2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing

DAG Scheduling Using a Lookahead Variant of the Heterogeneous Earliest Finish Time Algorithm

Luiz F. Bittencourt
Institute of Computing
University of Campinas
P.O. 6176, Campinas
São Paulo, Brazil
bit@ic.unicamp.br

Rizos Sakellariou
School of Computer Science
University of Manchester
Oxford Road, Manchester
M13 9PL, UK
rizos@cs.man.ac.uk

Edmundo R. M. Madeira
Institute of Computing
University of Campinas
P.O. 6176, Campinas
São Paulo, Brazil
edmundo@ic.unicamp.br

***Abstract*—Among the numerous DAG scheduling heuristics suitable for heterogeneous systems, the Heterogeneous Earliest Finish Time (HEFT) heuristic is known to give good results in short time. In this paper, we propose an improvement of HEFT, where the locally optimal decisions made by the heuristic do not rely on estimates of a single task only, but also look ahead in the schedule and take into account information about the impact of this decision to the children of the task being allocated. Preliminary simulation results indicate that the lookahead variation of HEFT can effectively reduce the makespan of the schedule in most cases without making the algorithm's execution time prohibitively high.**

I. Introduction

Directed Acyclic Graphs (DAGs) are frequently used to capture the precedence constraints of a number of applications. When such applications are executed on parallel computing resources, a decisive factor determining an application's performance is the order that independent tasks (that is, tasks which are not constrained by precedence constraints) will be scheduled onto resources. In its most general form, the problem of scheduling tasks of a graph onto a set of different machines is an NP-Complete problem [1]. As a result, over several years, a number of heuristics have been developed that attempt to strike a good balance between running time, complexity and schedule quality [2].

In recent years, there has been an increasing interest in DAG scheduling heuristics for heterogeneous resources, that is, resources whose capabilities may differ (as opposed to traditional homogeneous resources, such as multiprocessors). Part of the motivation for this work is the emergence of heterogeneous processing platforms, such as the Grid, and the increasing interest in a number of applications that can be modelled by a DAG, such as workflows [3], [4]. A number of heuristics suitable for DAG scheduling on heterogeneous resources have been suggested; a comparative evaluation of 20 different heuristics can be found in [5]. Among these heuristics, the *Heterogeneous Earliest Finish Time* heuristic (or HEFT in short) [6] has been one of the most often cited and used, having the advantage of simplicity and producing generally good schedules with a short makespan.

HEFT is essentially a *list scheduling* [1] heuristic that constructs first a priority list of tasks and then makes locally optimal allocation decisions for each task on the basis of the task's estimated finish time. However, a locally optimal decision for a given task may not necessarily be a good decision for the task's descendants in the DAG. The key idea of this paper is to enhance the process of scheduling tasks in HEFT, by *looking ahead* and considering information about the descendants of a task. Then, scheduling decisions for each task are made on the basis of what appears to be beneficial not only for the task itself, but both for the task and its children. Clearly, such an enhancement has the potential of increasing HEFT's execution time. However, HEFT is already a simple and fast algorithm and, as we demonstrate in our simulation experiments, such an increase is affordable.

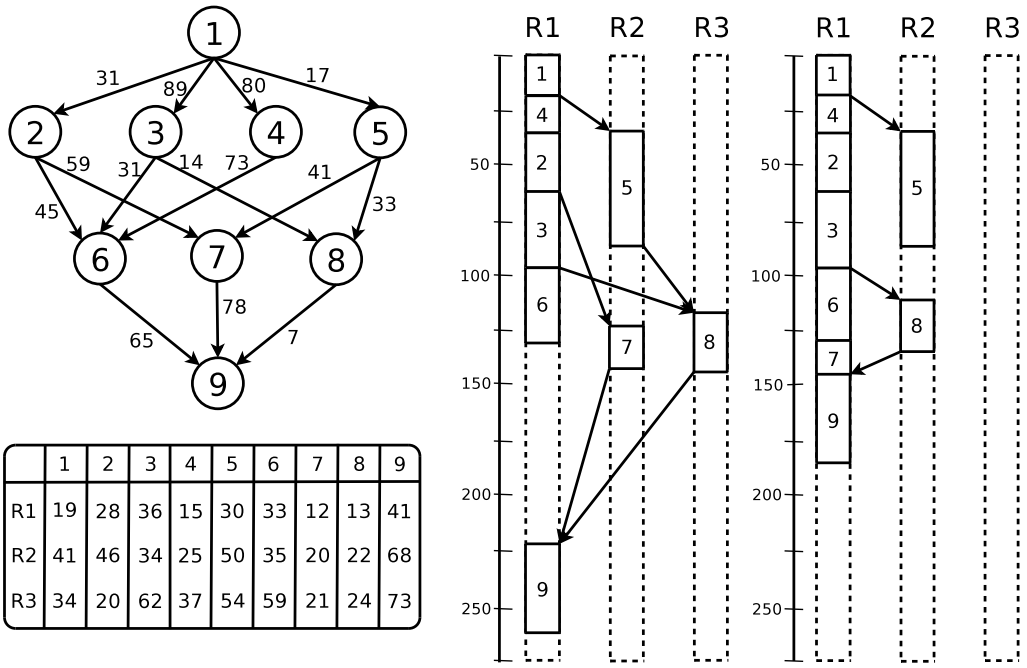
Thus, the contribution of this paper is the proposal and investigation of specific improvements for HEFT, which allow HEFT to make more informed scheduling decisions by employing the notion of *lookahead* to obtain extra information before allocating each task. As we demonstrate in our simulation experiments, using five different types of DAGs corresponding to real-world workflow applications, our suggested improvements, which are based on minimal lookahead information, may shorten the makespan in some cases by up to 20% on average.

