An Efficient Weighted Bi-objective Scheduling Algorithm for Heterogeneous Systems[⋆]

Idalmis Milián Sardiña, Cristina Boeres, and Lúcia Maria de A. Drummond

Instituto de Computação, Universidade Federal Fluminense, Brazil
{isardina,boeres,lucia}@ic.uff.br

Abstract. This paper proposes the *Makespan and Reliability Cost Driven* (MRCD) heuristic, a static scheduling strategy for heterogeneous distributed systems that not only minimizes the makespan but also maximizes the reliability of the application. The scheduling decisions made by MRCD are guided by a weighted function that considers both objectives simultaneously instead of prioritizing only one of the objectives. This work also introduces a classification of the solutions produced by weighted bi-objective schedulers to aid users to tune the weighting function in order to generate an appropriate solution in accordance with their needs. In comparison with related work, MRCD produced schedules with makespans that were significantly better then those produced by other strategies at expense of an insignificant deterioration in reliability.

1 Introduction

Resources in large parallel and distributed systems may not be available for a long period of time. Therefore, when executing an application on such systems it is important to tackle reliability and fault-tolerance issues. Aiming an efficient execution of parallel applications on distributed heterogeneous system, this work specifies a schedule strategy that considers not only the minimization of the running time or *makespan*, but also maximizes the reliability of the application execution. The target application is represented by a directed acyclic graph (DAG), whose vertices are the tasks and edges are dependencies between them. A weighted bi-objective cost function that integrates both objectives guides the decisions to schedule the tasks of the parallel application on heterogeneous systems susceptible to failures.

In the literature, there is a variety of works that deal with this problem, which specifies in the cost function the two objectives in distinct ways. The work in [1] is a real time system oriented scheduling algorithm which orders the tasks of a parallel application by their deadlines. The cost function is a hierarchical one, since for each task, the subset of processors that maximizes the reliability is identified, and then the processor from this subset that minimizes the earliest start time is selected. As a consequence, the results favour the reliability rather than the execution time of the application. The authors in [2] derived the importance of

[⋆] This work is supported by research grants from CNPq and FAPERJ.

the product *failure rate* $\times$ *task execution time* for the case of scheduling independent tasks onto processors and proposed an extension of list scheduling heuristics, like HEFT [3], by taking into account reliability. In [4] a weighted bi-objective list scheduling algorithm (BSA) is proposed. After sorting the tasks in non-increasing order of their priority, the chosen processor is the one that minimizes a weighted integrated cost function. The objectives makespan and reliability are normalized by their maximum values and combined in the function. Nonetheless, [4] presents the use of only three solutions: the one that minimizes only the makespan; the one that weights equally both objectives; and the one that maximizes only the reliability. It must be pointed out that there are also works which manipulate multiple objective functions [5]. However, these approaches usually belong to a class of heuristics which are out of the scope of this paper.

The *Makespan and Reliability Cost Driven* (MRCD) proposed here is a list scheduling heuristic that takes into account the reliability and makespan based on [1]. However, instead of considering both objectives in a hierarchical fashion, the proposed cost function integrates both objectives simultaneously. The way in which the strategy was designed allows that the maximization of the reliability does not compromise the minimization of the makespan. The experimental analysis shows that MRCD produced schedules with very efficient performance when compared with other works from the literature. Besides, a classification of the solutions generated by weighted bi-objective schedules is presented as a way to aid the choice of a solution that better achieves the users' needs.

2 Scheduling Model for Distributed Systems with Faults

In this work, the *application model* is based on the class of parallel applications that can be represented by directed acyclic graphs (DAGs). A task graph is denoted by $G = (V, E, \varepsilon, \omega)$, where: the set of vertices V represents *tasks*; E, the precedence relation among them; $\varepsilon(v)$ is the amount of work associated with task $v \in V$; and $\omega(u, v)$ is the weight associated with the edge $(u, v) \in E$, representing the amount of data units transmitted from task u to v.

