Energy-aware strategies for reliability-oriented real-time task allocation on heterogeneous platforms

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

Energy-aware strategies for reliability-oriented real-time task allocation on heterogeneous platforms

Li Han
East China Normal University
Shanghai, China
Univ Lyon, ENS Lyon, UCBL, CNRS,
Inria, LIP, Lyon, France
li.han@inria.fr

Yiqin Gao*
Univ Lyon, ENS Lyon, UCBL, CNRS,
Inria, LIP, Lyon, France
yiqin.gao@inria.fr

Jing Liu*
East China Normal University
Shanghai, China
jliu@sei.ecnu.edu.cn

Yves Robert
Univ Lyon, ENS Lyon, UCBL, CNRS,
Inria, LIP, Lyon, France
University of Tennessee Knoxville
Knoxville, USA
yves.robert@inria.fr

Frédéric Vivien*
Univ Lyon, ENS Lyon, UCBL, CNRS,
Inria, LIP, Lyon, France
frederic.vivien@inria.fr

ABSTRACT

Low energy consumption and high reliability are widely identified as increasingly relevant issues in real-time systems on heterogeneous platforms. In this paper, we propose a multi-criteria optimization strategy to minimize the expected energy consumption while enforcing the reliability threshold and meeting all task deadlines. The tasks are replicated to ensure a prescribed reliability threshold. The platforms are composed of processors with different (and possibly unrelated) characteristics, including speed profile, energy cost and failure rate. We provide several mapping and scheduling heuristics towards this challenging optimization problem. Specifically, a novel approach is designed to control (i) how many replicas to use for each task, (ii) on which processor to map each replica and (iii) when to schedule each replica on its assigned processor. Different mappings achieve different levels of reliability and consume different amounts of energy. Scheduling matters because once a task replica is successful, the other replicas of that task are cancelled, which calls for minimizing the amount of temporal overlap between any replica pair. The experiments are conducted for a comprehensive set of execution scenarios, with a wide range of processor speed profiles and failure rates. The comparison results reveal that our strategies perform better than the random baseline, with a gain of 40% in energy consumption, for nearly all cases. The absolute performance of the heuristics is assessed by a comparison with a lower bound; the best heuristics achieve an excellent performance, with an average value only 4% higher than the lower bound.

CCS CONCEPTS

• **Computer systems organization → Real-time systems**; *Reliability*; • **Theory of computation → Design and analysis of algorithms**; **Parallel algorithms**.

*Corresponding authors.

KEYWORDS

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

ACM Reference Format:
Li Han, Yiqin Gao, Jing Liu, Yves Robert, and Frédéric Vivien. 2020. Energy-aware strategies for reliability-oriented real-time task allocation on heterogeneous platforms. In . ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Real-time systems are composed of periodic tasks that are regularly input to a parallel computing platform and must complete execution before their deadlines. In many applications, another requirement is reliability: 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, 30]. 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 [11, 12].

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 [5]. 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. 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 task τ_i depends upon the processor which executes it. More precisely, the reliability $R(\tau_i, m_k)$ of a replica of τ_i mapped on processor m_k is $R(\tau_i, m_k) = e^{-\lambda_k c_{i,k}}$, where $c_{i,k}$ is the execution time of τ_i on m_k, and λ_k the failure rate of m_k. The total reliability of task τ_i is a function of the reliability of all its replicas (which we explicit in Equation 4 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 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 [9], even for deterministic tasks.

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 τ_i on m_k follows a uniform probability distribution in the range $[\beta_{\text{b/w}} c_{i,k}, c_{i,k}]$ for some constant $\beta_{\text{b/w}} < 1$ (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. Our mapping heuristics aim at minimizing energy consumption based upon a lower-bound formula that assumes no overlap between any two replicas of the same task, while our scheduling heuristics aim at minimizing such an overlap. 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:

Table 1: Key Notations

Notation	Explanation
N and M	number of tasks and of processors
p	period (deadline) for each task instance
$c_{i,k}$	WCET for task τ_i on processor m_k
$u_{i,k} = \frac{c_{i,k}}{p}$	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
$P(m_k)$	power consumed per time unit on processor m_k
E_s	total static energy consumption
$E_d(\tau_i, m_k)$	dynamic energy cost of task τ_i on processor m_k
$R(\tau_i, m_k)$	reliability of task τ_i on processor m_k

- The formulation of the tri-criteria optimization problem;
- The design of several mapping and scheduling heuristics;
- The characterization of a lower bound for energy consumption;
- An experimental evaluation based on a comprehensive set of simulations scenarios, showing that two of the heuristics achieve the best performance, and are always very close to the lower bound.