The rest of the paper is structured as follows. Section II introduces the problem, presents related work and shows a motivating example, which highlights the potential benefits of our proposal. In Section III, the proposed enhancements of HEFT are presented, which are then evaluated in Section IV. Finally, Section V concludes the paper.

II. Background

The application model used in this paper is a directed acyclic graph (DAG), where each node is a task and each edge represents a data dependence. Each task can only start its execution after receiving all data from its parents. The target environment is a set of heterogeneous computing resources fully connected by heterogeneous links. Computation costs for each task and the amount of data communication between two dependent tasks are given. Data transfer costs for each resource pair are also given, as well as the computation capacity of each resource. Every task can be executed on every resource. The scheduling

1066-6192/10 $26.00 © 2010 IEEE
DOI 10.1109/PDP.2010.56



	1	2	3	4	5	6	7	8	9
R1	19	28	36	15	30	33	12	13	41
R2	41	46	34	25	50	35	20	22	68
R3	34	20	62	37	54	59	21	24	73

Figure 1. An example illustrating the benefits of HEFT with lookahead information.

problem examined here consists in selecting resources for each task aiming at minimizing the overall execution time of the application (makespan), always obeying the precedence constraints imposed by the DAG.

DAG scheduling is a widely studied problem in both homogeneous and heterogeneous systems; for surveys and representative work addressing this problem we refer to [2], [7], regarding DAG scheduling in homogeneous systems, and [5], [6], [8], [9] regarding DAG scheduling in heterogeneous systems. Among the algorithms suitable for the latter problem, HEFT [6] has been one of the most frequently cited and used, owing this both to its simplicity and good performance.

HEFT belongs to the family of *list scheduling* [1] heuristics. Thus, HEFT assigns first a weight to each node and edge of the DAG. The weight for each node is calculated as the average computation cost to execute this task on every machine, while the weight for each edge is calculated as the average communication cost over all possible pairs of machines. A ranking value, $rank_u$, is computed by traversing the DAG upwards, with the $rank_u$ of a task t being t's weight plus the maximum value resulting from the weight of each child of t added to the weight of the edge connecting this child with t. Then, the tasks are scheduled in descending order of their $rank_u$ value on the resource which gives the smallest estimated finish time (EFT) for each task. Although, as demonstrated in [8], [10], HEFT is sometimes particularly sensitive to the ranking values produced using average values to compute the weights, it is not immediately clear under what circumstances other options to compute the weights may produce better quality schedules.

Similarly to HEFT, many DAG scheduling heuristics select resources based solely on attributes calculated for a single, selected task, ignoring any adverse consequences of such a scheduling decision to the task's successors. A notable exception to this rule is the k-DLA algorithm [11], where a statically computed attribute takes into account the number of neighbours of a resource in the network as well as the number of children of the task being scheduled. Our proposal takes this further by building upon HEFT and looking ahead as the schedule is generated, thereby considering the impact of every task's scheduling decision separately.

To illustrate the possible benefits, consider the simple DAG shown in Figure 1. The cost of executing each task on three different (heterogeneous) resources is given in the table below the graph. It is assumed that the three resources are connected with communication links of the same capacity. Thus, the communication requirements between tasks are only determined by the amount of data sent between tasks (shown next to each edge of the DAG). Two schedules are shown on the right-hand side of the figure. The first illustrates the schedule produced by the standard version of HEFT. The second illustrates the schedule produced by an improved version of HEFT, which uses lookahead information to assess the impact of each allocation decision to a task's children. The second schedule has a makespan, which is approximately 30% less than the makespan produced by HEFT (184 as opposed to 260 time units). Following ranking, the priority list of tasks used in our motivating example is $\{1, 4, 2, 3, 5, 6, 7, 8, 9\}$. The improvement is achieved when task 7 is scheduled. Without any lookahead information, the smallest EFT for task 7 is 141 on $R2$; then, task 8 gives a smallest EFT on $R3$ (143), and task 9 has the smallest EFT on $R1$ (260). When using lookahead information, task 7 is scheduled on

the resource that gives the smallest EFT for its children, in this case task 9 only.

This outcome can be achieved by assigning (provisionally) task 7 on each resource and attempting to schedule its child (that is, task 9) using HEFT. When task 7 is assigned to $R2$ its EFT is 141, and the best resource for task 9 would be $R1$ (which returns an EFT of 260). On the other hand, when task 7 is assigned to $R1$, although it has a worse EFT (143), the EFT for its child, task 9, would be 184, which is smaller than before (260). Thus, the algorithm decides to schedule task 7 on a resource which gives a worse EFT for this task, but which can reduce the EFT of its child. Then, task 8 is scheduled on $R2$, which results in the best EFT for task 9 in $R1$ (184). Finally, task 9 is scheduled on the resource which gives the best EFT for itself, since it does not have children.

