Minimizing Energy Consumption for Real-Time Tasks on Heterogeneous Platforms Under Deadline and Reliability Constraints

Yiqin Gao, Li Han, Jing Liu, Yves Robert, Frédéric Vivien

Minimizing energy consumption for real-time tasks on heterogeneous platforms under deadline and reliability constraints

Yiqin Gao[2†], Li Han[1*†], Jing Liu[1], Yves Robert[3,4] and Frédéric Vivien[3]

[1*]East China Normal University, China.
[2]Shanghai Jiao Tong University, China.
[3]Univ Lyon, EnsL, UCBL, CNRS, Inria, LIP, F-69342, LYON Cedex 07, France.
[4]University of Tennessee Knoxville, USA.

*Corresponding author(s). E-mail(s): hanli@sei.ecnu.edu.cn;
†These authors contributed equally to this work.

Abstract

As real-time systems are safety critical, guaranteeing a high reliability threshold is as important as meeting all deadlines. Periodic tasks are replicated to mitigate the negative impact of transient faults, which leads to redundancy and high energy consumption. On the other hand, energy saving is widely identified as increasingly relevant issues in real-time systems. In this paper, we formalize this challenging tri-criteria optimization problem, i.e., minimizing the expected energy consumption while enforcing the reliability threshold and meeting all task deadlines, and propose several mapping and scheduling heuristics to solve it. Specifically, a novel approach is designed to (i) map an arbitrary number of replicas onto processors, (ii) schedule each replica of each task instance on its assigned processor with less temporal overlap. The platform is composed of processing units with different characteristics, including speed profile, energy cost and fault rate. The heterogeneity of the computing platform makes the problem more complicated, because different mappings achieve different levels of reliability and consume different amounts of energy. Moreover, scheduling plays an important role in energy saving, as the expected energy consumption is the average over

all failure scenarios. Once a task replica is successful, the other replicas of that task instance can be canceled, which calls for minimizing the overlap between any replica pair. Finally, to quantitatively analyze our methods, we derive a theoretical lower-bound for the expected energy consumption. Comprehensive experiments are conducted on a large set of execution scenarios and parameters. The comparison results reveal that our strategies perform better than the random baseline under almost all settings, with an average gain in energy consumption of more than 40%, and our best heuristic achieves an excellent performance: its energy saving is only 2% less than the lower-bound on average.

Keywords: real-time systems, energy-aware systems, reliability, mapping, scheduling, heterogeneous platforms

1 Introduction

This paper focuses on energy minimization for the mapping and scheduling of real-time tasks under reliability constraints. An instance of each task is submitted periodically to a parallel computing platform and must complete execution before its deadline. The tasks have different deadlines, which typically calls for interrupting and resuming execution, e.g., in order to meet the deadline of a more urgent task that has been recently released. Many real-time applications are safety-critical, thus another constraint is to guarantee some reliability threshold for each task. This is because the execution of each task is prone to transient faults, so that several replicas of the same task must be executed in order to guarantee a prescribed level of reliability [2, 39]. Recently, several strategies have been introduced with the objective to minimize the expected energy consumption of the system while matching all deadlines and reliability constraints [13, 14].

This work aims at extending these energy-aware strategies in the context of heterogeneous platforms. Heterogeneous platforms have been used for safety-critical real-time systems for many years [8]. With the advent of multiple hardware resources such as multi-cores, GPUs, and FPGAs, modern computing platforms exhibit a high level of heterogeneity, and the trend is increasing. The multiplicity of hardware resources with very different characteristics in terms of speed profile, reliability level and energy cost, raises an interesting but challenging problem: given several device types, which ones should we keep and which ones should we discard, in order to achieve the best possible tri-criteria trade-off (time, energy, reliability)? Needless to say, this optimization problem is NP-hard, even with two identical error-free processors, simply because of matching deadlines.

This work provides several mapping and scheduling heuristics to solve the tri-criteria problem on heterogeneous platforms. To the best of our knowledge, these heuristics are the first ones that solve the problem with an unlimited number of replicas, thereby for arbitrary reliability thresholds. The design of

these heuristics is much more technical than in the case of identical processors. Intuitively, this is because the reliability of a replica of one task depends upon the processor which executes it, and is different for each task instance (which we define in Section 2.1 below). More precisely, the reliability of a replica of the j-th instance of i-th task mapped on processor m_k can be estimated by $R(\tau_{i,j}, m_k) = e^{-\lambda_k c_{i,k}}$, where $c_{i,k}$ is the worst case execution time (WCET, which is common for all instances of the same task) of task τ_i on m_k, and λ_k the failure rate of m_k. The total reliability of a task instance can be estimated by a function of the reliability of all its replicas (which we explicit in Equation 1 below); hence, it is not known until the end of the mapping process, unless we pre-compute an exponential number of reliability values. Then there are many processors to choose from, and those providing a high reliability, thereby minimizing the number of replicas needed to match the reliability threshold, may also require a high energy cost per replica: in the end, it might be better to use less reliable but also less energy-intensive processors. Furthermore, the reliability is not enough to decide for the mapping: if two processors offer similar reliabilities for a task, it might be better to select the one with smaller execution time, in order to increase the possibility of mapping other tasks without exceeding any deadline. Altogether, we face a complicated decision, and we provide several criteria to guide the mapping process.

Overall, the objective is to minimize the expected energy consumption while matching all deadlines and reliability constraints. The expected energy consumption is the average energy consumed over all failure scenarios. Consider a sample execution: whenever the execution of a task replica succeeds, all the other replicas are instantaneously deleted; therefore, the actual amount of energy consumed depends both upon the error scenario (which replica is the first successful) and upon the overlap between replicas (some replicas are partially executed and interrupted when the successful one completes). Given a mapping, the scheduling phase aims at reducing overlap between any two replicas of the same task. Note that having an overlap-free scheduling is not always possible, because of utilization constraints. Also, deciding whether an overlap-free scheduling exists for a given mapping is NP-hard [11], even for deterministic tasks. In a simpler version of the problem, when all tasks have the same period/deadline, preemptive scheduling is not needed because all tasks have equal priority, which greatly simplifies the design [12]. In the general case addressed in this paper, scheduling is much more complicated because of the different deadlines for each task, and the opportunity for optimizing energy is greatly reduced.

Finally, in actual real-time systems, tasks often complete before their worst-case execution time, or WCET, so that execution times are routinely modeled as stochastic. For instance, one typically assumes that the execution time of every instance of task τ_i on m_k follows a common uniform probability distribution in the range $[\beta_{\mathrm{b/w}} c_{i,k}, c_{i,k}]$ for some constant $\beta_{\mathrm{b/w}} < 1$ (ratio of best case over worst case). In the end, the expected energy consumption must also be averaged over all possible values for execution times in addition to over all

failure scenarios. To assess the performance of our heuristics, we use a comprehensive set of execution scenarios, with a wide range of processor speed profiles and failure rates. When the failure rate is low, most heuristics are equivalent, but when the failure rate is higher, only a few heuristics achieve good performance. Because we have no guarantee on the performance of the global mapping and scheduling process, we analytically derive a lower-bound for the expected energy consumption of any mapping. This bound cannot always be met. Nevertheless, we show that the performance of our best heuristics remains quite close to this bound in the vast majority of simulation scenarios.

The main contributions of the paper are the following:

1. The formulation of the tri-criteria optimization problem;
2. The design of the first mapping and scheduling heuristics that solve the problem with an unlimited number of replicas, thereby for arbitrary reliability thresholds;
3. The characterization of a lower-bound for energy consumption;
4. An experimental evaluation based on a comprehensive set of simulations scenarios, showing that our best heuristics achieve a significant gain over the whole spectrum of application and platform parameters.

The rest of the paper is organized as follows. Section 2 provides a detailed description of the optimization problem under study, including a few notes on its complexity. The mapping and scheduling heuristics are described in Section 3 and 4 respectively. The lower-bound of energy consumption is introduced in Section 5. Section 6 is devoted to a comprehensive experimental comparison of the heuristics. Section 7 presents related work. Finally, Section 8 gives concluding remarks and hints for future work.

2 Model

The inputs to the optimization problem are a set of real-time independent tasks, a set of non-identical processors and a reliability target. Key notations are summarized in Table 1.

2.1 Platform and tasks

The platform consists of M heterogeneous processors m_1, m_2, ..., m_M and a set of N periodic atomic tasks $\tau_1, \tau_2, \ldots, \tau_N$. Each task τ_i has WCET $c_{i,k}$ on processor m_k. The WCETs among different processors are not necessarily related. In the experiments, we generate the $c_{i,k}$ values with the method proposed in [1], where we have two parameters to control the correlation among task execution times and processors (see Section 6.1 for details). Each periodic task τ_i generates a sequence of *instances* with period p_i, which is equal to its deadline. The j-th instance of task τ_i is released at date $(j-1)p_i$ and its deadline is jp_i. The whole input pattern repeats every hyperperiod of length $L = \text{lcm}_{1 \leq i \leq n} p_i$. Each task τ_i has $\frac{L}{p_i}$ instances within the hyperperiod.

Table 1: Key Notations

Notation	Explanation
N and M	number of tasks and of processors
τ_i	the i-th task
$\tau_{i,j}$	the j-th instance of the i-th task
m_k	the k-th processor
p_i	period (deadline) for each instance of task τ_i
$L = \text{lcm}_{1 \leq i \leq n} p_i$	hyperperiod of the system
$w_{i,j,k}$	actual execution time for task instance $\tau_{i,j}$ on processor m_k
$c_{i,k}$	WCET for task τ_i on processor m_k
$u_{i,k} = \frac{c_{i,k}}{p_i}$	utilization of task τ_i executing on processor m_k
u_k	utilization of m_k (sum of utilization of replicas assigned to m_k)
$\mathcal{R}_i$	target reliability threshold for task τ_i
λ_k	failure rate of processor m_k
$R(\tau_{i,j}, m_k)$	reliability of task $\tau_{i,j}$ on processor m_k
$P_{k,s}$	static power consumed per time unit on processor m_k
$P_{k,d}$	dynamic power consumed per time unit on processor m_k
E_s	total static energy consumption of the system
E_d	total dynamic energy consumption of the system
$E_d(\tau_{i,j}, m_k)$	dynamic energy cost of task instance $\tau_{i,j}$ on processor m_k

As already mentioned, real-time tasks usually complete execution earlier than their estimated WCET: actual execution times are assumed to be data-dependent and non-deterministic, randomly sampled from some probability distribution whose support is upper bounded by the WCET. See Section 6.1 for details on the generation of actual execution times from WCET values. The *utilization* $u_{i,k}$ of task τ_i executing on processor m_k is defined as $u_{i,k} = \frac{c_{i,k}}{p_i}$. The utilization of a processor is the sum of the utilizations of all task instances that are assigned to it.

The execution of any task can be preempted, that is, to be temporarily stopped (for instance to allow for the execution of another task) and later resumed without any induced penalty.

2.2 Reliability

We consider transient faults, modeled by an exponential probability distribution of rate λ_k on processor m_k. Thus, fault rates differ from one processor to another*. This is a very natural assumption for a heterogeneous platform made of different-type processors. At the end of the execution of each task, there is an *acceptance test* to check the occurrence of soft errors induced by

*It is also assumed that transient faults hit tasks independently of one another, even on the same processor. This is questionable because some fault inter-dependence is expected in practice, either spatially (same processor) or temporally (bursts). However, this is the approach adopted by all relevant literature, such as [4, 21, 37, 38], due to intractability of dealing with non-independent faults.

the transient faults. It is assumed that acceptance tests are 100% accurate[†], and that the duration of the test is included within the task WCET [14].

The *reliability* of a task instance is the probability of executing it successfully without software faults. It is related to its execution time. But in our problem, only the tasks WCETs are known, but not the actual execution times. Thus, in our heuristics, we use the WCETs to estimate the reliability of task instances and to make mapping decisions. The reliability of task instance $\tau_{i,j}$ on processor m_k with WCET $c_{i,k}$ can be estimated by $R(\tau_{i,j}, m_k) = e^{-\lambda_k \times c_{i,k}}$. Two instances $\tau_{i,j}$ and $\tau_{i,j'}$ of the same task τ_i have the same estimated reliability if they are executed on the same processor, because their WCETs are the same. But the actual reliability during execution is not the same ($e^{-\lambda_k \times w_{i,j,k}}$ and $e^{-\lambda_k \times w_{i,j',k}}$ respectively) because of their different actual execution times.