The *architectural model* specifies the main features of the target architecture. Given the set $P = \{p_0, p_1, \ldots, p_{m-1}\}$ of available heterogeneous processors, the computational slowdown index $csi(p_j)$ is associated to each $p_j \in P$, which is inversely proportional to the computational power of p_j, as identified by [6]. The communication delay index $cdi_{i,j}$ estimates the latency cost associated with each communication link (p_i, p_j). Therefore, the execution time of a task v on a processor p_j is $et(v, p_j) = \varepsilon(v) \times csi(p_j)$. The communication time to send the amount $\omega(u, v)$ of data between the adjacent tasks u and v allocated to distinct processors p_u and p_v is given by $ct(u, v) = \omega(u, v) \times cdi(p_u, p_v)$. Note that $cdi(p_u, p_v) = 0$ if $p_u = p_v$. Upon the definition of these costs, a schedule algorithm may need to calculate the earliest time that a task v can start its execution on processor p_j, which is $EST(v, p_j) = \max_{u \in pred(v)}\{EST(u, p_u) + et(u, p_u) + ct(u, v)\}$ and depends on the schedule of v's predecessor tasks and the availability of the processor. The earliest finish time of task v on processor p_j is then $EFT(v, p_j) =$

$EST(v, p_j) + et(v, p_j)$. A schedule Sch of a DAG G specifies, for each $v \in V$, the processor and the time in which v must be completed. The schedule length or *makespan* of Sch is defined as $\mathcal{M}(Sch) = \max_{\forall v \in V}\{EFT(v, p_v)\}$.

The reliability of a system is based on the probability in which resources of the system execute tasks without any failure [1]. In this work, only processor failures are considered, which are statistically independent and follow a Poisson Law with a failure rate of $FR(p_j)$ $\forall p_j \in P$ [7]. This rate represents the number of processor failures per unit of time that can occur. It is assumed here that the communication links between processors are reliable, based on the assumption that communication protocols are able to tolerate their failures [7].The task reliability cost associated with the execution of v in p_j is then $RC(v, p_j) = FR(p_j) \times et(v, p_j)$, which should be minimized. Given the set of tasks allocated to p_j, $V_{p_j} \subset V$, the processor reliability cost is then $RC_p(p_j) = \sum_{\forall v \in V_{p_j}} RC(v, p_j)$. As a consequence, the system reliability cost associated with the execution of G on P in accordance with a schedule Sch is $RC_s(G, P, Sch) = \sum_{\forall p_j \in P} RC_p(p_j)$. Finally, as in [1], the total reliability of the scheduled G in P is $R_T = e^{-RC_s(G,P,Sch)}$, which is the probability that the application G can run successfully on the set P under the schedule Sch during its execution. As a matter of fact, R_T should be maximized and also, by minimizing the task reliability cost $RC(v, p_j)$, the total reliability R_T becomes close to one.

3 A Weighted Bi-objective Scheduling Strategy

The MRCD scheduling algorithm follows the traditional *list scheduling* framework: firstly, during the *ordering phase*, a priority is associated with the tasks. Then, during the *scheduling* phase, each task with the highest priority is allocated to the processor that minimizes a cost function. This work defines a weighted bi-objective function to guide the scheduling process, such that not only the makespan of the application is minimized but also the total reliability of the application scheduled in P is maximized. The specification of proper weights to the objectives can be however, troublesome. On an attempt to provide information to aid this specification, a metric is also generated by MRCD, which is the average difference between the two objectives.

The bottom level $blevel(v)$ is the priority assigned to the tasks in V as in [3]. During the *scheduling* phase, the algorithm seeks for the processor that optimizes the cost function. For doing so, MRCD also implements the *tasks insertion* procedure from [3]. Let v be the next task to be scheduled with the highest $blevel(v)$. The chosen processor p_v is the one that minimizes the cost function $f(v, p_v) = \min_{p_j \in P}\{(1 - w)EFT(v, p_j) + wRC(v, p_j)\}$. When $w = 0$, MRCD becomes similar to HEFT [3], in which the objective is only the minimization of the makespan. When $w = 1$, MRCD only considers the reliability.

A closer look to $f(v, p_j)$ can take one to note that the values of the objectives are not comparable and aggregating them is not straightforward. In Therefore, it is necessary to normalize both objectives $EFT(v, p_j)$ and $RC(v, p_j)$ for each task in V over the set of processors P. The normalization procedure transforms all the objectives so that they share the same minimum and maximum values 0 and 100. Then, the linear operator applied to the i^{th} objective is $O^i_n = norm(0, 100, O^{min}_i, O^{max}_i) = 100 \times \frac{O_i - O^{min}_i}{O^{max}_i - O^{min}_i}$, where O^{min}_i and O^{max}_i are the minimum and maximum values of the i^{th} objective, respectively. In this work, O^i_n can be either the normalized values $EFT_n(v, p_j)$ or $RC_n(v, p_j)$.