Clearly, our proposal increases the time complexity of the original HEFT. However, we believe the potentially significant benefits resulting from this increase (around 30% in the example above and up to 20% when averaged over several runs – see Section 4) are justified in practice because: (i) HEFT is already a simple and quick heuristic (for example, a standard HEFT implementation can generate a schedule for DAGs with 100s of nodes in less than a second); (ii) as indicated in [5], there are heuristics with a time complexity higher than HEFT, which, however, do not necessarily generate makespans as good as those generated by HEFT.

III. HEFT WITH LOOKAHEAD

We present four different possibilities to enhance HEFT with lookahead information. The four different versions correspond to all combinations of: (i) two different ways to select a resource for each task (based on two different approaches to calculate the EFT of the children of the task); and (ii) two different ways of selecting which task will be scheduled first (using either strictly the priority order calculated by HEFT's $rank_u$ attribute to rank tasks, or partly relaxing this ranking, motivated by the observations in [8], [10]).

The four versions studied are as follows.

1) *Lookahead*: this is the lookahead as described in the motivating example above, where the resource selected for a task t is the one which minimizes the maximum EFT from all t's children on all resources where t is tried.
2) *Lookahead with weighted average EFT*: the resource selected for t is the one which minimizes a weighted average EFT from all t's children, where the average is weighted using the $rank_u$ of each task.
3) *Lookahead and priority list change*: Provided that the first two unscheduled ready tasks (as ranked by HEFT) are independent, they are each considered, in turn, for allocation making use of lookahead information for the children of both. The task which was scheduled first when the maximum EFT was minimized is the one finally scheduled.
4) *Lookahead and priority list change with weighted average EFT*: same as the priority list change above, but using the weighted average EFT instead of EFT.

A. Lookahead

To implement the lookahead only versions (first two versions of the four versions listed above), we use the standard description of HEFT, including the calculation of a priority attribute $rank_u$ for each task t_i, defined as follows (see [6]):

$$rank_u(t_i) = \overline{w_i} + \max_{t_j \in succ(t_i)} (\overline{c_{i,j}} + rank_u(t_j)),$$

where $\overline{w_i}$ is the average execution time of t_i over all resources, $succ(t_i)$ is the set of immediate successors of t_i, and $\overline{c_{i,j}}$ is the average communication cost between t_i and t_j.

The lookahead version raises some issues that need to be tackled. Let t be the highest priority task ready to be scheduled. Task t is tried on every resource and its children are scheduled using HEFT to calculate their estimated finish time according to where t is scheduled. However, some (or all) children of t may not become ready immediately after scheduling t (as a result of a dependence to another, as yet unscheduled, parent). Still, to calculate the estimated finish time of t's children, we only consider their already scheduled parents and t, ignoring further delays that may arise due to any unscheduled parents. This means that the estimated finish time computed is rather optimistic.

Another issue is that one can think of several ways to select a resource for a task t based on the estimated finish time of its children. This may be because, for example, some of the children may have a low priority comparing to other children which are on the critical path. In this paper, we consider and evaluate two alternative ways:

1) Select the resource which minimizes the maximum EFT for t's children among all resources where t is tried. In other words, this means that t will be scheduled on the resource which gives the minimum completion time for all t's children, which were scheduled using HEFT.
2) Select the resource which minimizes the weighted average EFT, EFT_W, for t's children. The rationale of the weighted average is that the EFT of children with a low ranking value (that is, children that are expected to be scheduled the furthest in the future) is given a lower weight. The weighted average EFT_W for a task t is computed as follows:

$$EFT_W = \frac{\sum_{t_j \in L} (rank_u(t_j) \times EFT_{t_j})}{\sum_{t_j \in L} rank_u(t_j)},$$

where L is the *lookahead set* containing the tasks which are used to look ahead. In the lookahead version described here, L contains all children of t. When the set L is empty, t is scheduled on the resource which gives the best EFT for itself.

Algorithm 1 HEFT with lookahead

1: rank tasks using the $rank_u$
2: **while** there are unscheduled tasks **do**
3: $t \leftarrow$ unscheduled task with highest $rank_u$
4: $L \leftarrow$ children of t
5: **for all** resource r_i in the resources set R **do**
6: schedule t on r_i
7: schedule all tasks of L using HEFT
8: $EFT_{r_i} \leftarrow$ maximum EFT for tasks in L
9: return to the schedule state at the beginning of this loop
10: **end for**
11: schedule t on r_i such that $EFT_{r_i} \leq EFT_{r_k} \forall r_k \in R$
12: **end while**

The version based on the minimization of the maximum EFT values is shown in Algorithm 1. The version that makes use of the weighted average requires a modification in line 8 to assign EFT_W to EFT_{r_i}.