Note that a task instance cannot be replicated twice on the same processor. Thus, in the final schedule, task instance $\tau_{i,j}$ may have several replicas executing on different processors, in order to match its reliability threshold. Let $alloc(i,j)$ denote the index set of the processors executing a replica of $\tau_{i,j}$. The mapping achieves the following reliability $R(\tau_{i,j})$ for task instance $\tau_{i,j}$:

$$R(\tau_{i,j}) = 1 - \Pi_{k \in alloc(i,j)}(1 - R(\tau_{i,j}, m_k)) \tag{1}$$

Indeed, the task will succeed if at least one of its replicas does. The success probability is thus equal to 1 minus the probability of all replicas failing, which is the expression given in Equation (1). It also means that all other replicas (executing or not started) can be canceled when the first one is successfully finished.

Each instance $\tau_{i,j}$ of task τ_i has a reliability threshold $\mathcal{R}_i$ which is an input of the problem and that must be met by the mapping. In other words, the constraint writes $R(\tau_{i,j}) \geq \mathcal{R}_i$ for $1 \leq i \leq N$ and $1 \leq j \leq \frac{L}{p_i}$. This threshold is always satisfied during the actual execution, although we use the estimated reliability for mapping; this is because the actual execution time can never be greater than the WCET and, thus, the actual reliability is never smaller than the estimated one.

Because the tasks are independent, it is natural to assume that they might have different reliability thresholds: a higher threshold means that more resources should be assigned for the task to complete successfully with a higher probability. In the experiments we use $\mathcal{R}_i = \mathcal{R}$ for all tasks, but our heuristics are designed to accommodate different thresholds per task.

2.3 Power and energy

The power consumed per time unit on processor m_k is composed of two parts, static power ($P_{k,s}$) and dynamic power ($P_{k,d}$). The static power is consumed

[†]More precisely, it is assumed that acceptance tests are reliable enough to guarantee the absence of false negatives (recall) and false positives (precision) with a high probability, so that one can assume correctness of the test in practice.

permanently if the processor m_k is used in the schedule. In contrast, the dynamic power is consumed only if the processor is actually executing a task. To summarize, we have $2M$ input values, $\{P_{1,s}, P_{2,s} \dots P_{M,s}\}$ for static powers and $\{P_{1,d}, P_{2,d} \dots P_{M,d}\}$ for dynamic powers.

The total static energy consumption of the system during one hyperperiod is simply given by

$$E_s = \sum_{k \in Used} P_{k,s} \times L \tag{2}$$

where *Used* denotes the index set of the processors used by the schedule.

Similarly to Section 2.2, the dynamic energy consumption of one replica of task instance $\tau_{i,j}$ on processor m_k is estimated using the WCET $c_{i,k}$ of task τ_i:

$$E_d(\tau_{i,j}, m_k)^{estimated} = P_{k,d} \times c_{i,k} \tag{3}$$

in which we use the same value $c_{i,k}$ for all instances $\tau_{i,j}$ of the same task τ_i on m_k because they have the same WCET. Thus, the estimated dynamic energy consumption during one hyperperiod, which is used to design the schedule, is expressed as

$$E_d^{estimated} = \sum_{(i,j,k)|k \in alloc(i,j)} E_d(\tau_{i,j}, m_k)^{estimated} \tag{4}$$

But the actual energy consumption of $\tau_{i,j}$ will likely be lower than stated in Equation (3). First the actual execution time $w_{i,j,k}$ is typically smaller than the WCET $c_{i,k}$. In addition, recall that we cancel all other replicas of a task instance as soon as one of its replicas has successfully completed. Thus, the execution time of a replica can be reduced or even zeroed out by the success of another replica of the same task instance.

We define a scenario $s = \{W, F\}$ as the random drawing of the execution times $w_{i,j,k}$ and of the failure booleans $f_{i,j,k}$ for all (i, j, k) where $k \in alloc(i, j)$. $f_{i,j,k} = True$ if the replica of task instance $\tau_{i,j}$ executed on processor m_k can be successfully completed, and $f_{i,j,k} = False$ if it will be victim of a failure. Again, some replicas will be interrupted or never launched, depending upon the scenario. Given a scenario sc, we let $r_{i,j,k}$ denote the actual execution times of each replica of each task instance. These values are dynamically computed as the execution of the schedule progresses. We compute the actual dynamic energy as

$$E_d(\tau_{i,j}, m_k)_{sc}^{actual} = P_{k,d} \times r_{i,j,k} \tag{5}$$

$$E_{d,sc}^{actual} = \sum_{(i,j,k)|k \in alloc(i,j)} E_d(\tau_{i,j}, m_k)_{sc}^{actual} \tag{6}$$

Finally, we weight each scenario $sc \in S$ by its probability p_{sc}, where S is the set of all scenarios, and compute the expected dynamic energy consumption during one hyperperiod:

$$E_d^{expected} = \sum_{sc \in S} (p_{sc} * E_{d,sc}^{actual}) \tag{7}$$

Altogether, the expected energy consumption of the schedule during one hyperperiod becomes:

$$E^{expected} = E_s + E_d^{expected} \tag{8}$$

The optimization objective is to find the schedule which minimizes this value. Clearly, there is no closed-form formula for the expected energy consumption, whose value is approximated by Monte-Carlo simulations.

2.4 Optimization Objective

The objective is to determine a set of replicas for each task, a set of processors to execute them, and to build a schedule of length at most L, so that the expected energy consumption is minimized, while matching the reliability threshold $\mathcal{R}_i$ for each task τ_i and the deadline $j \times p_i$ of each task instance $\tau_{i,j}$. As already mentioned in Section 1, the expected energy consumption is an average made over all possible execution times randomly drawn from their distributions, and over all failure scenarios (with every component weighted by its probability to occur). An analytical formula is out of reach, and we use Monte-Carlo sampling in the experiments. However, we stress the following two points:

- To guide the design of the heuristics, we use a simplified objective function; more precisely, we use WCETs instead of (yet unknown) actual execution times, and we conservatively estimate the dynamic energy of a task as the sum of the dynamic energy of all its replicas. Because mapping decisions are based upon WCETs, the number of enrolled processors does not depend upon actual execution times and the static energy is always the same for all scenarios, namely the length of the period times the sum of the static powers of the enrolled processors (see Equation (2)).
- To assess the absolute performance of the heuristics, we derive a lower-bound for the dynamic energy. This bound is based upon actual execution times but neglects scheduling constraints and assumes no overlap between any two task replicas; hence it is not reachable in general. However, we show that our best heuristics achieve performance close to this bound.

2.5 Complexity

The global optimization problem is obviously NP-hard, since it is a generalization of the makespan minimization problem with a fixed number of parallel processors [7]. The optimization of the sole scheduling phase is also NP-hard for identical processors: if the number of replicas has already been decided for each task, and if the assigned processor of each replica has also been decided, the scheduling phase aims at minimizing the expected energy consumption by avoiding overlap between the replicas of a same task [11]. In fact, the heterogeneity of the processors, both in terms of power cost and fault rate, dramatically complicates the problem. Proposition 1 shows that the problem

remains NP-complete when mapping identical and deterministic tasks with same period onto processors with different fault-rates, even without any energy constraint; in other words, choosing the optimal subset of processors to execute each task and its replicas to match their reliability threshold is an intrinsically combinatorial problem! We formally state the corresponding decision problem:

Definition 1 (RELIABILITY) *Consider n identical and deterministic tasks τ_i with same period and deadline $p = 1$, and same reliability threshold $\mathcal{R}$, to be executed on a platform composed of M heterogeneous processors. On processor m_k, $1 \leq k \leq M$, each task τ_i has execution time $c_{i,k} = 1$ and reliability $R(\tau_i, m_k) = R_k$. Is it possible to map all tasks so as to match the reliability threshold for each task?*

Because the execution time on each processor is always equal to the period/deadline, the hyperperiod of length $L = 1$ is composed of a single instance of each task, and all replicas will necessarily be executed in parallel in the time interval $[0, 1]$; hence each processor executes at most one replica.

Proposition 1 *RELIABILITY is NP-Complete.*

Proof The problem is obviously in NP. For the completeness, we use a reduction from 3-PARTITION [7]. Consider an arbitrary instance $\mathcal{I}_1$ of 3-PARTITION: given $3m$ positive integers a_i, $1 \leq i \leq 3m$ with $\frac{B}{4} < a_i < \frac{B}{2}$ and $\sum_{i=1}^{3m} a_i = mB$, can we find a partition of $\{1, 2, \ldots, 3m\}$ into m subsets I_j such that $\sum_{i \in I_j} a_i = B$ for $1 \leq j \leq m$? We build the following instance $\mathcal{I}_2$ of RELIABILITY: we have $M = 3m$ processors and $n = m$ tasks. A replica on processor m_k, $1 \leq k \leq M$, has reliability $R_k = \frac{a_k}{A}$ where $A = 6B^2 + 2B^3$. Finally, the target reliability threshold for each task is $\mathcal{R} = \frac{B}{A}(1 - 3\frac{B}{A})$. The size of $\mathcal{I}_2$ is clearly polynomial (and even linear) in the size of $\mathcal{I}_1$.

We now show that $\mathcal{I}_2$ has a solution if and only if $\mathcal{I}_1$ does. Assume first that $\mathcal{I}_1$ has a solution and let I_j denote the m subsets of the partition. Note that each subset has exactly three elements because on the condition on the size of the a_i's. For $\mathcal{I}_2$, we map task τ_i, $1 \leq i \leq n = m$, on the three processors whose index belongs to I_i. This is a valid mapping onto the $M = 3m$ processors because the m subsets form a partition of $\{1, 2, \ldots, M\}$. Each task has three replicas. Writing $I_i = \{i_1, i_2, i_3\}$, the reliability of τ_i writes

$$\begin{aligned} \mathcal{R}_i &= 1 - \textstyle\prod_{q=1}^{3}(1 - R_{i_q}) \\ &= \frac{\sum_{q=1}^{3} a_{i_q}}{A} - \frac{\sum_{1 \leq q < r \leq 3} a_{i_q} a_{i_r}}{A^2} + \frac{a_{i_1} a_{i_2} a_{i_3}}{A^3} \\ &\geq \frac{\sum_{q=1}^{3} a_{i_q}}{A} - \frac{\sum_{1 \leq q < r \leq 3} a_{i_q} a_{i_r}}{A^2} \end{aligned} \tag{9}$$

Since $\sum_{q=1}^{3} a_{i_q} = B$ and $a_i < \frac{B}{2}$ for all $1 \leq i \leq 3m$, we obtain

$$\mathcal{R}_i \geq \frac{B}{A} - \frac{3B^2}{4A^2} \geq \frac{B}{A} - \frac{3B^2}{A^2} = \mathcal{R}$$

Hence, $\mathcal{I}_2$ has a solution.

