COMPARATIVE EVALUATION OF THE ROBUSTNESS OF DAG SCHEDULING HEURISTICS

Louis-Claude Canon and Emmanuel Jeannot
LORIA, INRIA, Nancy University, CNRS
Campus Scientifique – BP 239
54506 Vandoeuvre-lès-Nancy Cedex, France
louis-claude.canon@loria.fr
emmanuel.jeannot@loria.fr

Rizos Sakellariou and Wei Zheng
School of Computer Science,
The University of Manchester,
Oxford Road, Manchester M13 9PL, U.K.
rizos@cs.man.ac.uk
zhengw@cs.man.ac.uk

Abstract In this paper, we analyze the robustness of 20 static, makespan-centric, DAG scheduling heuristics of the literature. We also study if dynamically changing the order of the tasks on their assigned processor improves the robustness. Based on experimental results we investigate how robustness and makespan are correlated. Finally, the heuristics are experimentally evaluated and ranked according to their performance in terms of both robustness and makespan.

Keywords: DAG scheduling heuristics, robustness, makespan, stochastic.

1. Introduction

With the emergence of distributed heterogeneous systems, such as grids, and the demand to run complex applications such as workflows, the problem of choosing *robust* schedules becomes more and more important. Indeed, in such environments, a carefully crafted schedule based on deterministic, statically-known, estimates for the execution time of the different tasks that compose a given application, may prove to be grossly inefficient, as a result of various unpredictable situations that may occur at run-time. Still, the existence of a good schedule is an important factor affecting the overall performance of an application. Thus, to mitigate the impact of uncertainties, it is necessary to choose a schedule that guarantees *robustness*, that is, a schedule that is affected as little as possible by various run-time changes.

There are several ways to achieve robustness. A first approach is to overestimate the execution time of individual tasks. This results in a waste of resources as it induces a lot of idle time during the execution, if the task duration is much shorter than the estimation. Another solution is to reschedule tasks dynamically allocating them to an idle processor in order to take into account information that has been made available during the execution. However, rescheduling a task is costly as it implies some extra communication and synchronization costs. Relevant studies [19] indicate that, in addition to rescheduling, it is important to have a static schedule with good properties before the start of the execution. Therefore, even if a dynamic strategy is used, a good initial placement would reduce the possibility of making a (later to be proved) bad decision and, hence, would reduce the extra costs of resorting to a dynamic strategy.

A significant amount of work in the literature has focused on proposing static directed acyclic graph (DAG) scheduling heuristics that minimize the overall application execution time (known as the *makespan*). However, to the best of our knowledge, so far, no study has tried to evaluate these heuristics with respect to the robustness of the schedule they produce. In this paper, we assess the robustness of twenty DAG scheduling heuristics from the literature designed to minimize the makespan.

In the remainder of this paper, Section 2 reviews related work on robustness and provides the definition used in this paper. Section 3 presents the model used to assess heuristics in terms of robustness. Section 4 describes the methodology of the experiments, Section 5 presents the experimental results and Section 6 concludes the paper.

2. Related work

The literature is abundant of makespan-centric, static DAG scheduling heuristics. For our evaluation, we chose 20 of these heuristics, which include some of the most widely used and cited. Due to lack of space, we refer the reader to

the relevant publications for the description of the heuristics. The 20 heuristics, in alphabetical order, are: BIL [16], CPOP [4], DPS [1], Duplex [8], FCP [17], FLB [17], GDL [22], HBMCT [18], HCPT [12], HEFT [23], k-DLA [24], LMT [13], MaxMin [8], MCT [8], MET [8], MinMin [6], MSBC [10], OLB [8], PCT [15], WBA [6].