It is easy to observe that the time complexity of the lookahead algorithm increases the time complexity of HEFT by a factor of $r \times c$, where r is the number of resources and c is the average number of children per task. The algorithm can also be modified to look ahead recursively, achieving a higher depth of lookahead, by checking not only the children of a task but the children of the children and so on. Every additional level increases the complexity significantly (by another factor of $r \times c$). In addition, some preliminary simulation results indicated that there was not much to gain in terms of the makespan by considering a lookahead with depth of 2 and 3. As a result, in this paper we restrict our description and the discussion to a lookahead of depth 1 (that is, only the task's children and not their children too).

B. Lookahead and priority list change

Motivated by the observations made in [8], [10] (where it was demonstrated that, sometimes, relaxing the priority order of tasks, obtained through ranking, may give better quality schedules), we developed a version that may change the order of the first unscheduled task, t, with its successor task, t_1, in the priority list, depending on the outcome of lookahead information. Of course, it is assumed that these two tasks, t and t_1, are ready for execution (that is, all their parents have completed execution) and they are independent (that is, their execution order is not bound by a precedence constraint).

The idea behind this version is to look ahead also in the priority list, in addition to looking ahead for children's completion time, described hitherto. This version works in a similar way to Algorithm 1, but with two differences:

(i) in addition to the children of the first ready task, t, the set L includes the second ready task with highest priority, t_1, and the children of t_1;
(ii) the inner loop of Algorithm 1 (lines 5-10) is executed twice: first considering t as the first task to be scheduled and then considering t_1.

The schedule which gives the best overall result is then chosen.

Algorithm 2 HEFT with lookahead and priority change

1: rank tasks using the $rank_u$
2: **while** there are unscheduled tasks **do**
3: $t \leftarrow$ unscheduled task with highest $rank_u$
4: $t_1 \leftarrow$ unscheduled task with second highest $rank_u$
5: **for** $(i = 0; i < 2; i++)$ **do**
6: $L \leftarrow$ (children of t) $\cup$ t_1 $\cup$ (children of t_1)
7: **for all** resource r_i in the resources set R **do**
8: schedule t on r_i
9: schedule all tasks of L using HEFT
10: $EFT_{t,r_i} \leftarrow$ maximum EFT for tasks in L
11: return to the schedule state in the beginning of this loop
12: **end for**
13: $resource_t \leftarrow r_i$ such that $EFT_{t,r_i} \leq EFT_{t,r_k} \forall r_k \in R$
14: $EFT_t \leftarrow \min_{r_i \in R} EFT_{t,r_i}$
15: return to the schedule state in the beginning of this loop
16: $aux \leftarrow t$; $t \leftarrow t_1$; $t_1 \leftarrow aux$
17: **end for**
18: select t which resulted in the smallest EFT_t
19: schedule t on $resource_t$
20: **end while**

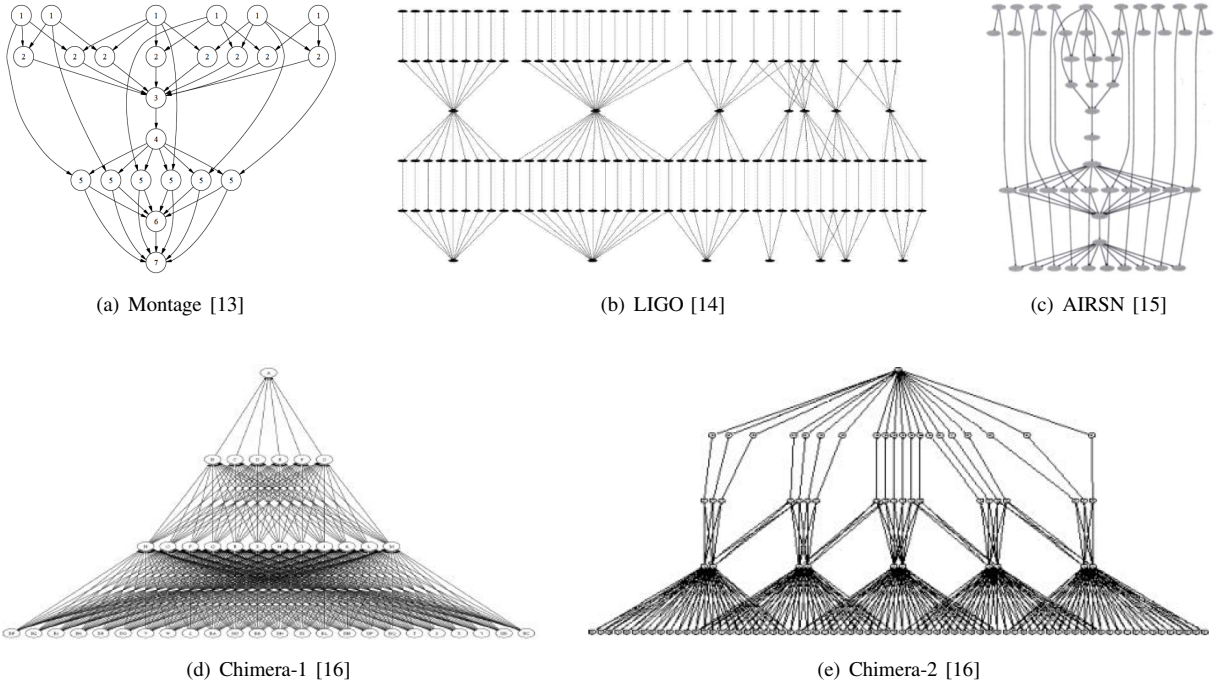
A version of this algorithm, which minimizes the completion time of all tasks in L, is illustrated in Algorithm 2. Line 10 can be changed to use the weighted average approach to compute a weighted average EFT_W. The algorithm works in a similar way to Algorithm 1, the only difference being that runs twice (considering a different task each time) and checks the children of both tasks under consideration.