Assume now that $\mathcal{I}_2$ has a solution, and let I_i denote the indices of the processors executing replicas of task τ_i for $1 \leq i \leq n$. The subsets I_i are pairwise disjoint, but we do not know whether they form a partition yet (some processors may have not been enrolled in the mapping). But assume that some I_i has only 1 element i_1: we get $\mathcal{R}_i = \frac{a_{i_1}}{A} \geq \mathcal{R}$. This leads to

$$a_{i_1} \geq B\left(1 - 3\frac{B}{A}\right) = B - \frac{3B^2}{A} \geq B - \frac{1}{2}$$

The last inequality comes from the definition of A which has been chosen large enough. But $a_{i_1} < \frac{B}{2} < B$ implies $a_{i_1} \leq B - 1$ since we have integers. Hence, a contradiction. Similarly, if some I_i has only 2 elements i_1 and i_2: we get

$$\mathcal{R}_i = \frac{a_{i_1} + a_{i_2}}{A} - \frac{a_{i_1}a_{i_2}}{A^2} \geq \mathcal{R}$$

hence $\frac{a_{i_1}+a_{i_2}}{A} \geq \mathcal{R}$. We conclude as before, because it implies that $a_{i_1} + a_{i_2} \geq B - \frac{1}{2}$, while $a_{i_1} + a_{i_2} < 2\frac{B}{2} = B$ implies $a_{i_1} + a_{i_2} \leq B - 1$. As a consequence, each subset I_i has at least 3 elements. We do have a partition of $\{1, 2, \ldots, 3m\}$, and each subset I_i has exactly 3 elements. Then, writing $I_i = \{i_1, i_2, i_3\}$ as before, we have $\mathcal{R}_i \geq \mathcal{R} = \frac{B}{A}(1 - 3\frac{B}{A})$ for all $1 \leq i \leq m$, where $\mathcal{R}_i$ is given by Equation (9). We derive that

$$\begin{aligned}\frac{\sum_{q=1}^{3} a_{i_q}}{A} &\geq \frac{B}{A}\left(1 - 3\frac{B}{A}\right) + \frac{\sum_{1 \leq q < r \leq 3} a_{i_q}a_{i_r}}{A^2} - \frac{a_{i_1}a_{i_2}a_{i_3}}{A^3} \\ &\geq \frac{B}{A}\left(1 - 3\frac{B}{A}\right) - \frac{a_{i_1}a_{i_2}a_{i_3}}{A^3}\end{aligned}$$

As $a_i < \frac{B}{2}$ for $1 \leq i \leq 3m$, we can deduce:

$$\frac{\sum_{q=1}^{3} a_{i_q}}{A} \geq \frac{B}{A}\left(1 - 3\frac{B}{A}\right) - \frac{B^3}{8A^3}$$

Recall that $A = 6B^2 + 2B^3$ and B is positive integer. Hence,

$$\begin{aligned}\sum_{q=1}^{3} a_{i_q} &\geq B\left(1 - 3\frac{B}{A}\right) - \frac{B^3}{8A^2} \geq B\left(1 - 3\frac{B}{A}\right) - \frac{B^3}{A} \\ &\geq B - \frac{3B^2 + B^3}{A} \geq B - \frac{1}{2}\end{aligned}$$

Then $\sum_{q=1}^{3} a_{i_q} \geq B$ because we have integers. But the sum of the $3m$ integers a_i is mB. Hence, all the sums are indeed equal to B, and we have a solution to $\mathcal{I}_1$. This concludes the proof. □

3 Mapping

In the mapping phase, we need to define the number of replicas for each task instance, as well as the execution processor for each replica, aiming at meeting the reliability target and deadline, while minimizing the energy cost. As hyperperiods are repetitive, we aim at finding a feasible mapping within one hyperperiod (time length L). One difficulty introduced by platform heterogeneity is that we do not know the number of replicas needed for each task

Table 2: Example

m_k	$E(\tau_i, m_k)$	$R(\tau_i, m_k)$	$\frac{R(\tau_i,m_k)}{E(\tau_i,m_k)}$	$-\frac{\log_{10}(1-R(\tau_i,m_k))}{E(\tau_i,m_k)}$
1	1	0.9	0.9	1
2	1	0.9	0.9	1
3	2	0.99	0.495	1
4	1	0.99	0.99	2
5	2	0.9	0.45	0.5

instance to reach its reliability threshold, before completing the mapping process, because different processors have different failure rates and speeds and, hence, they provide different reliabilities for each replica. Therefore, the simpler three-step method of [11, 14] cannot be applied.

Since the estimation of dynamic energy consumption $(E_d(\tau_{i,j}, m_k)^{estimated})$ and reliability $(R(\tau_{i,j}, m_k))$ of task instance $\tau_{i,j}$ on processor m_k do not depend on the index of instance j, these two values will be the same for all instances of the same task. Thus, in the following description, instead of making decision for each task instance $\tau_{i,j}$, we will use $E_d(\tau_i, m_k)$ and $R(\tau_i, m_k)$ as the estimated dynamic energy consumption and reliability on processor m_k, for all instances of task τ_i, and we will make the same mapping decision for all instances of the same task.

When mapping all the $\frac{L}{p_i}$ instances of a given task on a processor, we use the standard *Earliest Deadline First (EDF)* scheduling heuristic [18]. EDF tells us that a given processor is a fit for that replica if and only if the utilization of that processor does not exceed 1. Recall that the utilization of a processor is the sum of the utilizations of all task instances assigned to it.

As shown in Algorithm 1, given a set of tasks with their periods and reliability targets and a set of heterogeneous processors, we first order the tasks according to TaskMapCriterion, which can be either:

- *deW* (*inW*): decreasing (increasing) average work size $\bar{c_i} = \frac{c_{i,1}+c_{i,2}+\cdots+c_{i,M}}{M}$;
- *deMinW* (*inMinW*): decreasing (increasing) minimum work size $\bar{c_i} = \min_{1\leq k\leq M} c_{i,k}$;
- *deMaxW* (*inMaxW*): decreasing (increasing) maximum work size $\bar{c_i} = \max_{1\leq k\leq M} c_{i,k}$;
- *random*: random ordering.

Then, for each task in the ordered list, we order the processors for mapping its replicas according to ProcMapCriterion, which can be either:

- *inE*: increasing energy cost;
- *deR*: decreasing reliability;
- *deP*: decreasing ratio of $-\frac{\log_{10}(1-R(\tau_i,m_k))}{E(\tau_i,m_k)}$ (explained below);
- *random*: random ordering.

We use the example shown in Table 2 to explain how to design a better criterion in ProcMapCriterion. Assume there are five processors with

different energy and reliability configurations (the first two processors are identical). Considering only the reliability, we cannot distinguish between the third and fourth processors. Apparently, the fourth processor is better since it consumes less energy and provide the same level of reliability. The problem is the same when ordering processors only according to energy cost. This gives us a hint that we need to consider energy and reliability interactively. A first idea would be to use the ratio $\frac{R(\tau_i,m_k)}{E(\tau_i,m_k)}$, which expresses the reliability per energy unit of every instance of task τ_i executing on processor m_k. But consider a task instance with a reliability target $\mathcal{R}_i = 0.98$: it requires either the third processor alone or the first two processors together. Both solutions match the reliability goal with the same energy cost 4. We aim at a formula that would give the same weight to both solutions. The ratio $-\frac{\log_{10}(1-R(\tau_i,m_k))}{E(\tau_i,m_k)}$ is a good candidate, because the total energy cost is the sum of all processors while the reliability is a product. This discussion explains how we have derived the third criterion *deP* in ProcMapCriterion, namely to order processors by decreasing ratio of $-\frac{\log_{10}(1-R(\tau_i,m_k))}{E(\tau_i,m_k)}$.

For the mapping phase, for task τ_i, we add replicas for all its instances $\tau_{i,j}$ in the order of the processor list until the reliability target $\mathcal{R}_i$ of each instance is reached. The algorithm uses the probability of failure $PoF = 1 - R(\tau_i) = \Pi_{k \in alloc(i)}(1 - R(\tau_i, m_k))$ (Equation (1)). The mapping process always ensures that: (i) no two replicas of the same task are assigned to the same processor; (ii) the utilization u_k of each processor does not exceed 1.

4 Scheduling

In the scheduling phase, we aim at ordering the tasks mapped on each processor, with the objective to further minimize the energy consumption during execution. Recall that the success of any replica leads to the immediate cancellation of all the remaining replicas, a crucial source of energy saving. Thus, our approaches identify a primary replica for each task instance, then all other replicas for that instance become secondaries. The goal of the proposed scheduling heuristics is to avoid overlaps between the execution of the primary and secondary replicas of each task instance: the primary must be terminated as soon as possible, while the secondaries must be delayed as much as possible.

After the mapping phase, we have a static schedule on each processor, also called the *canonical* schedule, which is based upon the WCET of each task and EDF. The EDF schedule can use preemption, so that a given task instance may be split into several chunks. As a result, from the canonical schedule, each processor has an ordered list made up with all task instance chunks that it has to execute during the hyperperiod, together with their starting and finish times. As the actual execution time of a real-time task instance will be shorter than its WCET, the canonical schedule will never be executed exactly as such. Still, it can be used as the baseline to compute the maximum delay for secondary replicas without missing any deadline.

Algorithm 1 Replication setting and mapping

Input: A set of tasks τ_i with reliability targets $\mathcal{R}_i$ and periods p_i; a set of heterogeneous processors m_k

Output: An allocation σ_m of all replicas on the processors

```
1: Order tasks with TaskMapCriterion and renumber them τ_1, ..., τ_N
2: u ← [0, ..., 0]                    ▷ initialize the utilization of all processors to zero
3: for i ∈ [1, ..., N] do              ▷ iterate through the ordered list of tasks
4:     Order processors for task τ_i with ProcMapCriterion and renumber
       them m_1, ..., m_M   ▷ order processors for each task; this ordered list may
       be task dependent
5:     k = 1
6:     PoF = 1
7:     while 1 − PoF < R_i do
8:         u_temp = u_k + u_{i,k}   ▷ Utilization of m_k after adding a replica of τ_i
9:         if u_temp ≤ 1 then
10:            u_k = u_temp
11:            PoF = PoF × (1 − R(τ_i, m_k))
12:            For all instances of τ_i, add a replica on m_k
13:        end if
14:        k + +
15:        if k > M then
16:            return not feasible
17:        end if
18:    end while
19: end for
20: return σ_m
```

To determine the primary replica for each task instance, we could make the decision offline or online. The offline strategy means that we use pre-knowledge before real execution, where we consider the following criteria:

- EDF_WCET: choose the processor that can execute the replica the fastest;
- EDF_Energy: choose the processor that can execute the replica with the minimum dynamic energy consumption;
- EDF_Reliability: choose the processor that can execute the replica the most reliably.

Note that for offline strategies, the primary replicas for different task instances of the same task are on the same processor. After determining the primary replicas, we deploy the following techniques based on the canonical schedule to differentiate the primary replica and the secondary replicas. The techniques are accompanied by figures, in which different colors represent different tasks, and primary replicas are marked with a darker color:

- Scale the WCETs of all tasks by $\frac{1}{\alpha}$, where α is the utilization, which also gives a valid canonical schedule, but with longer worst case expected execution times for all tasks, which we call the scaled canonical schedule

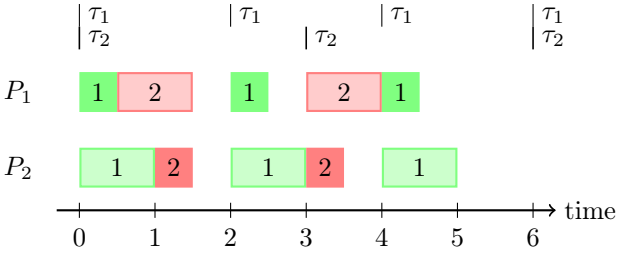
as shown in Figure 1. This gives a better reference to further delay the start time of secondary replicas. Here is why: at the mapping phase, as long as the total utilization of replicas that are mapped onto the processor is less than or equal to one, then we are able to find a valid scheduling using EDF. Assume we have mapped three tasks t_i, t_j, t_k onto processor p, and that the utilization is $\alpha = \frac{c_{i,p}}{p_i} + \frac{c_{j,p}}{p_j} + \frac{c_{k,p}}{p_k}$. Either we keep the mapping and have a fraction $1 - \alpha$ of the interval where p is idle, or we artificially slow down the execution times of all three tasks by a factor α, then we will have a new utilization $\beta = \frac{c_{i,p}}{p_i\alpha} + \frac{c_{j,p}}{p_j\alpha} + \frac{c_{k,p}}{p_k\alpha} = 1$, which also gives us a feasible mapping without any idle time. Then we could delay the start time of secondary replicas without conflict timeliness as long as they will finish executions not later than their finish times in the scaled canonical schedule (Figure 2).

- Consider a *scheduling interval* defined by two consecutive deadlines in the canonical schedule. Inside the interval, task chunks to be executed are ordered by the EDF policy. We observe that we can freely reorder the chunks without missing any deadline, by definition of an interval. It means that in each interval, we could reorder to execute all primary replicas first, and then secondary replicas (Figure 3).
- Since we have delayed the start time of secondary replicas, there are some idle slots in the schedule. We could take advantage of these idle slots by pre-fetching other primary replica chunks in the availability list: those primaries that have been released but which were scheduled later because they have lower EDF priority than the current secondary replicas.