Some work in the literature has attempted to define and model robustness; no widely accepted metric exists. In [2], the authors propose a general method to define a metric for robustness. First, a performance metric is chosen (this is the metric that needs to be robust). In our case, this performance metric is the makespan as we want the execution time of an application to be as stable as possible. Second, one has to identify the parameters that make the performance metric uncertain. In our case, it is the duration of the individual tasks and their communications. Third, one needs to find how a modification of these parameters changes the value of the performance metric. In our case, the answer is fairly simple, as an increase of the task or communication duration generally implies an increase of the execution time (even though, in some cases, a task may have a longer duration than expected and due to the structure of the schedule, such modification may not impact the overall makespan). Lastly, one has to identify the smallest variation of a parameter that makes the performance metric exceed an acceptable bound. A schedule A is said to be more robust than a schedule B if this variation is larger for A than for B. However, estimating this variation is the most difficult part as it requires to analyze deeply the structure of the problem and its inputs.

In order to simplify this framework, research in the context of evaluating the robustness of the makespan has proposed several other metrics, such as: the slack [7, 21, 19]; the probability that an execution exceed some expected bounds [20] (called the probabilistic metric); measures based on the Kolmogorov-Smirnov (KS) distance between the cumulative distribution (CDF) of the performance metric under normal operating conditions and the CDF of the same performance metric when perturbation applies [11]; or the differential entropy of the makespan [7]. In [9], we have studied the differences between these metrics and have concluded that the makespan standard deviation, the probabilistic metric and the differential entropy are highly correlated. This correlation was possibly due to the quasi-normality of the makespan distribution. Intuitively, the standard deviation of the makespan distribution indicates how narrow this distribution is. The narrower the distribution, the smaller the standard deviation is. This metric is related to the robustness because when two schedules are given the one for which the standard deviation is smallest is the one for which actual executions are more likely to have a makespan close to the average value. Mathematically, over several different values of the makespan, the standard deviation is given by $\sigma_M = \sqrt{avg(M^2) - avg(M)^2}$,

where $avg(M)$ is the average value of all makespan values available. The standard deviation will be used as a metric to assess robustness in this paper.

3. A Stochastic Model to Assess Robustness

We are given an application that is modeled by a stochastic task graph. This graph is a DAG, where vertices represent computational tasks and edges represent task dependencies (often due to communication). To model the uncertainty, task and communication cost are given by a random variable that follows a specific law (which can be different for all the tasks and communications). Hence, for each execution of the graph these costs may be different.

The task graph is executed on a set of heterogeneous resources. We assume that the topology of this infrastructure is complete (every machine can communicate to every one). We use the *related model* [14] concerning CPU capabilities: each CPU i is given a value τ_i, the time to execute one instruction. This means that if the cost of a task drawn from its random variable is c the execution time of this task on processor i is c_i. Concerning communication, we model each link by its latency (α) and its bandwidth (β). The time to send m bytes on link i is then $\alpha + \beta \times m$.

As we use static makespan-centric scheduling heuristics to map tasks onto the processors, we need to adapt the model to compute the schedule. We also need to compute the distribution of the makespan to determine its mean (average makespan) and its standard deviation (robustness).

To solve the above issues, we have proceeded as follows. Given a stochastic task graph, we transform it to a deterministic task graph by using only the mean value of the communication and task duration. With this deterministic task graph, we compute a schedule using one of our 20 heuristics. To compute the distribution of the makespan, we simulated, a large number of times, the execution of the schedule on the (heterogeneous) resources. This is a Monte-Carlo (MC) method, which means that each time a value for the duration of a task or communication is needed, this value is generated using the random variable that described it in the stochastic task graph. This allows us to compute the empirical distribution function (EDF), which converges to the true law of the makespan as the number of simulations increases, as stated by the Glivenko-Cantelli theorem. The precision achievable with a given number of MC simulations is given by the confidence intervals of the calculated approximations of the makespan mean and standard deviation. Since we consider the makespan distribution to be approximately normal, we use the Student's *t* and the chi-square distributions to compute these intervals and choose the number of simulations needed (see below).