IV. Simulation Results

We carried out extensive simulation with an in-house simulator developed in Java and used in other similar studies (e.g., [12]). In the simulation, it is assumed that the DAGs have dedicated use of the resources. We measured the average makespan for the lookahead and priority list change, both with the minimize maximum EFT and weighted average variations. Additionally, we measured the execution time of each algorithm. The simulations were run using 2 and 10 heterogeneous resources in the target environment, each one having a processing capacity (units of computation per time unit) randomly taken from the interval $(10, 100)$. The resources were fully connected by a heterogeneous network; each link had its bandwidth randomly taken from the interval $(10, 100)$.

Five different types of DAGs were used, each corresponding to a real-world workflow application. They are:

- Montage: 24 tasks; see Figure 3 in [13]
- AIRSN: 51 tasks; see Figure 5 in [15]
- LIGO: 166 tasks; see Figure 4 in [14]



(a) Montage [13] (b) LIGO [14] (c) AIRSN [15]

(d) Chimera-1 [16] (e) Chimera-2 [16]

Figure 2. Workflow applications used in the simulations.

- Chimera-1: 43 tasks; see Figure 6 in [16].
- Chimera-2: 123 tasks; see Figure 7 in [16].

The versions of the five DAGs used are reproduced in Figure 2.

The computation cost of each node was randomly taken from the interval $(500, 4000)$. For each type of DAG, we performed simulations using communication to computation ratios (CCR) of 0.5, 1.0, and 2.0. CCR is defined as the ratio between the amount of communication and the amount of computation performed in the DAG execution.

A. Results

The results are averaged over 500 runs and, along with some observations for each type of DAG, are presented next.

Montage: The average makespan results for the Montage DAG are shown in Figure 3. The highest makespan improvements over HEFT were obtained by the lookahead algorithm using the minimize maximum EFT for all children approach. Improvements range from 1.55%, with 2 resources and $CCR = 0.5$, to 15.2%, with 10 resources and $CCR = 2.0$.

AIRSN: The results for the AIRSN DAG are shown in Figure 4. The priority list change algorithm using the weighted average approach performed better with the AIRSN DAG with 2 resources and $CCR = 0.5$ (improvement of 2.85%) or $CCR = 1.0$ (improvement of 3.93%). With 2 resources and $CCR = 2.0$, the lookahead algorithm performed slightly better than the priority change, improving the makespan by 5.88%. With 10 resources, the results are similar for the lookahead and priority change when both are using the weighted average approach, with improvements of around 6%.

LIGO: The results for the LIGO DAG are shown in Figure 5. The makespan is similar for all algorithms with 2 resources, even though, when $CCR = 2$, algorithms using the weighted average approach performed slightly better; the improvement is low, around 0.25%. For simulations with 10 resources, when $CCR = 0.5$ the priority list change using weighted average was marginally better, improving the schedule of HEFT by 0.77%. With $CCR = 1.0$ both algorithms with the weighted average approach gave similar results, with improvement around 2%, while with $CCR = 2.0$ the lookahead with weighted average performed better than the other algorithms, resulting in an average makespan 7.2% lower than HEFT's makespan.

Chimera-1: The results for the Chimera-1 DAG are shown in Figure 6. When the CCR is 0.5 or 1.0 the results are similar for all four proposed algorithms. With 2 resources and $CCR = 0.5$ the improvement against HEFT is around 1.7%, while for $CCR = 1.0$ the improvement is around 3.5%. Results for 10 resources give an improvement of around 5.2% with $CCR = 0.5$ and 11% when $CCR = 1.0$. When the CCR is higher, the makespan improvement of the lookahead algorithm with weighted average is around 20%.

Chimera-2: The results for the Chimera-2 DAG are shown in Figure 7. They show similar trends to the results of Chimera-1. The best improvement with 2 resources is given by the lookahead with weighted average when $CCR = 2.0$ (2.6%); with 10 resources the maximum improvement is around 14.4% with $CCR = 2.0$.