Algorithm 1. *MRCD(G,P,w)*

```
$V_{ord} = \langle v_0, v_1, \ldots, v_{n-1}\rangle / blevel(v_i) \le blevel(v_{i+1}), i = 0, \ldots, n-2$;
for $i = 0, \ldots, n-1$
    $F = \infty$;
    $\forall p_j \in P$
        Calculate $EST(v_i, p_j)$ using $taskInsertion(v_i, p_j)$;
        $EFT(v_i, p_j) = EST(v_i, p_j) + et(v_i, p_j)$;
        $RC(v_i, p_j) = FR(p_j) \times et(v_i, p_j)$;
    $EFT_{min} = \min_{p_j \in P}\{EFT(v_i, p_j)\}$; $EFT_{max} = \max_{p_j \in P}\{EFT(v_i, p_j)\}$;
    $RC_{min} = \min_{p_j \in P}\{RC(v_i, p_j)\}$;
    $RC_{max} = \max_{p_j \in P}\{RC(v_i, p_j)\}$;
    $\forall p_j \in P$
        $EFT_n(v_i, p_j) = norm(0, 100, EFT_{min}, EFT_{max}, EFT(v_i, p_j))$;
        $RC_n(v_i, p_j) = norm(0, 100, EFT_{min}, RC_{max}, RC(v_i, p_j))$;
        $f(v_i, p_j) = (1 - w) \times EFT_n(v_i, p_j) + w \times RC_n(v_i, p_j)$;
        if $(f(v_i, p_j) < F)$
            $F = f(v_i, p_j)$; $p_{v_i} = p_j$;
    if $(\mathcal{M}(Sch) < EFT(v_i, p_{v_i}))$ $\mathcal{M}(Sch) = EFT(v_i, p_{v_i})$;
    $RC_s = RC_s + RC(v_i, p_{v_i})$;
    $Sch = Sch \cup \langle v_i, p_{v_i}, EST(v_i, p_{v_i})\rangle$;
$R_T = e^{-RC_s}$; $D(Sch) = \frac{\sum_{v_i \in V} RC_n(v_i, p_{v_i}) - EFT_n(v_i, p_{v_i})}{n}$;
```

Algorithm 1 shows the steps of the MRCD Algorithm for a given $G = (V, E, \epsilon, \omega)$ and set of processors P. The *ordering* phase of MRCD generates the list V_{ord} of tasks in non-decreasing order of $blevel(v)$ (line 1). The *scheduling* phase is executed for each $v_i \in V_{ord}$ from lines 2 to 19, where in lines 4 to 7, the earliest finishing time $EFT(v_i, p_j)$ and the minimal task reliability $RC(v_i, p_j)$ are calculated, for each $p_j \in P$. In lines 8 to 10, both the minimal and maximal values for $EFT(v_i, p_j)$ and $RC(v_i, p_j)$ over all $p_j \in P$ are collected and then applied to the normalization procedure. The normalized values of $EFT_n(v_i, p_j)$ and $RC_n(v_i, p_j)$ are then applied to the cost function $f(v_i, p_j)$ where the processor which minimizes it is identified (lines 15 and 16). MRCD generates the final schedule Sch, its makespan $\mathcal{M}(Sch)$ and the total reliability cost R_T, as seen in lines 17, 19 and 20, respectively. Furthermore, with the normalized values $EFT_n(v_i, p_{v_i})$ and $RC_n(v_i, p_{v_i})$, for each v_i in its best processor p_{v_i}, their difference portrays the imbalance degree between both objectives. The average difference or imbalance degree between the objectives $D(Sch)$ of the resulting schedule Sch is given by $D(Sch) = \frac{\sum_{v_i \in V} RC_n(v_i, p_{v_i}) - EFT_n(v_i, p_{v_i})}{n}$, as seen in line 20. The sign of $D(Sch)$ shows which objective most contributed to the specification of the produced schedule. If negative, the reliability cost contributed more, otherwise, the makespan dominates. In this way, if its module $|D(Sch)|$ is close to zero, the imbalance degree is practically negligible and an equally contribution of the objectives are held.