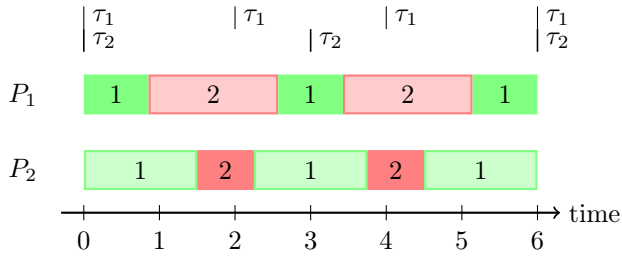
Based on the above ideas, considering one interval in the improved static schedule, it: 1) starts with the primary replica chunks; 2) fills in the idle slot by inserting other primary replica chunks that are available; 3) ends by the secondary replica chunks. Then, during the real execution, according to the failure case, the scheduler will dynamically cancel the execution of some secondaries if the corresponding primary replica finished successfully. If the replica selected as primary is the least expensive energy-wise (like the one selected by EDF_ENERGY), and if no secondary replica has started executing yet, then the energy consumption will be minimal for that task. However, because the choice of the primary replica is highly dependent on the characteristics of processors, different tasks may have their primary replicas on the same processor, which would lead to load imbalance and heavy overlaps. To avoid this situation, we design another type of method to choose primary replicas on-the-fly:

- EDF_START_TIME: choose the processor that can start the execution of the replica the earliest (given already made scheduling decisions).

This online strategy dynamically chooses the replica starting the earliest as the primary replica for each task instance. This tends to spread primaries onto different processors, which gives more slack time on each processor to reduce overlaps. As a side note, it will possibly be the case that two different instances of the same task have not the same primary processor. Then all other replicas of the same task become secondaries and are delayed according to the



(a) Canonical schedule



(b) Scaled canonical schedule

Figure 1: Example of scaling WCETs with two tasks τ_1 and τ_2 ($c_{1,1} = c_{2,2} = 0.5$, $c_{1,2} = c_{2,1} = 1$, $p_1 = 2$ and $p_2 = 3$) on each processor with total utilization $u_1 = \frac{7}{12}$ and $u_2 = \frac{4}{6}$.

scaled canonical schedule, as shown in Figure 2. We could not implement the "reordering inside intervals" shown in Figure 3 for EDF_Start_Time, which means that the end time of secondaries will not exceed the one planned in the scaled canonical schedule, as this is an offline improvement that requires the primary replica known in advance. But the online strategy has two major advantages compared to the offline strategies: (1) as long as we finish one replica successfully, we can safely cancel other replicas of the same task instance earlier than in offline schedules, which gives more flexibility to adjust the schedule afterwards; (2) the static schedules have to reserve time slots for the secondaries that correspond to their WCET (it is impossible to know their actual execution times before execution). Since their actual execution times are usually shorter, this dynamically frees some time slots that the online schedule uses to prefetch available primary replica chunks.

5 Lower bound

In this section, we explain how to derive a lower-bound for the expected energy consumption of a solution to the optimization problem, namely a mapping/scheduling heuristic that uses some of the selection criteria outlined in Sections 3 and 4.

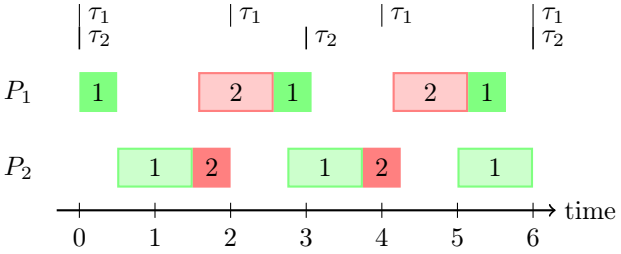


Figure 2: Static schedule when prioritizing primaries while delaying secondaries according to their finish times in the Scaled canonical schedule.

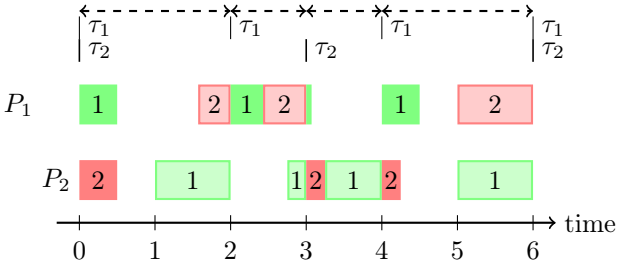


Figure 3: Reordering chunks freely inside intervals.

For each problem input, namely N tasks τ_i with reliability thresholds $\mathcal{R}_i$, M processors m_k with failure rates λ_k, and with all WCET $c_{i,k}$, we compute a solution, i.e., a mapping and scheduling of all replicas. We first use Monte-Carlo simulations (see Section 6) and generate several sets of values for the actual execution time $w_{i,j,k}$ of task instance $\tau_{i,j}$ on processor m_k. The values $w_{i,j,k}$ are drawn *uniformly across processors* as some fraction of their WCET $c_{i,k}$ (refer to Section 6.1 for details).

Now, for each set of values $w_{i,j,k}$, we generate a set of failure scenarios, compute the actual energy consumed for each scenario, and report the average of all these values as the expected energy consumption. A failure scenario operates as follows. We call an event the end of the execution of a replica on some processor. At each event, we flip a biased coin (weighted with the probability of success of the replica on that processor) to decide whether the replica is successful or not. If it is, we delete all other replicas of the same task instance. At the end of the execution, we record all the dynamic energy that has been actually spent, accounting for all complete and partial executions of replicas, and we add the static energy given by Equation (2). This leads to the energy consumption of the failure scenario. We average the values over all failure scenarios and obtain the expectation, denoted as $E(\{w_{i,j,k}\})$.

In addition, we also compute a lower-bound $LB(\{w_{i,j,k}\})$ as follows. Our goal is to accurately estimate the energy consumption of an optimal solution. Since the static energy depends upon the subset of processors that are used

in the solution (see Equation (2)), we need to try all possible subsets. Given a processor subset $\mathcal{S}$, we consider each task instance $\tau_{i,j}$ independently, and try all possible mappings of replicas of $\tau_{i,j}$ using only processors in $\mathcal{S}$. Thus we explore all subsets $\mathcal{T}$ of $\mathcal{S}$. A subset $\mathcal{T}$ is *safe* if mapping a replica of $\tau_{i,j}$ on each processor of $\mathcal{T}$ meets the reliability criterion $\mathcal{R}_i$, and if no strict subset of $\mathcal{T}$ is safe. Note that safe sets are determined using the WCETs $c_{i,k}$, and not using the $w_{i,j,k}$, because of the problem specification. Now for each safe subset $\mathcal{T}$, we try all possible orderings (there are $k!$ of them if $|\mathcal{T}| = k$); for each ordering, we compute the expected value of the dynamic energy consumption as follows: if, say, $\mathcal{T} = \{m_1, m_3, m_4\}$ and the ordering is m_3, m_4, m_1, then we compute

$$P_{3,d}w_{i,j,3} + (1 - e^{-\lambda_3 w_{i,j,3}})P_{4,d}w_{i,j,4} + (1 - e^{-\lambda_3 w_{i,j,3}})(1 - e^{-\lambda_4 w_{i,j,4}})P_{1,d}w_{i,j,1}.$$

We see that we optimistically assume no overlap between the three replicas, and compute the dynamic energy cost as the energy of the first replica (always spent) plus the energy of the second replica (paid only if the first replica has failed) plus the energy of the third replica (paid only if both the first and second replicas have failed), and so on. Note that here we use execution times and failure probabilities based upon the actual execution times $w_{i,j,k}$ and not upon the WCETs $c_{i,k}$. The value of the sum depends upon the ordering of the processors in $\mathcal{T}$. Hence, we check the 6 orderings and retain the minimal value. We do this for all safe subsets and retain the minimal value. Finally we sum the results obtained for each task instance and get the lower-bound for the original processor subset $\mathcal{S}$. We stress that this bound is not necessarily tight, because our computation assumes no overlap for any replica pair, and does not check the utilization of each processor (which may exceed 1). The final lower-bound $LB(\{w_{i,j,k}\})$ is the minimum over all processor subsets. Although the computation has exponential cost, due to the exploration of all processor subsets $\mathcal{S}$, the computation of the expected energy for a given ordering in a subset $\mathcal{T}$ of $\mathcal{S}$ obeys a closed-form formula.

6 Performance evaluation

This section assesses the performance of our different strategies to map and schedule real-time tasks onto heterogeneous platforms. In Section 6.1, we describe the parameters and settings used during the experimental campaign. We present the results in Section 6.2. The algorithms are implemented in C++ and in R. The related computing code and experimental data are publicly available in [5]. A companion research report with the comprehensive set of results is available [6].

6.1 Experimental methodology

In the experiments, we have $M = 10$ processors and $N = 20$ tasks. The hyperperiod L of the system is fixed at 300. The task instances $\tau_{i,j}$ arrive every

p_i, chosen between one of the following values: 20, 30, 50, 60, 100, 150. The set of WCETs is generated by the method proposed in [1], as mentioned in Section 2.1. The WCET values are controlled by the correlation factor between the different tasks (cor_{task}) and between the different processors (cor_{proc}). For example, $cor_{\text{task}} = 0$ (resp. $cor_{\text{proc}} = 0$) means that the WCET values between different tasks on one processor (resp. between different processors for one task) are completely randomly generated. Inversely, $cor_{\text{task}} = 1$ (resp. $cor_{\text{proc}} = 1$) means that the WCET values between different tasks on one processor (resp. between different processors for one task) are all the same. We vary these two parameters between 0.25 and 0.75 to visualize the result under different correlation conditions, while guaranteeing a certain degree of randomness. We also define a parameter *basicWork* as the estimated total utilization of the system with a single replica per task instance, in order to study the impact of system workload pressure:

$$basicWork = \frac{\sum_{i,k} u_{i,k}}{M^2} \tag{10}$$

In Equation (10), we use the average utilization on the M processors ($\frac{\sum_k u_{i,k}}{M}$) to estimate the pressure that one replica of task τ_i can give on the system. We sum up the average utilization of all N tasks, and we divide this value by M because M processors are available. Hence, *basicWork* represents an estimation of the fraction of time that processors are used if each task has a single replica. In the experiments, we vary *basicWork* from 0.1 to 0.3.

To generate the actual execution times for each task instance from the task WCETs, we use two parameters. The first one, $\beta_{\text{b/w}}$, is global to all tasks: $\beta_{\text{b/w}}$ is the ratio between the best-case execution time and the worst-case execution time. It is the smallest possible ratio between the actual execution time of a task instance and the WCET of that task. Therefore, the actual execution time of all instances of task τ_i on processor m_k belongs to $[\beta_{\text{b/w}} c_{i,k}, c_{i,k}]$. We consider five possible values of $\beta_{\text{b/w}}$: 0.2, 0.4, 0.6, 0.8, and 1. The second parameter, $\beta_{i,j}$, is task instance dependent: $\beta_{i,j}$ describes whether the task instance $\tau_{i,j}$ is a small one or a large one. $\beta_{i,j}$ is randomly drawn in $[0, 1]$. $\beta_{i,j} = 0$ means that task instance $\tau_{i,j}$ has the shortest possible execution time, and $\beta_{i,j} = 1$ means that the actual execution is equal to its worst case execution time. Overall, the actual execution time of task instance $\tau_{i,j}$ on processor m_k is thus defined as: $w_{i,j,k} = (\beta_{\text{b/w}} + (1 - \beta_{\text{b/w}})\beta_{i,j})c_{i,k}$.

For a processor m_k in the platform, we fix its static power $P_{k,s}$ at 0.001 as in the literature [29, 31, 33]. For the dynamic power and the failure rate, we have two sets of parameters. The first parameter set also follows values from previous work [29, 31, 33]. For this set, we have a relatively high dynamic power and very low failure rate. Therefore, the replicas using this first set of parameters succeed in almost all cases. Then, to evaluate our heuristics in the context when failures occur more frequently, we introduce a second set of parameters where the replicas have lower dynamic power and relatively high

failure rate. For the first set, we choose randomly the dynamic power $P_{k,d}$ between 0.8 and 1.2, and the failure rate λ_k between 0.0001 and 0.00023. And for the second set, we have $P_{k,d}$ 10 times smaller (between 0.08 and 0.12), and λ_k 100 times larger (between 0.01 and 0.023). With the second set of parameters, the actual reliability of one replica ranges from 0.1 to 0.99. To be more realistic, in our experiments processors with a larger dynamic power $P_{k,d}$ have a smaller failure rate λ_k. It means that a more reliable processor costs more energy than a less reliable one. We guarantee this by ordering inversely the $P_{k,d}$'s and the λ_k's after generating the random values.