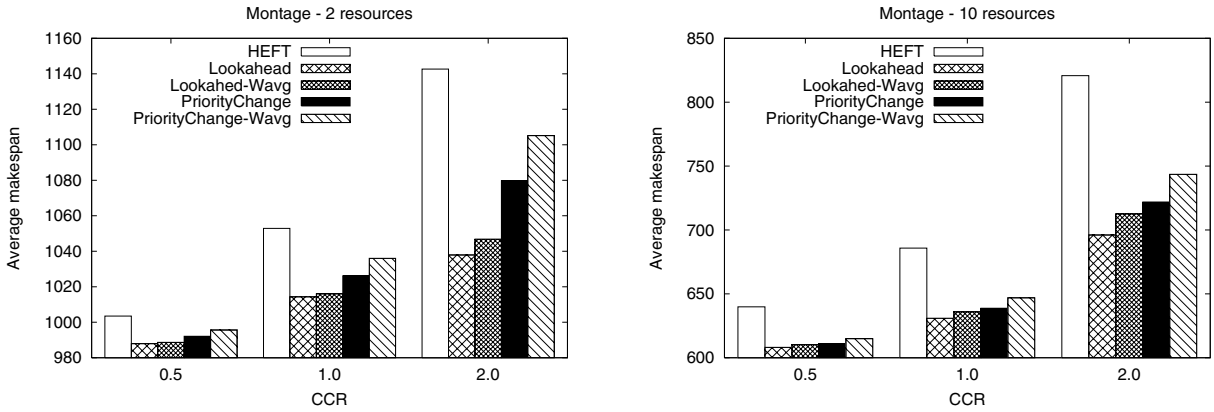


Figure 3. Average makespan using the Montage graph and 2 or 10 resources.

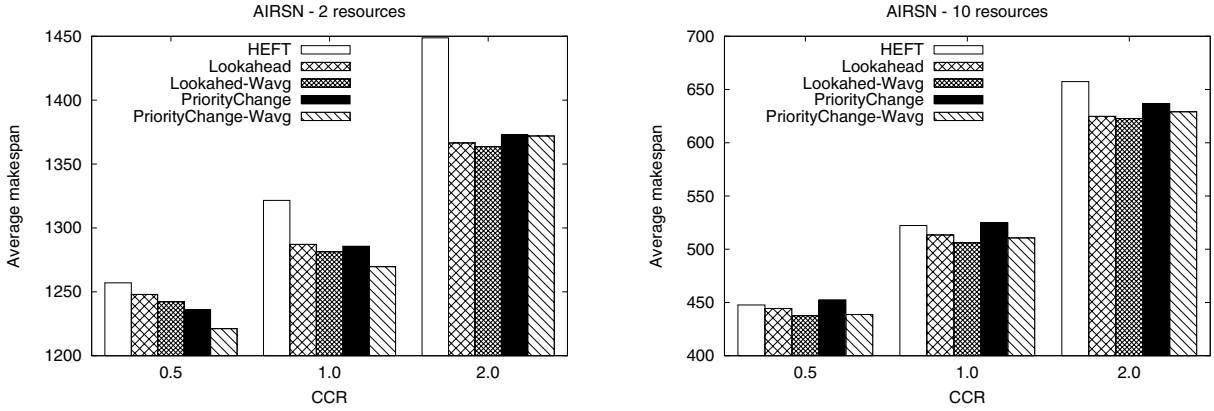


Figure 4. Average makespan using the AIRSN graph and 2 or 10 resources.

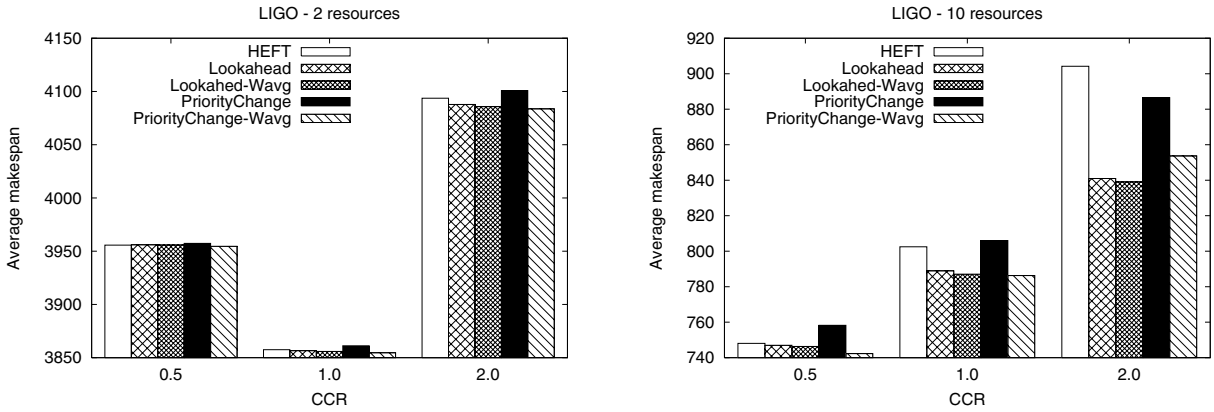


Figure 5. Average makespan using the LIGO graph and 2 or 10 resources.

B. Discussion

Analysing the results for each DAG we observe that the proposed algorithms appear to give better results when there are more resources and when the CCR is increased. The motivating example presented in Section II gives a hint that may explain this behaviour: when looking ahead for task 7, the algorithm found that the communication between task 7 and task 9 had to be prioritised instead of the EFT of task 7. Thus, when the CCR is high and there are more resources where tasks can be assigned to, lookahead information can foresee whether future data transfers may be too costly and thus avoid them.