Another issue that needs to be taken into account is the following. When doing a MC simulation of a deterministic schedule using a stochastic task graph,

it is not always possible, at runtime, to respect the start and end times of each task (that is, the times that were computed using static estimates). To address this problem, we propose (and use) two solutions. The first solution is that on each processor, we fully respect the order of the tasks, as it was produced by the schedule. A task is scheduled for execution only when all the tasks that, according to this schedule, must be executed before a given task have finished. We call this strategy *sequence*, because on a given processor, all tasks are executed in the same order than in the static schedule. The second solution is to respect processor assignments of tasks onto processors, but schedule ready tasks (that is, tasks whose parents have finished execution and all necessary data has been transmitted to these tasks) as soon as they become ready. This means that, sometimes, the order of the tasks, as given by the schedule for a single processor, may not be respected. We call this strategy *assignment*, because only the processor assignments in the schedule are respected, not the order as well.

4. Methodology

There are two phases in our experiments: a *deterministic* phase and a *stochastic* phase. In the first phase (deterministic), a specific DAG with static performance estimates is the input for each of the 20 static scheduling heuristics to generate a schedule. These schedules are further evaluated in the stochastic phase.

Two types of DAG are considered in our experiments. One type is derived from the Montage astronomy application [5]. The other is a random DAG, instances of which are randomly generated based on the following approach: (1) specify the number of nodes; (2) specify the number of levels; (3) randomly allocate the number of nodes at each level; (4) for each node except the exit, randomly appoint children nodes (at least one) in its lower neighbor level; (5) for each isolated node (non-entry node without parent), randomly appoint parent nodes in its upper neighbour level. In our experiments, we consider both Random and Montage DAGs with the following numbers of nodes: 58, 100, 500, 740, 1000, and 1186. In random DAGs, the number of levels is equal to the square root of the number of nodes. By combining each type of DAG with each different number of nodes, we generate 12 different DAGs.

We adopted the approach used in [3] to model task duration heterogeneity. A uniform random number R_{res} ranging from 1 to 10 is generated to describe resource heterogeneity, and another random number R_{task} following the same distribution is generated to describe task heterogeneity. Thus, the duration to run task i on resource j is determined by $T_{i,j} = R_{res} \times R_{task}$. In addition, the communication cost is modeled to satisfy that the ratio between mean task duration and mean communication duration is 1.0.

All DAGs can make use of 10 heterogeneous resources. Using this information, for each DAG generated as described above and for each of the 20 heuristics mentioned in Section 2, a static schedule is obtained, which will be assessed in the stochastic phase.

In the second phase of our experiments, once the deterministic graphs (and their schedules) have been produced, task durations are replaced by a random variable (RV) having as a mean the values described above. The distribution of these RV follows a Beta distribution with parameters $\alpha = 2$ and $\beta = 5$ (see [9] for a justification). In order to fully specify this, we also need to define the ratio between the maximum and the minimum bounds. We call this parameter the uncertainty level (UL) and set it to 1.1 on average with a very low dispersion (the UL is thus almost constant).

Finally, we need to settle the number of MC simulations in order to have a relevant precision for the calculated approximations of the makespan mean and standard deviation. To this end, we suppose that the makespan distribution is normal (as hinted in [9]). We can then easily measure the confidence intervals of these approximations. We see that for low variations of the makespan (as in our case), the variation of the standard deviation is preponderant and only depends on the number of MC simulations. To have less than 5% of precision with a confidence level of 99% we need 20,000 MC simulations. This amount increases quickly for better precision (750,000 for 1% of precision, for example).

5. Experiments

5.1 Normality

Our study is based on the hypothesis that the makespan of a stochastic graph is normal (it follows a Gaussian distribution). We validated this experimental hypothesis here by doing the Anderson-Darling (AD) test, which is one of the best EDF omnibus tests for normality. Intuitively, the statistic obtained corresponds to the distance of the EDF with a normal distribution. We observe that 96% of the schedules in the sequence case and 54% in the assignment case have an AD statistic smaller than 30 (the same as a Student distribution with 8 degrees of freedom). As these AD tests corroborate the normality assumption, we can reduce the simulation values to only 2 measures (average makespan and standard deviation) almost without loss of information in most cases.