Due to space limitations, and because the general trends about the relative performance of heuristics is the same in both sets, we only include here figures about the large failure rate set; figures for the small failure rate set can be found in the companion research report [6].

We vary the local reliability target $\mathcal{R}_i$ between 0.8 and 0.95 for the big failure rate set and between 0.9 and 0.98 for the small failure rate set. This is to give the system a reasonable freedom while mapping and scheduling. The reliability target is relatively high, implying that tasks need several replicas to reach it. But it is chosen low enough so that feasible mappings can be found in the vast majority of scenarios.

6.2 Results

In this section, we compare the performance of the different criteria presented in Sections 3 and 4, and we choose the criterion which performs the best. Next, we analyze the impact of the different parameters on the performance of the chosen criterion.

Due to lack of space, we choose only to present a representative subset of the results. The comprehensive set of all results is available in the extended version [6]. We choose as default values $\beta_{\mathrm{b/w}} = 1$, $basicWork = 0.3$, $\mathcal{R}_i = 0.95$. This set of parameters is chosen to observe results when the platform is under maximum pressure. We fix cor_{task} and cor_{proc} both at 0.5 as default values.

Each experiment is an average of 25 WCETs sets generated as follows. We first generate 5 sets of periods. For each of these sets of periods, 5 WCET matrices are generated. For each WCET matrix, we generate 10 sets of random $P_{k,d}$ and λ_k values. For each $P_{k,d}$ and λ_k generated, the final result is the average over 10 executions. Overall, we run $2,500$ randomly generated experiments for each set of $\beta_{\mathrm{b/w}}$, $basicWork$, $\mathcal{R}_i$, cor_{task} and cor_{proc} values. The total number of experiments ran is 2,700,000 for each heuristic.

Each result point is represented as the percentage of energy saved over the random baseline method (defined below). Hence, on all the figures and in all the tables, the higher, the better. The random baseline method is defined as follows: for each task, we add replicas randomly on available processors until reaching the task reliability target during the mapping phase; for scheduling, we randomly order replicas mapped on each processor and execute them in sequence and as soon as possible. We compare our different strategies to the lower-bound proposed in Section 5. During the scheduling phase, we

	random	*deW*	*inW*	*deMinW*	*deMaxW*	*inMinW*	*inMaxW*
$cor_{\text{task}} = 0.25, cor_{\text{proc}} = 0.25$	54.8%	54.7%	55.1%	55.0%	54.7%	54.8%	55.1%
$cor_{\text{task}} = 0.25, cor_{\text{proc}} = 0.50$	47.6%	47.5%	48.0%	47.6%	47.5%	48.0%	48.0%
$cor_{\text{task}} = 0.25, cor_{\text{proc}} = 0.75$	33.5%	33.8%	33.7%	33.8%	33.8%	33.7%	33.7%
$cor_{\text{task}} = 0.50, cor_{\text{proc}} = 0.25$	55.1%	55.4%	55.1%	55.4%	55.4%	55.1%	55.1%
$cor_{\text{task}} = 0.50, cor_{\text{proc}} = 0.50$	48.6%	48.7%	48.8%	48.7%	48.7%	48.8%	48.8%
$cor_{\text{task}} = 0.50, cor_{\text{proc}} = 0.75$	33.4%	33.6%	33.6%	33.6%	33.6%	33.6%	33.6%
$cor_{\text{task}} = 0.75, cor_{\text{proc}} = 0.25$	54.6%	54.9%	54.4%	54.9%	54.8%	54.4%	54.4%
$cor_{\text{task}} = 0.75, cor_{\text{proc}} = 0.50$	44.2%	44.4%	44.3%	44.3%	44.4%	44.3%	44.3%
$cor_{\text{task}} = 0.75, cor_{\text{proc}} = 0.75$	32.9%	33.2%	33.2%	33.2%	33.1%	33.2%	33.2%

Table 3: Percentage of energy saved over the baseline when using different task ordering heuristics during the mapping phase under big failure rate, with $basicWork = 0.3$, $\beta_{\text{b/w}} = 1$, and $\mathcal{R}_i = 0.95$.

add several other references called EDF_Plain, EDF_Ref_Paper, and Smallest. They all use the same mapping as our heuristics, but have different scheduling strategies: EDF_Plain is simply the plain EDF heuristic. EDF_Ref_Paper is the heuristic proposed in [14]. Smallest considers a single replica for each task instance, the replica which costs the least energy (hence, Smallest is not always a valid heuristic as it may not satisfy the reliability threshold). In fact, Smallest shows an ideal case for the scheduling phase: every instance of every task successfully executes one replica, the replica which costs the least energy, while all other replicas are not yet started and are canceled after the first replica is completed. In contrast, LowerBound calculates the lowest expected energy consumption without considering the load of processors. Thus, we can find in the experimental results that Smallest can have a better result than LowerBound in some cases.

For the 2,500 random trials for each setting, in the figures, we report the average number of replicas needed in total for the 20 tasks (on the left side), the average number of failures that occur per time unit (in the middle), and the average percentage of random trials in which we find feasible mapping (on the right side). These numbers are reported in black above the horizontal axis on each figure.

6.2.1 Task ordering criteria for the mapping phase

In Table 3, we summarize the performance of different task ordering criteria during the mapping phase for different cor_{task} and cor_{proc} parameter sets. We see that, for each (cor_{task}, cor_{proc}) group, the difference between the different task ordering criteria, including random, is not obvious. The difference is at most 0.7% between the best and the worst criteria for any given set of (cor_{task}, cor_{proc}) values. We conclude that these criteria do not critically influence energy consumption. Therefore, in the following, for ordering tasks, we will only consider *deMinW*, which performs globally slightly better than others. We now focus on how to select processors during the mapping phase, and on the scheduling strategies.

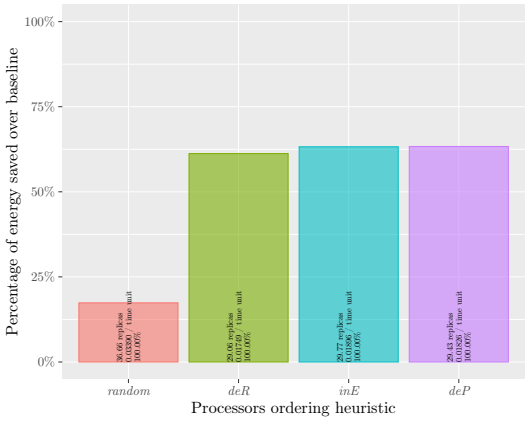


Figure 4: Percentage of energy saved over the baseline when using *deMinW* and different processor ordering heuristics during the mapping phase under big failure rate, with $cor_{\text{proc}} = 0.5$, $cor_{\text{task}} = 0.5$, $basicWork = 0.3$, $\beta_{\text{b/w}} = 1$, and $\mathcal{R}_i = 0.95$.

6.2.2 Processor ordering criteria for the mapping phase

Figure 4 shows the performance of different processor ordering criteria. We see from this figure that our criteria perform globally well. Our heuristics, random excepted, can save around 60% of energy compared to the baseline scheme. Among the different criteria, *inE* and *deP* have similar performance, and this performance is better than that of *deR*. In the following, we will consider *deP* as the default criterion for ordering processors.

6.2.3 Scheduling heuristics

For the scheduling strategies, Figure 5 shows that, after carefully selecting the mapping criterion, our heuristics can save up to 66.9% of the energy when compared to the random baseline. Among the different heuristics, we find that the EDF_Start_Time heuristic, which chooses the earliest replica as primary, and which has an energy saving performance of 64.2%, is the worst of our original heuristics. On the contrary, EDF_Energy performs the best (with a performance of 66.9%). It saves 8.4% more energy than EDF_Plain, and the performance is only 2.7% worse than Smallest. Recall that Smallest sums up, for each task instance, only one replica which costs the least energy, without considering failures or replica overlapping. It is very likely that this bound is unreachable, especially when the system is under high pressure. EDF_WCET performs similarly to EDF_Energy, and EDF_Reliability performs slightly worse. In fact, as the energy consumption of a replica depends a lot on its WCET, it is reasonable that EDF_WCET and EDF_Energy tend to choose the same primary replica and have similar performance. This result shows that it is better to choose

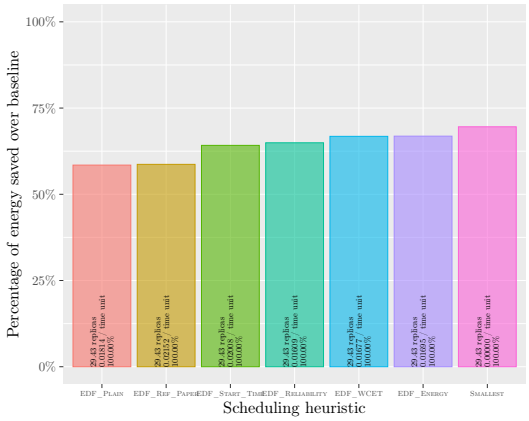


Figure 5: Percentage of energy saved over the baseline when using the *deMinW* and *deP* heuristics during the mapping phase and different criteria during the scheduling phase under big failure rate, with $cor_{\text{proc}} = 0.5$, $cor_{\text{task}} = 0.5$, $basicWork = 0.3$, $\beta_{\text{b/w}} = 1$, and $\mathcal{R}_i = 0.95$.

a replica which costs less energy as primary. In contrast, the earliest starting replica may not lead to minimal energy consumption, although we could have expected that it would induce less overlap among replicas.

In the following paragraphs, we only consider EDF_Energy and we focus on the variation of its performance for the different parameters. We add EDF_Plain, EDF_Start_Time, and Smallest as references.

6.2.4 Task and processor correlation

Figures 6 and 7 show results when task correlation or processor correlation varies. We see that, when cor_{task} and cor_{proc} are not high (< 0.75), the performance remains relatively stable. The EDF_Energy strategy achieves an energy saving percentage over the random baseline of around 65%. This result is close to the Smallest and LowerBound references. When cor_{task} and cor_{proc} is high, we can observe that the performance decreases. In the case of $cor_{\text{task}} = 0.75$ or $cor_{\text{proc}} = 0.75$, the performance of EDF_Energy decreases respectively to 57.3% and 45.4%, but the difference with LowerBound remains small (3.8% and 7.2% respectively).

The figures also show that the performance of EDF_Start_Time is closer to that of EDF_Energy when cor_{task} or cor_{proc} is high. The reason is that, with a higher cor_{task} or cor_{proc}, the WCETs of different tasks on the same processor, or the WCETs of the same task on different processors, are more similar. In both cases, the orders of the processors for different tasks become more similar, and are more closely related to the power and/or the reliability parameters ($P_{k,d}$ and λ_k). Therefore, after the mapping phase, we have a few fully used processors, while the other processors have a low load. The overlapping increases, and the performance becomes worse. Inversely, when cor_{task} or

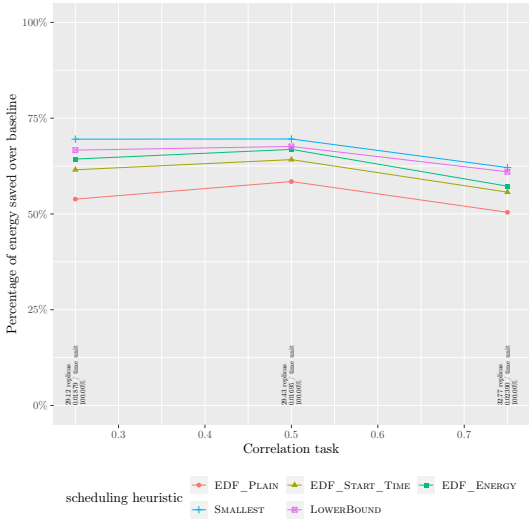


Figure 6: Percentage of energy saved over the baseline when using *deMinW* and *deP* during mapping for different scheduling criteria, under big failure rate, when varying cor_{task}, with $cor_{\text{proc}} = 0.5$, $basicWork = 0.3$, $\beta_{\text{b/w}} = 1$, and $\mathcal{R}_i = 0.95$.