3.1 A Classification of the Weights Applied to the Cost Function

When many and probably conflicting objectives are optimized simultaneously, there is no longer a single optimal solution but rather a whole set of possible solutions of equivalent quality. In the problem tackled in this work, the two objectives might be conflicting: while a processor can finish the execution of an application task quickly, a high failure rate can be assigned to it depending on the reliability model. Therefore, there is no single optimum solution to be found and actually a variety of solutions can be optimal in some sense. This work provides the users the option to assess the trade-offs between the objectives with additional information about this set of solutions.

Let S be the set of all feasible solutions for the bi-objective problem tackled here. Let $S_k = \langle Sch_k, \mathcal{M}(Sch_k), R_T(Sch_k), D(Sch_k) \rangle$ be one solution of the bi-objective problem and the associated additional information to be analysed. A solution $S_k \in S$ dominates another solution S_q if the following conditions are satisfied: (i) S_q is not better than S_k in both objectives, i.e. $\mathcal{M}(Sch_k) \leq \mathcal{M}(Sch_q)$ and $R_T(Sch_k) \geq R_T(Sch_q)$; (ii) S_k is absolutelly better than S_q in at least one of the objectives, i.e. $\mathcal{M}(Sch_k) < \mathcal{M}(Sch_q)$ or $R_T(Sch_k) > R_T(Sch_q)$. The solution S_k is dominant and S_q is dominated by S_k. If S_k does not dominate S_q and vice-versa, the solutions are incomparable [5]. A variety of feasible solutions can be produced by MRCD for the same input DAG G and target system if different w values are given to the algorithm. Let W be the set of values associated with w of the cost function in MRCD. In accordance with the dominance condition (i) and (ii) described above, the set of all solution in S that are dominants and incomparable is denoted as S' (non-dominated). Note that the solutions in S' are not dominated by any other solution in S. The concept of dominance can be sometimes too weak for applications, in cases where one objective can be improved significantly at a cost of a small deterioration of the other. This concept does not necessarily tell which solutions to chose, but rather which solutions to avoid [5].

A classification is then proposed, having the knowledge of the solutions in S'. The solution $S_e \in S'$ that offers an equilibrium between the objectives are those with the smallest value of $|D(Sch_e|$. The set of solutions S_{R_T} is compounded of those $S_k \in S'$ with the smallest $D(Sch_k)$, and consequently are the ones with the highest total reliability $R_T(Sch_k)$. On the other hand, the set of solutions denoted by $S_{\mathcal{M}}$ contains the $S_k \in S'$ with the highest $D(Sch_k)$, i.e. are those with the minimum makespan $\mathcal{M}(Sch_k)$.

4 Experimental Results

The proposed scheduling algorithm MRCD was compared with other scheduling strategies from the literature, which were divided into two classes: the first one produces a unique solution, denoted as Class 1, encompasses the heuristics HEFT [3], RCDMod (based on [1]) and RHEFT (based on [2]); and Class 2, containing BSAMod based on [4] that can also produce several solutions. BSAMod emplows a cost function that although seems similar to MRCD produces different results, added to the fact that a distinct normalization procedure is used. In order to have a fair comparison with MRCD, some of these heuristics were implemented with small changes, as will be describe next.

In Class 1, since HEFT minimizes only the makespan of the resulting schedule, to fairly compare with MRCD, this work derived R_T based on the final schedule produced by HEFT. The work [1] also gives a solution for the bi-objective scheduling problem, but uses a hierarchical cost function. Differently from the original proposal, RCDMod was implemented with no time constraints (deadlines) and the employed cost function used the tasks completion times instead of their start times. Finally, RHEFT implements the list scheduling approach of HEFT, but considers both objectives in its cost function as the multiplication of the execution time and failure rate as in [2]. In Class 2, BSAMod also tackles the bi-objective problem by using the function $f(v,p) = \sqrt{w\left(\frac{EFT(v,p_j)}{\max_{p\in P}\{EFT(v,p)\}}\right)^2 + (1-w)\left(\frac{RC(v,p_j)}{\max_{p\in P}\{RC(v,p)\}}\right)^2}$, as proposed in [4], but sorts tasks by $blevels(v)$ and executes the task insertion procedure.