5.2 Comparison of the sequence and assignment strategies

For each type of DAG, we have represented the performance of all the heuristics in Figures 1 and 2. Each heuristic has a different symbol. The x-axis represents the average makespan of the schedule produced by the heuristic. The

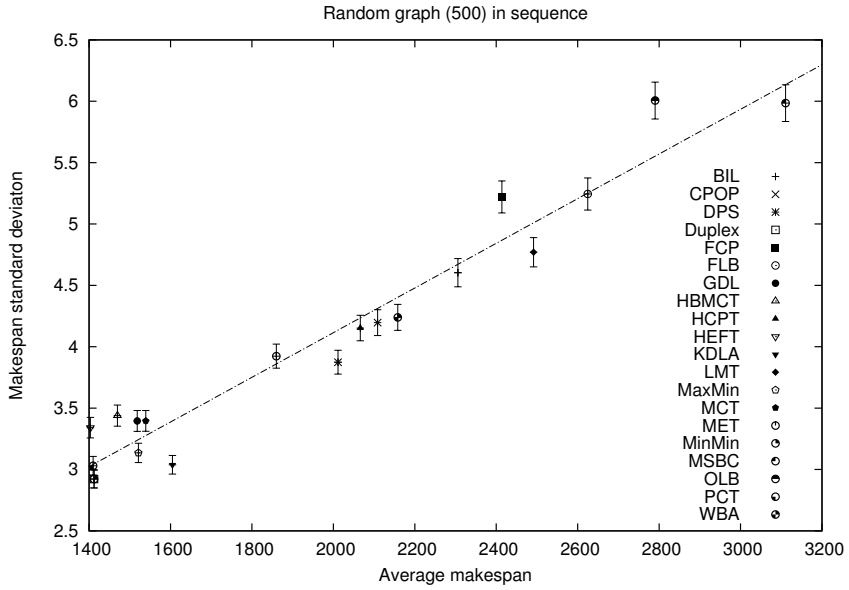

Figure 1. Mean vs. standard deviation of the makespan of different heuristics with the sequence strategy.

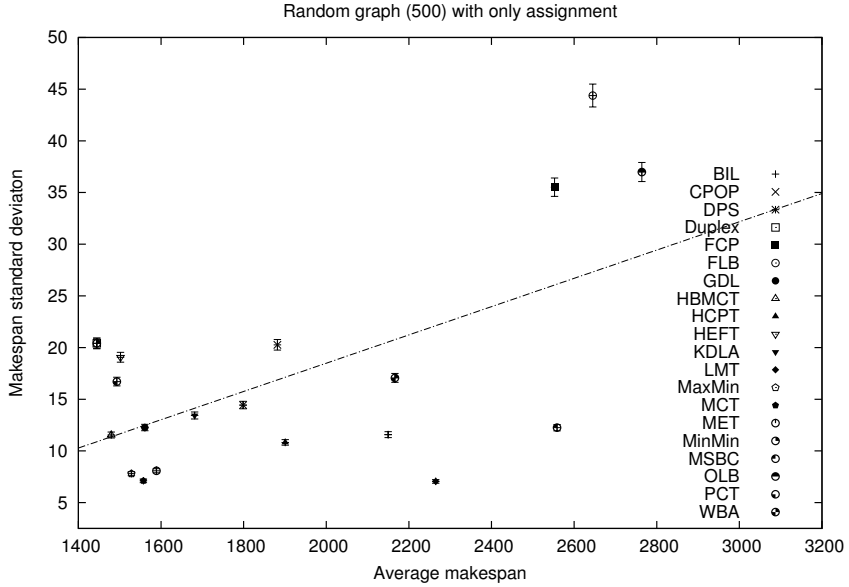

Figure 2. Mean vs. standard deviation of the makespan of different heuristics with the assignment strategy.