A second observation is that the priority list change versions do not seem to give significantly better results despite the additional complexity. Many times, their performance seems to be worse than the performance of the lookahead alone versions. This is an interesting observation, because it seems to indicate that considering

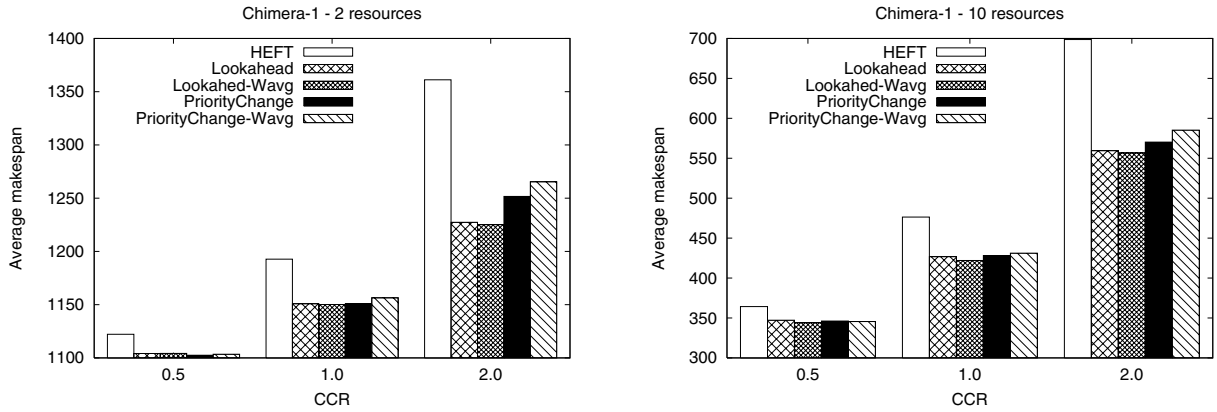


Figure 6. Average makespan using the Chimera-1 graph and 2 or 10 resources.

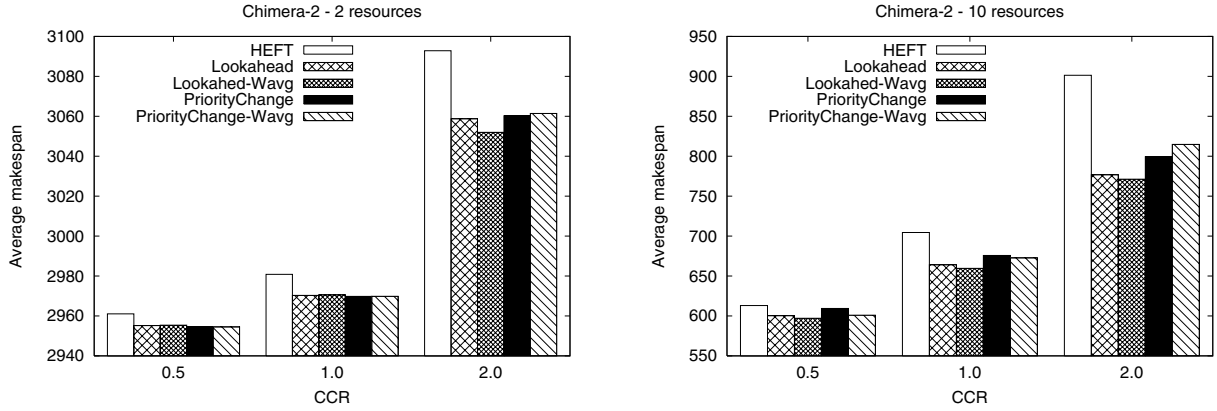


Figure 7. Average makespan using the Chimera-2 graph and 2 or 10 resources.

the impact of scheduling decisions on children alone is sufficient as a remedy to the problems observed in [8], [10], that motivated the idea of partially relaxing the priority order of tasks.

C. Algorithm Execution Time

An important point to evaluate with the proposed algorithms is how long they take to run; since they consider a wider search space than HEFT, they have a higher time complexity. The execution time of each algorithm for simulations with 10 resources is shown in Figure 8. As expected, the lookahead versions took more time to run than HEFT, and the priority change versions took even more time than the lookahead. Furthermore, as expected, a high number of edges results in higher execution times. For example, the Chimera-1 DAG has a high edge density; the running time of the algorithms for Chimera-1 is much higher than for AIRSN even though the latter has more tasks. Nevertheless, in absolute terms, as the measured execution times demonstrate, none of the algorithms has a prohibitively high execution time when compared to HEFT. The execution time for the lookahead alone (no priority list) versions is at most four times more than HEFT for Chimera-2, which has a high edge density; still, in absolute terms, this was less than 0.5 seconds, which

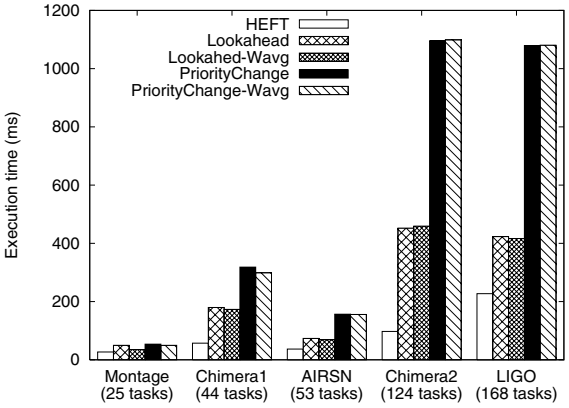


Figure 8. Algorithms execution time.

is not prohibitively high and makes it attractive for real practice (especially with regards to the expected benefits in the makespan). For other DAGs with a smaller edge density, the lookahead version was twice as slow as HEFT.