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 performance lower bound 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 $m_1, m_2, \ldots, m_M$ 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 the 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, which is equal to its deadline. In this work, we assume that all tasks have the same period p, so that a single instance of each task must execute every p seconds. Note that assuming that all tasks are atomic and with same period is the standard assumption for real-time task graphs (or DAGs) [23].

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 $u_{i,k} = \frac{c_{i,k}}{p}$. The utilization of a processor is the sum of the utilizations of all tasks that are assigned to it.

2.2 Power and energy

The power consumed per time unit on processor m_k is

$$P(m_k) = P_{k,s} + gP_{k,d} \tag{1}$$

where $P_{k,s}$ is the static power; g represents the system state and indicates whether dynamic power $P_{k,d}$ is currently being consumed by m_k: when m_k executes a task, $g = 1$, otherwise $g = 0$. 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 dynamic energy consumption $E_d(\tau_i, m_k)$ of task τ_i on processor m_k is estimated using the WCET:

$$E_d(\tau_i, m_k) = P_{k,d} \times c_{i,k} \tag{2}$$

but we use the value derived from the actual execution time in the experiments. The total static energy consumption is simply given by

$$E_s = \sum_{k \in Used} P_{k,s} \times p \tag{3}$$

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

2.3 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 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 [12].

The *reliability* of a task instance is the probability of executing it successfully, in the absence of software faults. The reliability of task τ_i on processor m_k with WCET $c_{i,k}$ is $R(\tau_i, m_k) = e^{-\lambda_k \times c_{i,k}}$. During the mapping phase, task τ_i will have several replicas executing on different processors, in order to match some reliability threshold. Let $alloc(i)$ denote the index set of the processors executing a replica of τ_i. The mapping achieves the following reliability $R(\tau_i)$ for task τ_i:

$$R(\tau_i) = 1 - \Pi_{k \in alloc(i)}(1 - R(\tau_i, m_k)) \tag{4}$$

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 (4).

Each 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 $R(\tau_i) \geq \mathcal{R}_i$ for $1 \leq i \leq N$. 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.

2.4 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 p, so that expected energy consumption is minimized, while matching the deadline p and reliability threshold $\mathcal{R}_i$ for each task τ_i. 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 (3)).
- 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 [4]. The optimization of the sole scheduling phase is also NP-hard: 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 [9]. Even if the task deadline was not constraining (very large deadline with respect to the worst-case execution time of tasks), the problem would remain NP-hard. We formally state this latter problem and show that it is NP-hard.

Definition 1 (MinEnergyMaxReliability)**.** Consider an heterogeneous platform composed of M heterogeneous processors, m_1, ..., m_M, and N (non-periodic) tasks $\tau_1, \tau_2, ..., \tau_N$. Executing task τ_i on processor m_k has an energy cost of $E(\tau_i, m_k)$ and has a probability of success of $R(\tau_i, m_k)$. Let $\mathcal{E}$ and $\mathcal{R}$ be two constants. The MinEnergyMaxReliability decision problem is: is it possible to schedule the tasks on the processors so that: (i) the total energy consumed does not exceed $\mathcal{E}$; and (ii) the probability that all tasks succeed is at least $\mathcal{R}$?

Lemma 1. *Problem MinEnergyMaxReliability is NP-complete.*

Proof. We prove this result by a reduction from the 2-partition problem [4]. Let I_1 be an instance of 2-partition with N positive integers, $a_1, ..., a_N$. Let $S = \sum_{i=1}^{N} a_i$. The question is: is it possible

to find a subset A of $\{1, ..., N\}$ such that

$$\sum_{\substack{1\le i\le N\\ i\in A}} a_i = \sum_{\substack{1\le i\le N\\ i\notin A}} a_i = \frac{S}{2}.$$

From $\mathcal{I}_1$ we build an instance $\mathcal{I}_2$ of MinEnergyMaxReliability as follows. We have $M = 2$ processors. Then we have N tasks each having the same execution times on both processors: $c_{i,1} = c_{i,2} = a_i$. The failure rates are defined by: $\lambda_1 = \frac{1}{S}$ and $\lambda_2 = 1$. The static energy is null, $P_{1,s} = P_{2,s} = 0$, and the dynamic energy is defined by $P_{1,d} = 1$, $P_{2,d} = \frac{1}{S}$. Therefore, we have:

$$\begin{cases} E(\tau_i, m_1) = a_i \text{ and } R(\tau_i, m_1) = e^{-\frac{a_i}{S}} \\ E(\tau_i, m_2) = \frac{a_i}{S} \text{ and } R(\tau_i, m_2) = e^{-a_i} \end{cases}$$

Finally, we let $\mathcal{E} = \frac{1}{2}(S+1)$ and $\mathcal{R} = e^{-\frac{1}{2}(S+1)} = e^{-\mathcal{E}}$.

