used in the comparison.

5. Conclusion

This paper presented two novel heuristics for task scheduling on heterogeneous machines: one heuristic for DAG scheduling, and another for scheduling independent tasks. Both heuristics have good performance behaviour. The first heuristic adopts a hybrid approach for DAG scheduling; by reducing the problem to smaller subproblems for scheduling independent tasks, it may be used to give rise to a separate new family of heuristics. The Balanced Minimum Completion Time heuristic proposed in this paper for scheduling independent tasks showed good behaviour both when used in the relevant phase of the hybrid heuristic and in comparison with other related heuristics. It is noted though that the success of the BMCT heuristic comes with an increased complexity. This may be less important if BMCT is used in the third phase of the hybrid heuristic for DAG scheduling (where the number of independent tasks to be scheduled is relatively small), but it might be significant for problems involving relatively large numbers of independent tasks. A more detailed study would be useful. Finally, additional evaluations using different applications, task execution values, etc. would provide further insight in the behaviour of the heuristics.

References

[1] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali. Task execution time modeling for heterogeneous computing systems. In *Heterogeneous Computing Workshop*, 2000.

[2] 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.

[3] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61:810–837, 2001.

[4] T. L. Casavant and J. G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering*, 14(2):141–154, 1988.

[5] B. Cirou and E. Jeannot. Triplet: A clustering scheduling algorithm for heterogeneous systems. In *International Conference on Parallel Processing Workshop*, 2001.

[6] M. Iverson, F. Ozguner, and G. Follen. Parallelizing existing applications in a distributed heterogeneous environment. In *Heterogeneous Computing Workshop*, 1995.

[7] Y. K. Kwok and I. Ahmad. Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 7(5):506–521, May 1996.

[8] Y. K. Kwok and I. Ahmad. Benchmarking and comparison of the task graph scheduling algorithms. *Journal of Parallel and Distributed Computing*, 59:381–422, 1999.

[9] A. Radulescu and A. J. C. van Gemund. On the complexity of list scheduling algorithms for distributed memory systems. In *ACM International Conference on Supercomputing*, 1999.

[10] A. Radulescu and A. J. C. van Gemund. Fast and Effective Task Scheduling in Heterogeneous Systems. In *Heterogeneous Computing Workshop*, 2000.

[11] A. Radulescu and A.J.C. van Gemund. Low-Cost Task Scheduling for Distributed-Memory Machines. *IEEE Transactions on Parallel and Distributed Systems*, 13(6), pp. 648-658, June 2002.

[12] S. Ranaweera and D. P. Agrawal. A task duplication based scheduling algorithm for heterogeneous systems. In *14th International Parallel and Distributed Processing Symposium (IPDPS'00)*, Cancun, Mexico, May 2000.

[13] G. C. Sih and E. A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architecture. *IEEE Transactions on Parallel and Distributed Systems*, 4(2):175–187, February 1993.

[14] H. Topcuoglu, S. Hariri, and M. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, March 2002.

[15] 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.

[16] 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.

Biographical Note

Rizos Sakellariou received his PhD degree in Computer Science from the University of Manchester in 1997 for a thesis on symbolic analysis techniques with applications to loop partitioning and scheduling for parallel computers. Since January 2000 he has been a Lecturer in Computer Science at the University of Manchester. He was a postdoctoral research associate at Rice University and has held visiting appointments with the University of Cyprus, the University of Illinois at Urbana-Champaign, and the Universitat Politecnica de Catalunya. His research interests are in the areas of parallel and distributed systems, performance evaluation, modelling and prediction, scheduling for parallelism, optimizing and parallelizing compilers, and Grid and Internet computing.

Henan Zhao received the BSc degree in Software Engineering from the University of Manchester, UK, in 2001. She is currently a PhD student in the Computer Science Department, the University of Manchester. Her research interests include task scheduling for heterogeneous and Grid environments, workflow scheduling algorithms, performance modeling, reservation scheduling and divisible scheduling.