All heuristics were executed over synthetic applications represented by three classes of DAGs: one that represents the Gaussian Elimination, denoted by G_n; the diamond DAG Di_n, which represents, for example, matrix multiplication; and random generated DAGs R_n, graphs with irregular topologies. In all the cases, n denotes the number of tasks of the DAG. In all graphs, a unit cost was associated with the communications between tasks. The same does not hold for the tasks weights. In G_n, each $v \in V$ has a distinct computation weight, while in both R_n and Di_n, the weight $\epsilon(v) = 50$.

The simulated distributed system was composed of m processors divided into three groups, denoted as P_0, P_1 and P_2, each one with $m/3$ processors. The processor failure rate, $FR(p_i)$, was uniformly generated in $[10^{-5}, 3.3 \times 10^{-5}]$, $\forall p_i \in P_0$; $[3.4 \times 10^{-5}, 6.6 \times 10^{-5}]$, for $p_i \in P_1$; $[6.7 \times 10^{-5}, 10^{-4}]$, for $p_i \in P_2$, as proposed in [1]. Concerning the computational slowdown index, the adopted values were $csi(p_i) = 73$, for $p_i \in P_0$; $csi(p_i) = 53$, for $p_i \in P_1$; $csi(p_i) = 33$, for $p_i \in P_2$, which were obtained from [6]. The failure rate is per second and the makespan value in seconds.

An illustration of the proposed scheduling algorithm and the classification of w values given by MRCD considering G_{702} and the architectural scenario with 24 processors is shown in Table 1 where: w is the weight applied to the cost function $f(v, p_j)$; $\mathcal{M}$ is the makespan of the produced schedule; R_T is the total reliability; and the associated average difference D. The columns in bold show S', the non-dominated solutions. Among them, three solutions are particularly interesting under the classification proposed in Section 3.1: $S_{\mathcal{M}}$, S_{R_T} and S_e for $w = 0.4$, $w = 0.9$ and $w = 0.7$, respectively. From the definition of D and the normalized values RC_n and EFT_n in Algorithm 1, for high values of w and consequently, low values of D, MRCD tends to maximize reliability, otherwise, the makespan is minimized.

Table 1. Solution provided by MRCD for G_{702}

w	0.1	0.2	0.3	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**
$\mathcal{M}$	902.8	902.8	902.8	**902.8**	**906.3**	**1266.4**	**3463.3**	**7072.2**	**12687.4**
RT	0.44084	0.44768	0.45400	**0.45539**	**0.46016**	**0.48362**	**0.53184**	**0.59064**	**0.62604**
D	49.1	45.7	42.8	**41.0**	**38.7**	**27.8**	**-4.8**	**-36.3**	**-57.3**

4.1 Comparing MRCD with Other Heuristics

The first set of experiments compares the results of MRCD with the heuristics in Class 1. Table 2 presents the pair makespan and the system reliability $(\mathcal{M}, R_T)$, and also average difference D for each DAG, considering the three following comparisons. The first comparison C1 gives the results of HEFT and MRCD, C2 shows the results of RCDMod and MRCD, and C3 compares RHEFT and MRCD. In each case, the MRCD solutions selected were: $S_{\mathcal{M}}$ from S' in C1 for a fair comparison with HEFT; $S_{R_T} \in S'$ in the case of C2, since RCDMod gives higher priority to the processors that maximizes reliability; and the dominant solution or in case of MRCD solutions do not dominate the RHEFT solution, the MRCD solution closest to the RHEFT one is shown. A solution S_i is the closest one to another solution S_j if it presents the smallest Euclidian distance from it considering normalized values.

It can be observed that with the increase of n, the makespan also grows and the reliability decreases. As the number of tasks executed on a fixed number of processors grows, the processors loads also increase and consequently their reliability costs. Table 2 also shows that the solutions in C1 have worse reliability than the results in C2 and C3, but the makespan are amazingly smaller. It happened because in the employed environment, the fastest processors were also the most susceptible to failures, and the chosen values for w gave priority to the minimization of the makespan. Most interestingly, MRCD presented better solutions than HEFT in all graphs. By choosing MRCD solutions with the greatest D values (the $S_{\mathcal{M}}$ solution), the algorithm showed to be capable of producing solutions with makespans as good as (or better than) those of HEFT, but with better reliability.