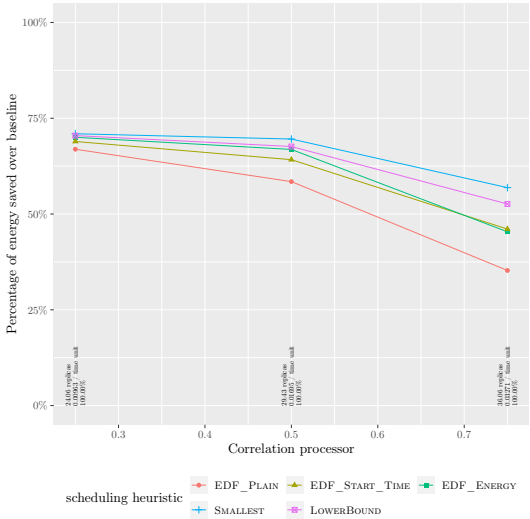


Figure 7: Percentage of energy saved over the baseline when using *deMinW* and *deP* during mapping for different scheduling criteria, under big failure rate, when varying cor_{proc}, with $cor_{\text{task}} = 0.5$, $basicWork = 0.3$, $\beta_{\text{b/w}} = 1$, and $\mathcal{R}_i = 0.95$.

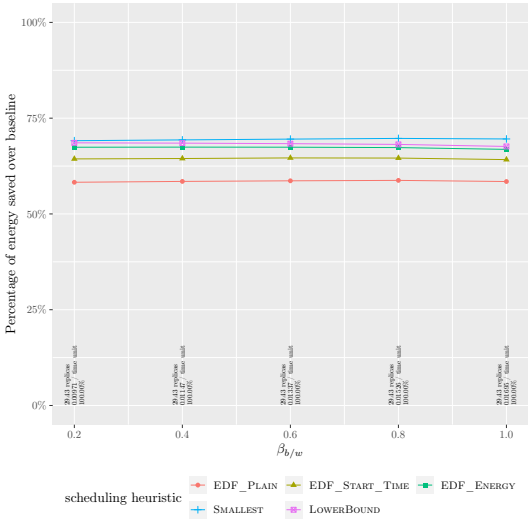


Figure 8: Percentage of energy saved over the baseline when using *deMinW* and *deP* during mapping for different scheduling criteria, under big failure rate, when varying $\beta_{\mathrm{b/w}}$, with $cor_{\mathrm{task}} = 0.5$, $cor_{\mathrm{proc}} = 0.5$, $basicWork = 0.3$, and $\mathcal{R}_i = 0.95$.

cor_{proc} is not high, if we simply choose the replica which costs the least energy as the primary replica, the primary replicas are more randomly distributed on the different processors because of the high randomness of WCETs, and the overlapping is minimal. Then it is possible that we finish the primary replica early and with success, and delete all secondary replicas before starting the processing of any of them. This is why, when cor_{task} or cor_{proc} is high, the EDF_Energy strategy cannot save as much energy as in other cases.

6.2.5 Task variability

Figure 8 presents the results when $\beta_{\mathrm{b/w}}$ varies. We observe that the result of each heuristic is (almost) independent of the value of $\beta_{\mathrm{b/w}}$. This is because we map and schedule tasks based on their WCETs, so the mapping and scheduling results are independent of the value of $\beta_{\mathrm{b/w}}$. Furthermore, each task τ_i has the same β_i on the different processors. Therefore the energy savings tend to be similar. More precisely, the performance of EDF_Energy varies only for 0.5% when $\beta_{\mathrm{b/w}}$ increases from 0.2 to 1. So we can conclude that the result is independent of $\beta_{\mathrm{b/w}}$.

When $\beta_{\mathrm{b/w}}$ is small, actual execution times can greatly differ from the WCETs which are used for mapping and scheduling. However, in this case, the heuristics perform as well as in the case $\beta_{\mathrm{b/w}} = 1$. This shows that the heuristics are very robust.

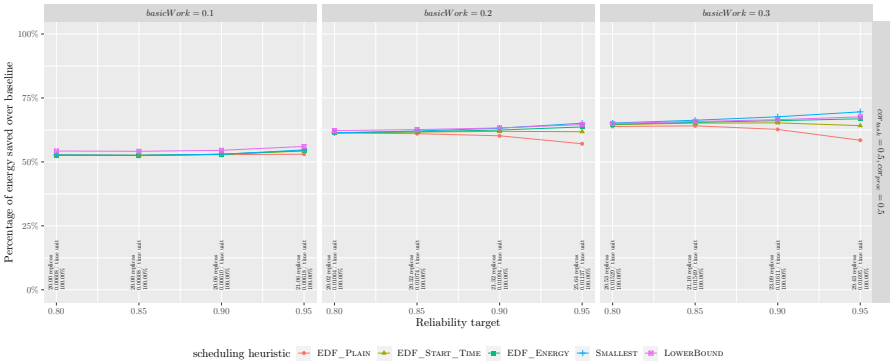


Figure 9: Percentage of energy saved over the baseline when using *deMinW* and *deP* during mapping for different scheduling criteria, under big failure rate, when varying *basicWork* and $\mathcal{R}_i$, with $cor_{\text{task}} = 0.5$, $cor_{\text{proc}} = 0.5$ and $\beta_{\text{b/w}} = 1$.

6.2.6 Utilization and reliability threshold

On Figure 9, we observe the performance of the different heuristics when varying both *basicWork* and $\mathcal{R}_i$. We see that, when *basicWork* and $\mathcal{R}_i$ increase, the differences between heuristics become larger. EDF_Plain performs worse, while the performance of EDF_Energy remains stable. The difference between the Smallest and LowerBound references and our heuristics also increases with *basicWork* and $\mathcal{R}_i$. When *basicWork* $= 0.3$ and $\mathcal{R}_i = 0.95$, this difference of percentages is only of 1.5% for Smallest and 0.4% for LowerBound. In fact, with the increase of the system load, the WCETs increase, and each replica has a higher chance to be victim of a failure. But LowerBound does not consider the load of processors, and Smallest estimates the energy consumption without considering failures or overlapping. This explains the growing gap between these references and the heuristics when *basicWork* and $\mathcal{R}_i$ increase.

In summary, our strategy maintains a stable performance when *basicWork* and $\mathcal{R}_i$ increase. In contrast, EDF_Plain performs worse. The difference between EDF_Energy and the LowerBound reference is lower than 1% even in the worst case, which stresses the high performance of the designed strategy.

6.2.7 Number of replicas, number of failures and success rate

We counted the number of replicas that failed during the execution in each experiment. In the set with small failure rate, we have on average 0.0002 failed replicas per time unit. On the contrary, in the set with big failure rate, the average number of failed replicas per time unit increases to 0.0129.

In Figure 4, we observe that *random* needs more replicas and is the victim of a larger number of failures. *deR* achieves the lowest total number of replicas

	$cor_{\text{task}} = 0.25$	$cor_{\text{task}} = 0.50$	$cor_{\text{task}} = 0.75$
$cor_{\text{proc}} = 0.25$	100%	100%	100%
$cor_{\text{proc}} = 0.50$	99.971%	100%	100%
$cor_{\text{proc}} = 0.75$	99.111%	99.992%	100%

Table 4: Success rate of random trials under big failure rate, when varying cor_{task} and cor_{proc}.

and total number of failures encountered because it always uses the processors with highest reliability. *deP* achieves a slightly lower number of replicas and failures than *inE* because it considers both reliability and energy.

In Figure 5, the total number of replicas is the same for all criteria because they all use the same mapping. For the number of failures, the difference is not obvious between strategies. Unsurprisingly, EDF_Reliability has the lowest failure rate. Then, EDF_WCET and EDF_Energy have slightly better failure rates than the other heuristics.

Figures 6 and 7 show that the number of replicas and failures remain relatively stable when cor_{task} or cor_{proc} is not high. But there is a relative increase when cor_{task} or cor_{proc} reaches 0.75.

As for $\beta_{\text{b/w}}$, we see on Figure 8 that, when $\beta_{\text{b/w}}$ increases, the number of replicas remains the same, while the number of failures increases. This is because we map replicas according to the task WCET which is independent on the value of $\beta_{\text{b/w}}$. However, when $\beta_{\text{b/w}}$ is larger, we have longer actual execution times and, thus, a higher probability that failures actually happen.

While increasing $\mathcal{R}_i$ and *basicWork*, we observe in Figure 9 that the number of replicas increases, and also the number of failures. With higher $\mathcal{R}_i$, we need more replicas to satisfy the reliability target, and more replicas executed at the same time means a higher probability that failures actually occur. As for *basicWork*, a higher *basicWork* means larger average WCETs. Longer replicas lead to lower reliability. Thus failures happen more frequently, and we need also more replicas to reach the reliability threshold.

The success rate can be found in Tables 4 and 5. We observe that the proportion of experiments for which we can find a feasible mapping decreases when the system pressure becomes large. The fraction of instances for which we can build a solution also decreases when cor_{task} is low and cor_{proc} is high. This is why we used $\mathcal{R}_i = 0.95$ and *basicWork* $= 0.3$ as the highest reliability target and basic work parameter, and we avoided to use extreme correlation values in our experiments. Within all platforms in the experiments, all the tested heuristics are able to find a valid solution in all tested configurations with small failure rate. And in the big failure rate cases, heuristics are able to build valid solutions for more than 99.8% of the instances. The very high success rate of our experiments means that we can be confident in our results and that they do not suffer of any bias.

	$basicWork = 0.1$	$basicWork = 0.2$	$basicWork = 0.3$
$\mathcal{R} = 0.80$	100%	100%	100%
$\mathcal{R} = 0.85$	100%	100%	100%
$\mathcal{R} = 0.90$	100%	100%	99.940%
$\mathcal{R} = 0.95$	100%	100%	98.825%

Table 5: Success rate of random trials under big failure rate, when varying cor_{task} and cor_{proc}.

6.2.8 Additional scenarios

To evaluate our methods under various real-world scenarios, we set up three additional experiments and observed the changes in energy savings of our heuristics. The different settings of these simulations are as follows:

- *none*: Default setting as described in Section 6.1.
- *depend*: In this setting, we randomly generate multiple time intervals for each processor during the whole execution, and within these intervals, failures occur randomly following an exponential distribution. In this setting, tasks have a higher failure probability during certain time intervals, rather than solely being related to their execution times.
- r: In this setting, we attempt to simulate the scenario where acceptance testing is not perfect. Referring to the settings in [14], we simulate acceptance testing success rates of 95% and 99%, denoted as $r95$ and $r99$, respectively.
- *exp*: In this setting, we implement a scenario where the actual execution time of tasks is not uniformly distributed but rather generated according to an exponential distribution within the range $[\beta_{\text{b/w}} c_{i,k}, c_{i,k}]$. We tested two cases in which exponential distribution parameter lambda equal to 1 and 2, denoted as $exp1$ and $exp2$, respectively.

Due to space limitations, we are unable to showcase the complete experimental results for these additional scenarios. However, full version of the figures and tables can be found in [5].

In Figure 10, we can observe that in *depend* and *none* settings, our best heuristic achieves very similar energy savings, with a difference of less than 1%. Simultaneously, we could observe from the bar charts that the number of failures occurring in *depend* setting is more than twice that of *none* setting, which is consistent with the increased failure probability in *depend* setting.

Figure 11 demonstrates the energy savings (see different lines) and failure occurrences of our best heuristic in $exp2$, $exp1$, and *none* settings, under different $\beta_{\text{b/w}}$ conditions. It can be observed that the percentage of energy saved is the same across different settings. However, the occurrence of failures, shown with bar charts, is less in $exp2$, followed by $exp1$, and the most in *none*. This is because in $exp2$, compared to $exp1$ or *none*, there is a higher probability that the actual execution time of tasks will be shorter, resulting in a reduced occurrence of failure.

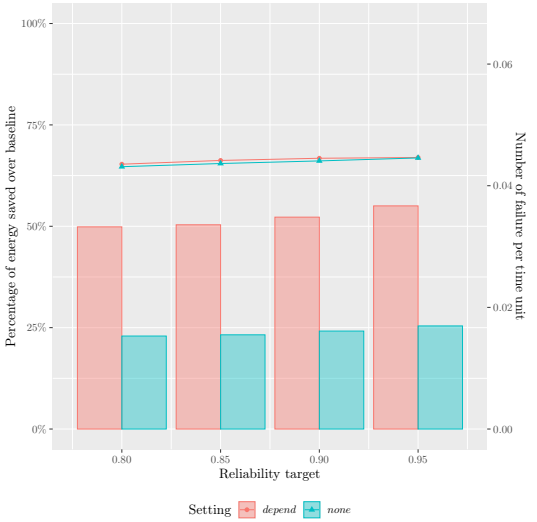


Figure 10: Percentage of energy saved over the baseline of *none* and *depend* settings, when using *deMinW* and *deP* during mapping, and using the EDF_ENERGY scheduling criterion, under big failure rate, when varying $\beta_{\mathrm{b/w}}$, with $cor_{\mathrm{task}} = 0.5$, $cor_{\mathrm{proc}} = 0.5$, $basicWork = 0.3$, and $\mathcal{R}_i = 0.95$.