strategy	m58	m100	m500	m740	m1000	m1186	r58	r100	r500	r740	r1000	r1186
assignment	0.55	0.20	-0.19	0.76	0.66	0.60	0.73	0.05	0.59	0.44	0.60	0.36
sequence	0.81	0.85	0.62	0.71	0.73	0.62	0.47	0.87	0.97	0.91	0.97	0.89

Table 1. Correlation between makespan and robustness for the assignment and sequence strategies for different kind of graphs.

y-axis shows the standard deviation, the metric we use for robustness; the error bars correspond to the confidence intervals of each point with a confidence level of 99% (the probability for every point to be inside this range is 0.99). In addition, we plot the best fitting linear function for the points based on the least squares method. It helps to see the degree of correlation between the average value of the makespan and the robustness (more profound in the sequence case). The two figures shown allow the reader to compare the sequence and the assignment strategies for a certain type of DAG. While the average makespan does not change significantly in each case, the robustness is considerably worse with the second strategy.

As observed in the above example, the makespan mean and standard deviation are highly correlated. We compute the linear correlation coefficients (or Pearson coefficient) for each case and exhibit them in Table 1. This coefficient denotes the linear relationship existing between two RV (the mean and standard deviation estimators here). It takes values between -1, in the case of a decreasing linear relationship, and 1, in the case of an increasing linear relationship. Values close to 0 indicate the absence of a linear relationship. When restricting to the sequence strategy, the results show a strong correlation between mean and standard deviation in most cases (more than 0.7 in 75% of cases for the sequence case). This confirms the results in [9] and extends them in that the currently studied schedules are near-optimal.

We now investigate the effect of the choice between assignment and sequence on the schedule performance. In the example above, the most notable impact was an increase of the standard deviation in the assignment case. We compute the ratio between the assignment case and the sequence case for the mean and the standard deviation respectively, and show that this increase is a general trend. Tables 2 and 3 summarize these ratios by regrouping them with respect to the task graph or with respect to the heuristics. This table can be read as follows. For the Montage graph with 58 nodes (m58), the minimum ratio is 0.92 and the maximum ratio is 1.08, for the average makespan. For the standard deviation, 75% of the cases (from the 20 heuristics) have a ratio lower than 1.05. The first five columns indicate that, in most cases, the makespan remains extremely stable (with only a few extra-cases having more than 10% of difference). However, when there is a difference, the assignment strategy

Graph	Mean					Standard deviation				
	Min	25%	Med	75%	Max	Min	25%	Med	75%	Max
m58	0.92	1.00	1.00	1.00	1.08	0.79	1.00	1.00	1.25	13.8
m100	0.91	1.00	1.01	1.05	1.10	0.95	1.81	4.33	5.85	10.5
m500	0.88	1.00	1.00	1.00	1.00	0.94	1.00	1.00	1.25	4.90
m740	0.86	1.00	1.00	1.00	1.00	0.89	1.00	1.00	1.00	1.13
m1000	0.85	0.98	1.00	1.00	1.00	0.96	1.00	1.00	1.01	1.22
m1186	0.87	0.99	1.00	1.00	1.00	0.94	1.00	1.00	1.00	1.18
r58	0.92	1.00	1.00	1.01	1.06	0.99	1.00	1.00	2.55	11.3
r100	0.87	0.99	1.00	1.02	1.05	0.81	1.00	1.32	2.58	5.72
r500	0.82	0.91	1.01	1.03	1.07	1.48	2.50	3.88	5.93	8.46
r740	0.84	0.99	1.04	1.10	1.17	3.14	4.58	6.53	8.32	11.1
r1000	0.81	0.90	1.00	1.05	1.08	2.34	3.20	3.72	4.57	6.99
r1186	0.78	0.97	1.01	1.10	1.16	3.72	6.09	7.45	9.33	14.0

Table 2. Tukey's five number summary (quartiles) of ratio between the assignment case and sequence case for makespan and robustness; task graph view.