In C2, RCDMod presented slightly better reliability than the results S_{R_T} for each DAG in Table 2. Remark that such heuristic employs a hierarchical cost function that prioritizes the reliability only, and although the MRCD solutions also consider reliability, the makespan was not neglected since the cost function integrates both objectives. However, it is important to note that the makespan of the RCDMod are almost the double of those presented by MRCD. Finally in C3, MRCD solutions dominated RHEFT solutions in many graphs. RHEFT employs

Table 2. Results for Gauss, Random and Diamond DAGs on $m = 24$ processors in Class 1

	C1			C2			C3		
DAG_N	HEFT	MRCD	D	RCDMod	MRCD	D	RHEFT	MRCD	D
G_{152}	(190.0, 0.92071)	**(190.0, 0.93294)**	30.3	(2406.0, 0.95761)	(1283.3, 0.95546)	-58.7	(694.9, 0.94573)	(449.4, 0.94287)	-13.2
G_{252}	(318.7, 0.83511)	**(318.7, 0.85879)**	31.0	(5201.9, 0.91061)	(2715.6, 0.90609)	-58.1	(1309.6, 0.88130)	(856.3, 0.87844)	-7.6
G_{377}	(480.4, 0.71847)	**(480.4, 0.75140)**	32.7	(9603.8, 0.84124)	(4981.5, 0.83350)	-57.2	(2204.6, 0.78693)	(1494.8, 0.78560)	-6.3
G_{527}	(675.1, 0.58537)	**(675.1, 0.61228)**	35.6	(15976.7, 0.75007)	(8211.0, 0.73851)	-57.1	(3560.9, 0.66798)	(2273.0, 0.66578)	-4.4
G_{702}	(902.8, 0.44064)	**(902.8, 0.45539)**	41.0	(24685.6, 0.64124)	(12687.4, 0.62604)	-57.3	(5258.9, 0.52865)	**(3463.3, 0.53184)**	-4.8
R_{80}	(330.0, 0.82648)	**(330.0, 0.84159)**	27.7	(5840.0, 0.90021)	(3066.0, 0.89523)	-58.7	(1022.0, 0.85271)	**(949.0, 0.86478)**	-13.2
R_{98}	(330.0, 0.79053)	**(330.0, 0.80765)**	29.9	(7154.0, 0.87917)	(3650.0, 0.87303)	-58.7	(1460.0, 0.83060)	**(1314.0, 0.84035)**	-10.6
R_{152}	(396.0, 0.69599)	**(396.0, 0.69907)**	42.0	(11096.0, 0.81895)	(5694.0, 0.81015)	-56.9	(1679.0, 0.73349)	**(1533.0, 0.75338)**	-5.95
R_{256}	(528.01, 0.54215)	**(528.00, 0.54215)**	38.2	(18688.0, 0.71434)	(10512.0, 0.70276)	-60.0	(1606.01, 0.56659)	**(1314.0, 0.58304)**	12.8
R_{364}	(759.0, 0.41596)	**(759.0, 0.41804)**	49.2	(26572.0, 0.61983)	(13505.0, 0.60385)	-54.5	(2993.0, 0.45685)	**(1825.0, 0.46744)**	15.3
Di_{81}	(580.9, 0.83000)	**(580.9, 0.83218)**	43.6	(5913.0, 0.89903)	(3066.0, 0.89393)	-59.7	(3285.0, 0.89249)	**(3066.0, 0.89393)**	-59.7
Di_{100}	(660.0, 0.79495)	**(660.0, 0.79665)**	43.9	(7300.0, 0.87686)	(3723.0, 0.87061)	-60.5	(4599.0, 0.87214)	(3723.0, 0.87061)	-60.5
Di_{144}	(825.0, 0.71535)	**(825.0, 0.71903)**	44.7	(10512.0, 0.82760)	(5329.0, 0.81907)	-61.0	(6935.0, 0.82170)	(5329.0 0.81907)	-61.0
Di_{256}	(1155.0, 0.54076)	**(1155.0, 0.54940)**	46.6	(18688.0, 0.71434)	(9417.0, 0.70122)	-62.5	(12994.0, 0.70626)	(9417.0, 0.70122)	-62.5
Di_{361}	(1452.0, 0.41585)	**(1452.0, 0.42095)**	49.5	(26353.0, 0.62228)	(13286.0, 0.60623)	-62.7	(19126.0, 0.61335)	(13286.0, 0.60623)	-62.7

a cost function that also integrates both objectives but in a distinct manner than from MRCD. In C3, MRCD presented makespans which are almost half of those produced by RHEFT, while the reliabilities were very close.