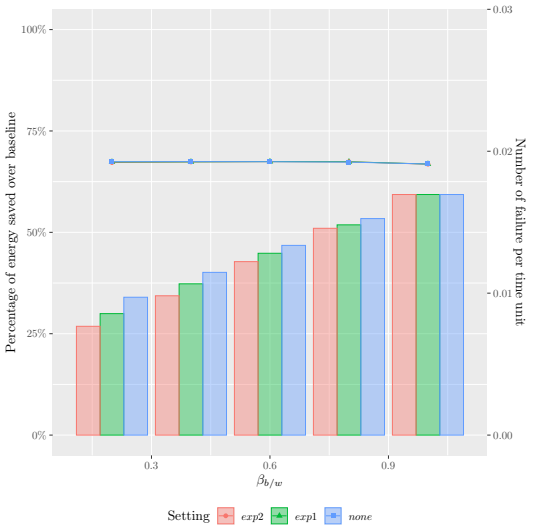


Figure 11: Percentage of energy saved over the baseline of *none*, *exp*1 and *exp*2 settings, when using *deMinW* and *deP* during mapping, and using the EDF_ENERGY scheduling criterion, under big failure rate, when varying $\mathcal{R}_i$, with $cor_{\mathrm{task}} = 0.5$, $cor_{\mathrm{proc}} = 0.5$, $basicWork = 0.3$ and $\beta_{\mathrm{b/w}} = 1$.

	*r*95	*r*99	*none*
EDF_Plain	52.1%	57.4%	58.5%
EDF_Ref_Paper	52.4%	57.4%	58.7%
EDF_Start_Time	61.0%	63.8%	64.2%
EDF_Reliability	64.0%	64.8%	64.9%
EDF_WCET	66.5%	66.9%	66.8%
EDF_Energy	66.5%	67.0%	66.9%
Smallest	71.7%	70.1%	69.6%

Table 6: Percentage of energy saved over the baseline of *none*, *r*99 and *r*95 settings, when using *deMinW* and *deP* during mapping, and using different scheduling criteria under big failure rate, with $cor_{\text{task}} = 0.5$, $cor_{\text{proc}} = 0.5$, $\beta_{\text{b/w}} = 1$, $basicWork = 0.3$, and $\mathcal{R}_i = 0.95$.

	random	*deR*	*inE*	*deP*
EDF_Plain	6.9%	57.2%	58.2%	58.5%
EDF_Ref_Paper	8.8%	57.3%	58.1%	58.7%
EDF_Start_Time	17.0%	62.1%	64.0%	64.2%
EDF_Reliability	23.6%	62.1%	65.3%	64.9%
EDF_WCET	23.8%	64.2%	66.8%	66.8%
EDF_Energy	24.2%	64.4%	66.8%	66.9%
LowerBound	67.6%			
Smallest	51.8%	67.4%	69.7%	69.6%

Table 7: Percentage of energy saved over the baseline when using different processor ordering criteria during mapping and using different scheduling criteria under big failure rate, with $cor_{\text{task}} = 0.5$, $cor_{\text{proc}} = 0.5$, $\beta_{\text{b/w}} = 1$, $basicWork = 0.3$, and $\mathcal{R}_i = 0.95$.

Table 6 lists the results of all scheduling heuristics in *r* and *none* settings. We can observe that, as the success probability of acceptance testing decreases, the degradation of EDF_Plain and EDF_Ref_Paper is most significant, reaching approximately 5%, while the energy saving ratio of our best heuristic remains stable with a decrease of less than 1%.

Through these experiments, we have verified that our best heuristic can typically maintain an excellent energy saving ratio in different settings, validating the versatility of our heuristic.

6.2.9 Summary

In conclusion, in the experiments we compared different criteria for the mapping and scheduling phases, and we also studied the influence of the different parameters. Tables 7 and 8 present a synthetic view of the results achieved by different heuristics, or for different correlations, when the system is under the highest pressure ($\beta_{\text{b/w}} = 1$, $basicWork = 0.3$, and $\mathcal{R}_i = 0.95$).

	$cor_{\text{task}} = 0.25$	$cor_{\text{task}} = 0.5$	$cor_{\text{task}} = 0.75$
$cor_{\text{proc}} = 0.25$	72.4%	70.0%	67.1%
$cor_{\text{proc}} = 0.50$	64.3%	66.9%	57.3%
$cor_{\text{proc}} = 0.75$	42.6%	45.4%	46.4%

Table 8: Percentage of energy saved over the baseline when using the *deMinW* task ordering criterion and the *deP* processor ordering criterion during mapping, and using the EDF_ENERGY scheduling criterion, under big failure rate, when varying cor_{task} and cor_{proc}, with $\beta_{\text{b/w}} = 1$, $basicWork = 0.3$, and $\mathcal{R}_i = 0.95$.

Among the different criteria used in the heuristics, we observe from Table 7 that the *deP* method is the best processor ordering for the mapping phase, and EDF_ENERGY performs best among the scheduling criteria. In the case of high system pressure, our best criterion can still achieve 66.9% of energy saving compared to the random baseline. The difference of performance is 42.7% if we use *random* as the processor ordering criterion during the mapping phase, and the difference is only 2.7% compared to the SMALLEST reference during the scheduling phase. To study the efficiency of our selected heuristic under different cor_{task} and cor_{proc}, Table 8 presents the performance while using *deMinW*, *deP* and EDF_ENERGY as criteria. Our strategies can save more than 40% of the energy consumed by the baseline in all cases, and this percentage can be as high as 72.4% in the best case. Furthermore, we report a median energy saving percentage of only 1.6% less than that of the LOWERBOUND; and an average of only 2.0% less.

We can confidently conclude that our best strategies perform remarkably well over the whole experimental settings.

7 Related work

There is a huge literature on multi-criteria task scheduling problem. In this section, we only reference some very recent work closely related to our problem.

7.1 Non-periodic task scheduling

We start from introducing works dealing with non-periodic tasks. [29] maximizes the reliability of an energy-constrained DAG executed on a heterogeneous platform while using DVFS. Conversely, [33] minimizes the energy consumption of a reliability-constrained DAG executed on a heterogeneous platform while using, or not, DVFS. On the other hand, [20] deals simultaneously with makespan, execution cost and energy consumption, while considering DAG and using DVFS. [17] designs energy-aware scheduling strategies for independent tasks based on user's customized QoS preferences, such as cost, makespan, utilization of processors, etc. A group of authors published a book [32] and several articles on the problem of DAG scheduling on heterogeneous platforms. In Chapter 2 of book [32] and in [30] these authors consider

the energy minimization when scheduling a DAG with or without DVFS. However, these two references do not consider reliability. In [31] they considered the same problem while satisfying some reliability goal.

Overall, these studies do not consider real-time applications. The deadlines which constrain real-time tasks render problems significantly harder to tackle.

7.2 Real-time task scheduling on homogeneous platforms

Liu and Layland first introduced the Earliest Deadline First (EDF) and the Rate Monotonic (RM) scheduling policies for real-time systems and provided the utilization bounds for both policies in 1973 [18]. Since then, the real-time scheduling problem has been extensively studied.

There exists a very significant literature on real-time scheduling for multiprocessor systems. However, most work is devoted to homogeneous processor systems, as exemplified by the survey [3] which ignores altogether heterogeneous systems, and by the more recent survey [25] where only 9 of the 78 references deal with heterogeneous platforms. [14] minimizes the energy when scheduling independent tasks with different deadlines on a homogeneous platform while satisfying some threshold on reliability. The study [11] improved the solution from [14], in particular by carefully avoiding overlaps between primary and secondary replicas. [13] considers the same problem; however, it uses checkpointing to cope with failures when all other works consider replication. We refer the interested reader to [3, 11, 14, 25] for a comprehensive overview of the related work for homogeneous platforms.

7.3 Real-time task scheduling on heterogeneous platforms

Heterogeneous platforms make the problem even harder because processors can have different speeds, energy costs, and failure rates. Therefore, the processor preferred for one task by one of the objectives and constraints —deadline satisfaction, energy minimization, reliability threshold satisfaction— may be the worst processor for another objective or constraint. The heuristics have thus to perform complicated trade-offs in these three-criteria settings. Following paragraphs introduce a set of works about real-time task scheduling problem on heterogeneous platform, while dealing with different targets and task models.

Some related works target the scheduling of real-time applications on heterogeneous platforms, but without considering fault tolerance. For instance, [36] targets the execution of a DAG, but considering neither energy consumption nor fault-tolerance (when DAGs are scheduled, tasks are always assumed to have the same deadline). [10] targets the execution of independent tasks that access shared resources, the access to resources being exclusive. Their objective is to maximize the number of instances for which a solution is found. [19], [24] and [35] minimize energy consumption by using DVFS, [19] when scheduling independent tasks, [24] a DAG, and [35] a moldable application. [28] considers the scheduling of independent tasks and DAGs under an energy constraint,

while [27] considers the scheduling of independent tasks under a thermal constraint. [34] proposes a fully polynomial-time approximation scheme (FPTAS) for minimizing the energy consumption for a set of independent tasks executed on a set of heterogeneous (unrelated) processing elements.

Some related work considers the execution of real-time applications on heterogeneous failure-prone platforms but is limited to coping with a single failure per task or per processor. [22] maximizes the reliability of the considered DAG but does not consider energy consumption and follows the primary/backup technique and, thus, is limited to at most one failure per task of the DAG. [23] attempts to maximize resource utilization (and does not consider energy) when scheduling a set of independent tasks. It assumes that at most one processor can fail, which enables the simultaneous scheduling of several backup tasks on the very same processor, since at most one of them will need to be executed. [16] minimizes the energy consumed for the execution of a DAG while satisfying a reliability threshold. The proposed solution uses DVFS and Power Mode Management (i.e., the ability to switch off idle processors to low-power inactive state). This solution, however, cannot produce a schedule more reliable than the original one. It also supports at most one fault per processor. [9] minimizes the energy consumed for the execution of a set of independent tasks while satisfying a reliability threshold using DVFS and following a primary-backup approach.

Very few studies consider the execution of real-time applications on heterogeneous failure-prone platforms and can cope with two or more failures per task. [26] minimizes the energy consumed for the execution of a set of independent tasks while satisfying a reliability threshold. The proposed solution uses DVFS. This solution, however, is based on a primary-backup approach that is then extended. This approach, by design, cannot produce a schedule more reliable than the original one with two replicas per task, strongly relies on DVFS, and schedules several replicas of a same task on the same processor (what most other approaches forbid). [8] targets the execution of a DAG on a heterogeneous platform while satisfying a reliability threshold. However, the objective is not the minimization of energy consumption but the maximization of the *utilization of energy consumption*, which can be seen as a yield of reliability improvement with respect to increased energy consumption. As a consequence, [8] produces energy greedy schedules (see subplots (a-1), (b-1), and (c-1) of Figure 1 in [8]). There exists a few works which consider the same problem as our work, but for a DAG instead of for a set of independent tasks. [15] proposes a DAG scheduling algorithm for energy minimization under deadline and reliability constraints. But they do not use replication method to improve the reliability. They only use DVFS and evaluate if the reliability target can be reached with different frequency levels. In Chapter 3 of the already mentioned book [32], the authors also consider cost minimization (which can be energy minimization) when scheduling a DAG under deadline and reliability constraints. Because of the dependence between tasks and the chosen as-soon-as-possible scheduling of [32], this solution tends to schedule

simultaneously the different replicas of a single task. As already pointed out in the studies [11, 14] this can lead to a significant waste of energy. Therefore, it would have been unfair to compare our solution to that of [32] and [15] applied to independent tasks.

From what precedes, we have identified only a single existing solution that enables to schedule real-time tasks on heterogeneous platforms while minimizing energy consumption and satisfying some bound on the overall reliability. However, this solution being dedicated to DAGs lacks the possibility to minimize overlapping between replicas of a same task, which has been previously shown to be crucial [11] and which we have given special care to in this paper (see Section 4).