Heuristics	Mean					Standard deviation				
	Min	25%	Med	75%	Max	Min	25%	Med	75%	Max
BIL	0.93	0.99	1.00	1.04	1.13	0.98	1.00	2.10	4.16	11.0
CPOP	0.87	0.90	0.91	0.99	1.10	0.79	1.01	1.11	4.52	10.5
DPS	0.78	0.85	0.88	0.92	1.03	0.89	1.01	3.24	5.40	9.59
Duplex	1.00	1.00	1.02	1.03	1.06	1.00	1.00	2.40	4.07	6.98
FCP	1.00	1.00	1.01	1.06	1.06	1.00	1.00	1.50	5.62	6.80
FLB	0.94	0.99	1.00	1.00	1.03	0.96	1.00	1.93	3.68	8.46
GDL	1.00	1.00	1.01	1.04	1.16	0.99	1.00	2.24	5.91	11.1
HBMCT	1.00	1.00	1.02	1.06	1.10	0.98	1.00	3.41	7.31	9.72
HCPT	0.86	0.88	0.92	0.98	1.08	0.94	1.08	3.30	5.45	13.8
HEFT	1.00	1.00	1.00	1.07	1.16	1.00	1.00	1.57	6.48	14.0
KDLA	1.00	1.00	1.02	1.05	1.17	0.99	1.00	2.79	5.55	8.94
LMT	0.91	0.96	0.98	1.00	1.03	0.83	0.99	1.00	2.16	8.04
MaxMin	0.98	1.00	1.00	1.01	1.02	0.81	1.00	1.68	2.56	6.20
MCT	1.00	1.00	1.01	1.02	1.05	1.00	1.00	2.72	4.67	8.50
MET	0.85	0.87	0.94	1.00	1.00	0.92	1.00	1.56	6.99	11.3
MinMin	1.00	1.00	1.01	1.03	1.06	1.00	1.00	1.65	4.07	6.98
MSBC	0.82	0.87	0.98	1.00	1.00	1.00	1.04	1.20	2.53	4.46
OLB	0.98	1.00	1.00	1.00	1.02	1.00	1.00	1.00	5.11	8.52
PCT	1.00	1.00	1.00	1.07	1.16	0.84	1.00	1.19	6.25	11.1
WBA	0.99	1.00	1.00	1.00	1.05	1.00	1.00	1.03	3.70	8.01

Table 3. Tukey's five number summary (quartiles) ratio between the assignment case and sequence case for makespan and robustness; heuristics view.

Rank	Montage		Random	
	mean	std dev	mean	std dev
1	GDL [1.7]	GDL [2.0]	HEFT [2.7]	HEFT [3.7]
2	HBMCT [3.7]	HEFT [2.8]	PCT [3.3]	PCT [4.2]
3	BIL [4.2]	KDLA [3.3]	Duplex [3.7]	HBMCT [4.8]
4	HEFT [4.5]	PCT [3.5]	GDL [4.8]	Duplex [5.7]
5	PCT [4.5]	BIL [5.7]	MinMin [5.5]	GDL [6.3]
6	KDLA [6.3]	HBMCT [6.8]	MCT [7.2]	KDLA [6.3]
7	Duplex [7.0]	FCP [9.0]	KDLA [7.3]	MaxMin [6.7]
8	MCT [8.5]	Duplex [10.7]	MaxMin [7.5]	MinMin [7.0]
9	MinMin [9.2]	MSBC [10.8]	HBMCT [7.8]	MCT [9.5]
10	MaxMin [9.8]	MaxMin [11.0]	BIL [12.0]	WBA [11.0]
11	FCP [11.0]	CPOP [11.3]	FCP [12.5]	BIL [12.3]
12	WBA [11.7]	MCT [12.3]	WBA [13.0]	DPS [12.3]
13	MSBC [13.7]	WBA [13.0]	LMT [14.2]	HCPT [12.7]
14	OLB [13.8]	MinMin [13.5]	CPOP [14.3]	LMT [13.7]
15	CPOP [14.2]	LMT [13.8]	FLB [14.3]	CPOP [14.0]
16	FLB [15.0]	OLB [14.5]	HCPT [14.3]	FCP [14.8]
17	LMT [16.0]	FLB [14.7]	DPS [14.8]	FLB [15.2]
18	MET [18.3]	DPS [15.7]	MET [15.5]	MET [15.2]
19	DPS [18.5]	HCPT [16.5]	OLB [16.8]	OLB [17.0]
20	HCPT [18.5]	MET [19.0]	MSBC [18.3]	MSBC [17.7]