The second set of experiments was performed with a varying number of processors, on the DAGs: G_{1034}, R_{546} and Di_{529}, as seen in Table 3. With an increasing number of processors, the makespan diminishes while reliability increases, since the scheduling strategies can allocate tasks on more appropriate processors concerning failure rates and computational slowdown indexes. In relation with the comparisons C1, C2 and C3, the same behaviour previously described is also detected.

9 The next sets of experiments were performed considering BSAMod of Class 2. The set S was generated by executing both MRCD and BSAMod with $W = \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$ for 24 processors, initially. A varying number of tasks and processors were also considered in these experiments. Table 4 shows the percentage difference of both makespan and reliability for each w value. The line $\mathcal{M}$ presents the percentage improvement on the makespan produced by MRCD over BSAMod and in line R_T, the percentage improvements on the reliability of BSAMod over MRCD. Note that an outstanding improvement on the makespan is produced by MRCD while the reliability remains almost the

Table 3. Comparison in Class 1 for G_{1034}, R_{546} and Di_{529} with a varying number m of processors

		C1			C2			C3		
DAG_N	m	HEFT	MRCD	D	RCDMod	MRCD	D	RHEFT	MRCD	D
G_{1034}	24	(1504.1, 0.86253)	**(1499.8, 0.86628)**	41.3	(44389.8, 0.92320	(22758.4, 0.91922)	-55.8	(8743.9, 0.88970)	**(5309.0, 0.89017)**	2.1
	45	(1335.8, 0.84661)	**(1335.8, 0.88030)**	35.2	(44389.8, 0.93558)	(23805.3, 0.92982)	-51.5	(9291.4, 0.91737)	**(4623.8, 0.91816)**	-11.0
	66	(1335.8, 0.85161)	**(1335.8, 0.89391)**	31.5	(22229.9, 0.95234)	(9640.3, 0.94698)	-49.8	(5673.5, 0.93053)	**(4940.6, 0.93929)**	-29.3
	87	(1335.8, 0.84610)	**(1335.8, 0.90048)**	28.4	(22229.9, 0.95234)	(7797.8, 0.94846)	-54.9	(4885.1, 0.93406)	**(4137.6, 0.93877)**	-25.5
	108	(1335.8, 0.84508)	**(1335.8, 0.90251)**	28.4	(44389.8, 0.95234)	(11617.2, 0.94738)	-54.4	(6556.8, 0.93440)	(4108.4, 0.93349)	-15.6
R_{546}	24	(1122.0, 0.87705)	**(1122.0, 0.87737)**	36.4	(39858.0, 0.93076)	(20513.0, 0.92717)	54.8	(4453.0, 0.88868)	**(2628.0, 0.89193)**	16.5
	45	(629.0, 0.87248)	**(627.0, 0.87358)**	57.7	(39858.0, 0.94196)	(20805.0, 0.93659)	-49.8	(2920.0, 0.89700)	**(2263.0, 0.90978)**	1.44
	66	(627.0, 0.87567)	**(627.0, 0.88199)**	46.5	(19929.0, 0.95710)	(8541.0, 0.95214)	-51.3	(2847.0, 0.90954)	**(1533.0, 0.91425)**	5.8
	87	(627.0, 0.87032)	**(627.0, 0.88323)**	49.3	(19929.0, 0.95710)	(6935.0, 0.95350)	-56.0	(2409.0, 0.91396)	**(1347.0, 0.91838)**	2.91
	108	(627.0, 0.86827)	**(627.0, 0.89088)**	39.7	(39858.0, 0.95710)	(10804.0, 0.95271)	-47.2	(2628.0, 0.91306)	**(1274.0, 0.91814)**	7.3
Di_{529}	24	(1881.0, 0.87795)	**(1881.0, 0.87919)**	51.2	(38617.0, 0.93284)	(19418.0, 0.92927)	-63.2	(29492.0, 0.93114)	(19418.0, 0.92927)	-63.2
	45	(1571.1, 0.86558)	**(1571.1, 0.87024)**	66.7	(38617.0, 0.94371)	(20367.0, 0.93856)	-52.1	(31828.0, 0.94131)	(20367.0, 0.93856)	-52.1
	66	(1505.1, 0.86760)	(1505.2, 0.87470)	65.3	(19418.0, 0.95841)	(8833.0, 0.95399)	-54.6	(16133.0, 0.95778)	(8833.0, 0.95399)	-54.6
	87	(1485.1, 0.86033)	(1485.2, 0.87114)	69.7	(19418.0, 0.95841)	(7081.0, 0.95513)	-59.6	(13067.0, 0.95719)	7081.0, 0.95513)	-59.6
	108	(1485.1, 0.86340)	(1485.2, 0.87542)	66.1	(38617.0, 0.95841)	(10293.0, 0.95412)	-57.8	(22630.0, 0.95688)	(10293.0, 0.95412)	-57.8