8 Conclusion

In this work, we have studied the problem of executing real-time tasks with different deadlines on an heterogeneous platform, with several objectives: minimizing the expected energy consumption, ensuring certain reliability thresholds, and meeting all deadlines. Our approaches decide for each task: how many replicas should be launched, on which processor to map them, and the scheduling of each replica on its assigned processor. We designed mapping heuristics to assign arbitrary number of replicas onto nonidentical processors, aiming at meeting the given reliability target and deadlines, while minimizing the expected energy cost. In the scheduling phase, we tagged one replica per task as "primary" replica and the other ones as "secondary" replicas. We aimed at reducing overlap between primary replica and secondary replicas of each task instance to further minimize the energy consumption. To obtain an absolute measure for the evaluation of our heuristics, we have computed a theoretical lower-bound on the expected energy consumption.

Extensive simulations were conducted for a wide range of processing speed profiles, failure rates and execution scenarios. The results show that, overall, our strategies perform better than the baseline in nearly all cases. Moreover, our best heuristic achieves an excellent performance which is very close to the LowerBound (on average the percentage of energy saving of our best heuristic is only 2% less than that of LowerBound). This performance was reached by considering processors in the *deP* order when mapping the replicas of a task (roughly speaking, *deP* is the ratio of a task failure rate by its energy cost), and by tagging as "primary" the replicas which cost the smallest dynamic energy. Finally, we point out that while all decisions were taken with worst-case execution times (WCETs) of tasks as input, the simulations used actual execution times; the performance of the best heuristic was not influenced even when the actual execution times were far smaller than the WCETs, showing the robustness of our approach.

From a practical perspective, the good news is that our strategies perform very closely to the lower bound on a comprehensive set of experimental

scenarios. Still, for future work on the theory side, it would be nice to derive approximation algorithms, even though their performance ratio is likely to involve large factors such as the performance ratio of the fastest machine over that of the slowest machine. Future work will also aim at extending the algorithms to periodic graphs of tasks instead of independent task sets. The dependences between tasks will dramatically complicate the mapping and scheduling problems. Dealing with the combination of fail-stop errors (permanent failures, such as processor crashes) and transient faults is another challenging problem, and would require a brand new approach combing checkpoints and replication.

Acknowledgments

We thank the reviewers for their comments and suggestions. A shorter and preliminary version of this work appears in the proceedings of ICPP'2020 [12]. This paper is partially supported by the National Key Research and Development (2019YFA0706404), in part by the National Nature Science Foundation of China (62202168, 61972150), Shanghai Sailing Program 21YF1411100, and a JoRISS grant from ENS de Lyon and ECNU. The work of Yiqin Gao was supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

References

[1] Canon LC, El Sayah M, Héam PC (2018) A Markov chain Monte Carlo approach to cost matrix generation for scheduling performance evaluation. In: 2018 International Conference on High Performance Computing & Simulation (HPCS), IEEE, pp 460–467

[2] Chen H, Wen J, Pedrycz W, et al (2020) Big data processing workflows oriented real-time scheduling algorithm using task-duplication in geo-distributed clouds. IEEE Trans Big Data 6(1):131–144

[3] Davis RI, Burns A (2011) A survey of hard real-time scheduling for multiprocessor systems. ACM Comput Surv 43(4):35:1–35:44. https://doi.org/10.1145/1978802.1978814, URL http://doi.acm.org/10.1145/1978802.1978814

[4] Deng Z, Cao D, Shen H, et al (2021) Reliability-aware task scheduling for energy efficiency on heterogeneous multiprocessor systems. The Journal of Supercomputing 77:11,643–11,681

[5] Gao Y (2021) Heterogeneous real-time systems. https://figshare.com/articles/software/Minimizing_energy_consumption_for_real-time_tasks_on_heterogeneous_platforms_under_deadline_and_reliability_constraints/14450538

[6] Gao Y, Han L, Liu J, et al (2021) Minimizing energy consumption for real-time tasks on heterogeneous platforms under deadline and reliability constraints. Research report 9403, INRIA

[7] Garey MR, Johnson DS (1979) Computers and Intractability, a Guide to the Theory of NP-Completeness. W.H. Freeman and Company

[8] Guo T, Liu J, Hu W, et al (2018) Energy-aware fault-tolerant scheduling under reliability and time constraints in heterogeneous systems. In: Huang DS, Gromiha MM, Han K, et al (eds) Intelligent Computing Methodologies. Springer, pp 36–46

[9] Guo Y, Zhu D, Aydin H, et al (2017) Exploiting primary/backup mechanism for energy efficiency in dependable real-time systems. Journal of Systems Architecture 78:68 – 80. https://doi.org/https://doi.org/10.1016/j.sysarc.2017.06.008, URL http://www.sciencedirect.com/science/article/pii/S1383762116302624

[10] Han JJ, Cai W, Zhu D (2018) Resource-aware partitioned scheduling for heterogeneous multicore real-time systems. In: Proceedings of the 55th Annual Design Automation Conference. ACM, New York, NY, USA, DAC '18, pp 124:1–124:6, https://doi.org/10.1145/3195970.3196103, URL http://doi.acm.org/10.1145/3195970.3196103

[11] Han L, Canon LC, Liu J, et al (2020) Improved energy-aware strategies for periodic real-time tasks under reliability constraints. In: 40th IEEE Real-Time Systems Symposium (RTSS)

[12] Han L, Gao Y, Liu J, et al (2020) Energy-aware strategies for reliability-oriented real-time task allocation on heterogeneous platforms. In: ICPP'2020, the 49th Int. Conf. on Parallel Processing. ACM Press

[13] Han Q (2015) Energy-aware fault-tolerant scheduling for hard real-time systems. PhD thesis, Florida International University, https://doi.org/10.25148/etd.FIDC000077

[14] Haque MA, Aydin H, Zhu D (2017) On reliability management of energy-aware real-time systems through task replication. IEEE Transactions on Parallel and Distributed Systems 28(3):813–825

[15] Hu B, Cao Z, Zhou M (2021) Energy-minimized scheduling of real-time parallel workflows on heterogeneous distributed computing systems. IEEE Transactions on Services Computing pp 1–1. https://doi.org/10.1109/TSC.2021.3054754

[16] Huang K, Jiang X, Zhang X, et al (2018) Energy-efficient fault-tolerant mapping and scheduling on heterogeneous multiprocessor real-time systems. IEEE Access 6:57,614–57,630. https://doi.org/10.1109/ACCESS.2018.2873641

[17] Khorsand R, Ramezanpour M (2020) An energy-efficient task-scheduling algorithm based on a multi-criteria decision-making method in cloud computing. International Journal of Communication Systems 33(9):e4379. https://doi.org/https://doi.org/10.1002/dac.4379, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.4379, e4379 IJCS-19-0734.R2, https://arxiv.org/abs/https://onlinelibrary.wiley.com/doi/pdf/10.1002/dac.4379

[18] Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM) 20(1):46–61

[19] Moulik S, Chaudhary R, Das Z (2020) Hears: A heterogeneous energy-aware real-time scheduler. Microprocessors and Microsystems 72:102,939. https://doi.org/https://doi.org/10.1016/j.micpro.2019.102939, URL http://www.sciencedirect.com/science/article/pii/S0141933119302017

[20] Paknejad P, Khorsand R, Ramezanpour M (2021) Chaotic improved picea-g-based multi-objective optimization for workflow scheduling in cloud environment. Future Generation Computer Systems 117:12–28. https://doi.org/https://doi.org/10.1016/j.future.2020.11.002, URL https://www.sciencedirect.com/science/article/pii/S0167739X20330260

[21] Peng J, Li K, Chen J, et al (2022) Reliability/performance-aware scheduling for parallel applications with energy constraints on heterogeneous computing systems. IEEE Transactions on Sustainable Computing 7(3):681–695. https://doi.org/10.1109/TSUSC.2022.3146138

[22] Qin X, Jiang H (2006) A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems. Parallel Computing 32(5-6):331–356

[23] Qiu W, Zheng Z, Wang X, et al (2013) An efficient fault-tolerant scheduling algorithm for periodic real-time tasks in heterogeneous platforms. In: 16th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC 2013), pp 1–7, https://doi.org/10.1109/ISORC.2013.6913213

[24] Safari M, Khorsand R (2018) Energy-aware scheduling algorithm for time-constrained workflow tasks in DVFS-enabled cloud environment. Simulation Modelling Practice and Theory 87:311 – 326. https://doi.org/https://doi.org/10.1016/j.simpat.2018.07.006, URL http://www.sciencedirect.

com/science/article/pii/S1569190X18300984

[25] Sheikh SZ, Pasha MA (2018) Energy-efficient multicore scheduling for hard real-time systems: A survey. ACM Trans Embed Comput Syst 17(6):94:1–94:26. https://doi.org/10.1145/3291387, URL http://doi.acm.org/10.1145/3291387

[26] Sridharan R, Mahapatra R (2010) Reliability aware power management for dual-processor real-time embedded systems. In: Proceedings of the 47th Design Automation Conference. ACM, New York, NY, USA, DAC '10, pp 819–824, https://doi.org/10.1145/1837274.1837480, URL http://doi.acm.org/10.1145/1837274.1837480

[27] Tsai T, Chen Y, He X, et al (2018) STEM: a thermal-constrained real-time scheduling for 3D heterogeneous-ISA multicore processors. IEEE Transactions on Computers 67(6):874–889. https://doi.org/10.1109/TC.2017.2783941

[28] Valentin EB (2017) Scheduling hard real-time tasks in heterogeneous multiprocessor platforms subject to energy and temperature constraints. PhD thesis, Universidade Federal do Amazonas

[29] Xiao X, Xie G, Xu C, et al (2018) Maximizing reliability of energy constrained parallel applications on heterogeneous distributed systems. Journal of Computational Science 26:344 – 353. https://doi.org/https://doi.org/10.1016/j.jocs.2017.05.002, URL http://www.sciencedirect.com/science/article/pii/S1877750317304933

[30] Xie G, Zeng G, Xiao X, et al (2017) Energy-efficient scheduling algorithms for real-time parallel applications on heterogeneous distributed embedded systems. IEEE Transactions on Parallel and Distributed Systems 28(12):3426–3442. https://doi.org/10.1109/TPDS.2017.2730876

[31] Xie G, Chen Y, Xiao X, et al (2018) Energy-efficient fault-tolerant scheduling of reliable parallel applications on heterogeneous distributed embedded systems. IEEE Transactions on Sustainable Computing 3(3):167–181

[32] Xie G, Zeng G, Li R, et al (2019) Scheduling Parallel Applications on Heterogeneous Distributed Systems. Springer Singapore

[33] Xu H, Li R, Pan C, et al (2019) Minimizing energy consumption with reliability goal on heterogeneous embedded systems. Journal of Parallel and Distributed Computing 127:44–57

[34] Yang C, Chen J, Kuo T, et al (2009) An approximation scheme for energy-efficient scheduling of real-time tasks in heterogeneous multiprocessor systems. In: 2009 Design, Automation Test in Europe Conference Exhibition, pp 694–699, https://doi.org/10.1109/DATE.2009.5090754

[35] Zahaf HE, Benyamina AEH, Olejnik R, et al (2017) Energy-efficient scheduling for moldable real-time tasks on heterogeneous computing platforms. Journal of Systems Architecture 74:46 – 60. https://doi.org/https://doi.org/10.1016/j.sysarc.2017.01.002, URL http://www.sciencedirect.com/science/article/pii/S138376211730019X

[36] Zahaf HE, Capodieci N, Cavicchioli R, et al (2019) A C-DAG task model for scheduling complex real-time tasks on heterogeneous platforms: preemption matters, URL https://hal.archives-ouvertes.fr/hal-01971594, working paper or preprint

[37] Zhou J, Hu XS, Ma Y, et al (2019) Improving availability of multicore real-time systems suffering both permanent and transient faults. IEEE Transactions on Computers 68(12):1785–1801. https://doi.org/10.1109/TC.2019.2935042

[38] Zhu D, Melhem R, Mosse D (2004) The effects of energy management on reliability in real-time embedded systems. In: IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004., pp 35–40, https://doi.org/10.1109/ICCAD.2004.1382539

[39] Zhu X, Wang J, Guo H, et al (2016) Fault-tolerant scheduling for real-time scientific workflows with elastic resource provisioning in virtualized clouds. IEEE TransParallel and Distributed Systems 27(12):3501–3517