Table 4. Makespan and robustness ranking of the heuristics for the montage and random task graph cases.

allows more gain than the sequence strategy. Regarding the robustness metric, in most cases the assignment strategy is at least two times worse than the sequence strategy and in extreme cases, it can be up to one order of magnitude worse. This signifies that the assignment strategy is inferior in term of robustness but almost equal in terms of average makespan performance. In Table 3, this comparison can also be thought as a kind of sensitivity analysis of the stability of the schedule generated by a given heuristic. Even though the quantity of schedules is too low to draw any conclusion with respect to this point, it appears that heuristics such as LMT, MaxMin, MSBC are among the most stable. Similarly, the montage graph seems to be in general less sensitive than random graphs.

5.3 Heuristic comparison

In this last part, we rank every heuristic with the sequence strategy as this strategy has been shown superior in the previous section. Table 4 features the best heuristics in term of both the mean and the standard deviation of the makespan, and for the two types of task graph (random and montage). While

the precision for the makespan mean is always below 0.1%, the precision for the standard deviation is only 5%. We observe that the best heuristic for the montage graphs is GDL and for the random graphs, HEFT (in term of both average makespan and robustness).

6. Conclusion

In this paper we have studied the robustness of 20 static makespan-centric DAG scheduling heuristics from the literature, using as a metric for robustness the standard deviation of the makespan over a large number of measurements.

Our results are three-fold. First, we have shown that it is better to respect the static order of the tasks on the processors than to change this order dynamically. Second, we have shown that robustness and makespan are somehow correlated: as it has been suggested elsewhere [19], schedules that perform well statically tend to be the most robust. Third, we have shown that, for the cases we have studied, heuristics such as HEFT, HBMCT, GDL, PCT, are among the best for both makespan and robustness.

Future work can be directed to the study of robustness-centric heuristics like slack-based or convex clustering strategies. Another direction is to develop multi-criteria strategies (that both optimize robustness and makespan). Lastly, it would be interesting to see how to deal with stochastic information inside a deterministic heuristic, instead of only using the mean, as in this present work.

References

[1] I. Ahmad, M.K. Dhodhi, and R. Ul-Mustafa. DPS: Dynamic Priority Scheduling Heuristic for Heterogeneous Computing Systems. *IEE Proceedings – Computers & Digital Techniques*, 145(6), pp. 411-418, 1998.

[2] S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim. Measuring the Robustness of a resource Allocation. *IEEE Transactions on Parallel and Distributed Systems*, 15(7), pp. 630-641, July 2004.

[3] S. Ali, H.J. Siegel, M. Maheswaran, D. Hensgen and S. Ali. Task Execution Time Modeling for Heterogeneous Computing Systems. *Proceedings of the 9th Heterogeneous Computing Workshop*, pp. 185-199, 2000.

[4] O. Beaumont, V. Boudet, and Y. Robert. The Iso-Level Scheduling Heuristic for Heterogeneous Processors. *Proceedings of the 10th Euromicro Workshop on Parallel, Distributed and Network-Based Processing (PDP2002)*, 2002.