same as BSAMod. A set of similar experiments was carried out considering a varying number of processors, for the DAGs G_{1034}, R_{546} and Di_{529}. The main conclusion is that the improvements on the makespan provided by MRCD were significant at a cost of a small deterioration of the reliability for any number of processors, as can be seen in Table 5.

Table 4. Comparison between BSAMod and MRCD for G_{702}

DAG	0.00	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90	1.00	Average
M(%)	0.00	18.50	49.00	75.00	106.00	142.00	113.00	10.00	17.00	13.00	0.00	49.4
R_T (%)	0.00	2.80	3.90	5.60	7.80	8.20	5.20	-0.80	0.30	0.20	0.00	3.01

Finally, comparing all the solutions generated by MRCD with BSAMod for the values in W for G_{152}, G_{702}, Di_{81}, Di_{361} on 24 processors, it could be observed that all MRCD solutions were dominant or incomparable. In the case of R_{80} and R_{364}, in only one case MRCD was dominated by BSAMod. In summary, one can conclude the benefits of applying the implemented list scheduling algorithm together with the cost function $f(v, p_v)$ in MRCD. Actually, it was observed that,

Table 5. Comparison between BSAMod and MRCD for a varying number of processors

	G_{1034}			R_{546}			Di_{529}		
m	24	87	213	24	87	213	24	87	213
Avg. $\mathcal{M}(\%)$	47.2	41.1	41.1	-3.34	5.55	8.25	105	71.5	68.3
Avg. R_T (%)	0.61	1.41	1.28	-0.13	2.04	0.52	1.23	3.08	2.79

since the normalization procedure of BSAmod only divides by the maximum value of the given objective, it leaded to poor choices.

5 Concluding Remarks

This work proposes a weighted cost function and a classification of the solutions provided by MRCD. An experimental analysis shows that MRCD can find efficient solutions when compared with the other heuristics under evaluation. The comparison with HEFT shows the importance of using a bi-objective function that considers both minimization of the makespan and maximization of the reliability. From the evaluation of RCDMod and MRCD, one can conclude the importance of using an integrated bi-objective function, while the importance of the weights and the cost function provided by MRCD are shown when comparing with RHEFT. Finally, when considering BSA, which is also a weighted bi-objective scheduling heuristics, MRCD provides outstanding improvements on the makespan while keeping similar reliability results. As future work, MRCD will be used in conjunction with a fault tolerance approach that is primary-backup based, with the objective to recover real MPI applications from faults and execute them efficiently on distributed heterogeneous systems.

References

1. Qin, X., Jiang, H.: A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems. Parallel Computing 32(5), 331–356 (2006)
2. Dongarra, J., Jeannot, E., Saule, E., Shi, Z.: Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems. In: Proc. 19th Annual ACM Symp. on Parallelism in Algorithms and Architectures (SPAA '07). ACM Press, New York (2007)
3. Topcuouglu, H., Hariri, S., Wu, M.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. 13(3), 260–274 (2002)
4. Hakem, M., Butelle, F.: Reliability and scheduling on systems subject to failures. In: Proc. of the Int. Conf.e on Parallel Processing (ICPP), p. 38 (2007)
5. Gal, T., Hanne, T., Stewart, T. (eds.): Multicriteria Decision Making: Advances in MCDM Models, Algorithms, Theory, and Applications. Kluwer Academic, Dordrecht (1999)
6. Nascimento, A.P., Sena, A.C., Boeres, C., Rebello, V.E.F.: Distributed and dynamic self-scheduling of parallel MPI grid applications. Concurrency and Computation: Practice and Experience 19(14), 1955–1974 (2007)
7. Girault, A., Saule, 1., Trystram, D.: Reliability versus performance for critical applications. J. Parallel and Distrib. Computing 69(3), 326–336 (2009)