V. Conclusion

This paper presented lookahead approaches to schedule DAGs in heterogeneous systems, which are implemented as an extension of the popular *Heterogeneous Earliest*

Finish Time (HEFT) algorithm. The first approach proposed uses lookahead information from the DAG, trying to minimize the estimated finish time of the children of the task being scheduled. The second approach looks ahead in the priority list too, changing the order of the tasks being scheduled to see which order can give a better estimated finish time.

Preliminary simulations with a number of DAGs corresponding to real-world applications have indicated that the lookahead versions proposed can significantly improve the schedule returned by HEFT, especially in cases where the communication cost is higher with respect to computation, without resulting in a prohibitively high running time. The first approach proposed seems to give results, which are as good as the results of the second approach, with lower complexity, and consequently lower running time. The proposed lookahead algorithms may shorten the makespan in some cases by up to 20% on average. Such an improvement is important because, despite the increased complexity, HEFT performs better than many higher time complexity algorithms. In addition, thanks to its simplicity and good performance, it appears to be one of the most frequently used DAG scheduling heuristics, which makes it worth investigating situations where enhancements of the original algorithm may lead to improved schedules.

Future work can examine different approaches to look ahead as well as further work on the feasibility of a higher lookahead depth. Another point for future investigation is the effect of lookahead strategies to other similar DAG scheduling heuristics and the optimization of the lookahead strategies so that they only selectively consider lookahead information.

ACKNOWLEDGEMENT

This work was carried out while the first author was on a 10-month visit to the University of Manchester, with support from CAPES (3248-08-9) and FAPESP (05/59706-3), which is gratefully acknowledged.

REFERENCES

[1] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, 2008.

[2] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Surveys*, vol. 31, no. 4, pp. 406–471, 1999.

[3] I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, *Workflows for e-Science. Scientific Workflows for Grids*. Springer, 2007.

[4] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-science: An overview of workflow system features and capabilities," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 528–540, 2009.

[5] L.-C. Canon, E. Jeannot, R. Sakellariou, and W. Zheng, "Comparative Evaluation of the Robustness of DAG Scheduling Heuristics," in *Integrated Research in Grid Computing, CoreGRID Integration Workshop*, Greece, April 2008, pp. 63–74.

[6] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing." *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[7] A. S. Wu, H. Yu, S. Jin, K.-C. Lin, and G. Schiavone, "An Incremental Genetic Algorithm Approach to Multiprocessor Scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 9, pp. 824–834, 2004.

[8] R. Sakellariou and H. Zhao, "A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems." in *13th Heterogeneous Computing Workshop*. IEEE Computer Society, 2004, pp. 111–123.

[9] J.Yu, R. Buyya, and K. Ramamohanarao, "Workflow scheduling algorithms for grid computing." *Studies in Computational Intelligence*, vol. 146, pp. 173–214, 2008.

[10] H. Zhao and R. Sakellariou, "An Experimental Investigation into the Rank Function of the Heterogeneous Earliest Finish Time Scheduling Algorithm," in *9th International Euro-Par Conference*, Klagenfurt, Austria, 2003, pp. 189–194.

[11] N. Woo and H. Y. Yeom, "k-Depth Look-Ahead Task Scheduling in Network of Heterogeneous Processors," in *International Conference on Information Networking, Wireless Communications Technologies and Network Applications (ICOIN'02)*. Cheju Island, Korea: Springer-Verlag, 2002, pp. 736–745.

[12] L. F. Bittencourt and E. R. M. Madeira, "A performance-oriented adaptive scheduler for dependent tasks on grids," *Concurrency and Computation : Practice and Experience*, vol. 20, no. 9, pp. 1029–1049, 2008.

[13] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.

[14] A. Ramakrishnan, G. Singh, H. Zhao, E. Deelman, R. Sakellariou, K. Vahi, K. Blackburn, D. Meyers, and M. Samidi, "Scheduling data-intensive workflows onto storage-constrained distributed resources," in *CCGRID '07: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 401–409.

[15] Y. Zhao, J. Dobson, I. Foster, L. Moreau, and M. Wilde, "A notation and system for expressing and executing cleanly typed workflows on messy scientific data," *SIGMOD Records*, vol. 34, no. 3, pp. 37–43, 2005.

[16] J. Annis, Y. Zhao, J. Voeckler, M. Wilde, S. Kent, and I. Foster, "Applying Chimera virtual data concepts to cluster finding in the Sloan Sky Survey," in *Supercomputing '02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*. Los Alamitos, CA, USA: IEEE Computer Society Press, 2002, pp. 1–14.