[5] G.B. Berriman, J.C. Good, A.C. Laity, A. Bergou, J. Jacob, D.S. Katz, E. Deelman, C. Kesselman, G. Singh, M. Su and R. Williams. Montage: a Grid Enabled Image Mosaci Service for the National Virtual Observatory. *Astronomical Data Analysis Software and Systems XIII (ADASS XIII)*, Vol. 314, 2004.

[6] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Task Scheduling Strategies for Workflow-Based Applications in Grids. *CCGrid 2005*, 2005.

[7] L. Bölöni and D. C. Marinescu. Robust scheduling of metaprograms. *Journal of Scheduling*, 5(5), pp. 395-412, September 2002.

[8] T.D. Braun, H.J. Siegel, N. Beck, et al. A Comparison of Eleven Static Heuristic for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, 61, pp. 810-837, 2001.

[9] L.-C. Canon and E. Jeannot. A Comparison of Robustness Metrics for Scheduling DAGs on Heterogeneous Systems. In *HeteroPar'07*, Sept. 2007.

[10] H. Chen. On the Design of Task Scheduling in the Heterogeneous Computing Environments. *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, 2005.

[11] D. England, J. Weissman, and J. Sadagopan. A New Metric for Robustness with Application to Job Scheduling. *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing*, pp. 135-143, July 2005.

[12] T. Hagras and J. Janecek. A Simple Scheduling Heuristic for Heterogeneous Computing Environments. *Proceedings of the 2nd International Symposium on Parallel and Distributed Computing*, pp. 104-110, 2003.

[13] M. Iverson, F. Ozguner, and G. Follen. Parallelizing Existing Applications in a Distributed Heterogeneous Environment. *Proceedings of the 4th Heterogeneous Computing Workshop (HCW'95)*, 1995.

[14] J.W.S. Liu and C.L. Liu. Bounds on scheduling algorithms for heterogeneous computing systems. *Proceedings of IFIP Congress 74*, pp. 349-353, 1974.

[15] S. Manoharan and N. P. Topham. An Assessment of Assignment Schemes for Dependency Graphs. *Parallel Computing*, 21(1), pp. 85-107, 1995.

[16] H. Oh and S. Ha. A Static Scheduling Heuristic for Heterogeneous Processors. *Proceedings of the 2nd International Euro-Par Conference*, vol. 2, pp. 573-577, 1996.

[17] A. Radulescu and A. Van Gemund. Fast and Effective Task Scheduling in Heterogeneous Systems. *Proceedings of the 9th Heterogeneous Computing Workshop (HCW)*, pp. 229-238, 2000.

[18] R. Sakellariou and H. Zhao. A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems. *Proceedings of the 13th Heterogeneous Computing Workshop (HCW)*, IEEE Computer Society Press, 2004.

[19] R. Sakellariou and H. Zhao. A low-cost rescheduling policy for efficient mapping of workflows on grid systems. *Scientific Programming*, 12(4), December 2004, pp. 253-262.

[20] V. Shestak, J. Smith, H. J. Siegel, and A. A. Maciejewski. A Stochastic Approach to Measuring the Robustness of Resource Allocations in Distributed Systems. *2006 International Conference on Parallel Processing*, August 2006.

[21] Z. Shi, E. Jeannot, and J. J. Dongarra. Robust Task Scheduling in Non-Deterministic Heterogeneous Computing Systems. *Proceedings of IEEE International Conference on Cluster Computing*, September 2006.

[22] 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), pp. 175-187, 1993.

[23] 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*, 13(3), pp. 260-274, 2002.

[24] N. Woo and H.Y. Yeom. K-Depth Look-Ahead Task Scheduling in Network of Heterogeneous Processors. *Lecture Notes in Computer Science*, Vol. 2344, pp. 736-745, 2002.