One can easily check that the size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$, that all the $E(\tau_i, m_k)$'s are positive and that all the $R(\tau_i, m_k)$'s are strictly between 0 and 1.

Let us consider any mapping of the M tasks on the two processors. Let A be the index set of tasks mapped on processor 1 in this mapping, and let $\mathcal{A} = \sum_{i\in A} a_i$. Let E be the total energy consumed by this mapping and let R be the reliability of the whole set of tasks. We have: $E = \mathcal{A}\frac{S-1}{S} + 1$ and $\ln(R) = -S + \mathcal{A}\frac{S-1}{S}$. One can then easily show that $\mathcal{I}_2$ has a solution if and only if $\mathcal{I}_1$ has a solution. □

3 MAPPING

In the mapping phase, we need to define the number of replicas for each task, as well as the execution processor for every replica, aiming at meeting the reliability target while minimizing the energy cost. One difficulty introduced by platform heterogeneity is that we do not know the number of replicas needed for each task 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 [9, 12] cannot be applied.

As shown in Algorithm 1, given a set of tasks with their reliability targets and a set of heterogeneous processors, we first order the tasks according to TaskMapCriteria, which includes:

- *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 $\bar{c_i} = \min_{1\le k\le M} c_{i,k}$;
- *deMaxW* (*inMaxW*): decreasing (increasing) maximum work size $\bar{c_i} = \max_{1\le k\le M} c_{i,k}$:
- *random*: random ordering.

Then, for each task in the ordered list, we order the processors for mapping its replicas according to ProcMapCriteria, which includes:

- *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.

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	2	0.99	0.495	1
3	1	0.99	0.99	2
4	2	0.9	0.45	0.5

We use the example shown in Table 2 to explain how to design a better criteria in ProcMapCriteria. Assume there are four processor sets with different energy and reliability configurations. Considering only the reliability, we cannot distinguish between the second and third sets. Apparently, the third set is better since its processors consume 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 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 one processor from the second set or two processors from the first set. 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 criteria *deP* in ProcMapCriteria, 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, we add replicas for task τ_i in the order of the processor list until the reliability target $\mathcal{R}_i$ 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 (4)). 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 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. Our approach is to identify a primary replica for each task, then all its other replicas become secondaries. The goal of the proposed scheduling is to avoid overlap between the execution of the primary and secondary replicas for each task: the primary must be terminated as soon as possible, while the secondaries must be delayed as much as possible. Whenever a primary replica of a task succeeds, the energy consumption will be minimal for that task if no secondary replica has started executing yet. Our scheduling algorithm uses a layered approach: first we map the first replica of each task, which we call the primary replica; and then, in a round-robin fashion, we map the remaining replicas (if they exist), which we call the secondaries. Here is a detailed description of Algorithm 2:

Algorithm 1: Replication setting and mapping

```
Input: A set of tasks τ_i with reliability targets R_i;
a set of heterogeneous processors m_k
Output: An allocation σ_m of all replicas on the processors
1  begin
2      order all the tasks with TaskMapCriteria and renumber them
         τ_1, ..., τ_N
       /* initialize the utilization of all processors to zero */
3      u ← [0, ..., 0]
       /* iterate through the ordered list of tasks            */
4      for i ∈ [1, ..., N] do
           /* order processors for each task                   */
5          order all processors for task τ_i with ProcMapCriteria and
             renumber them proc_1, ..., proc_M
           /* this ordered list may differ from task to task   */
6          k = 1
7          PoF = 1
8          while 1 − PoF < R_i do
9              temp = u_k + u_{i,k}
10             if temp ≤ 1 then
11                 u_k = temp
12                 PoF = PoF × (1 − R(τ_i, m_k))
13                 add one replica of τ_i on proc_k
14             k++
15             if k > m then
16                 return not feasible
17     return σ_m
```

Algorithm 2: Scheduling

```
Input: An allocation σ_m of all replicas on the processors
Output: An order of execution on each processor
1  begin
2      order all the tasks with TaskSchedCriteria and renumber them
         τ_1, ..., τ_N
       /* insert the primary replica for each task at the
          beginning of each processor schedule                */
3      for i ∈ [1, ..., N] do
4          if PrimarySchedCriteria is "time" then
5              schedule the primary replica of τ_i that could finish at the
                 earliest
6          else if PrimarySchedCriteria is"energy" then
7              schedule the primary replica of τ_i that consumes the
                 minimum energy
       /* insert the secondaries backwards from the end of each
          processor schedule                                  */
8      reverse the task ordering
9      while there is still at least one replica to be scheduled do
10         for i ∈ [1, ..., N] do
11             if there is still a replica of τ_i to be scheduled then
12                 if SecondarySchedCriteria is "time" then
13                     schedule the secondary replica of task τ_i that could
                         start the latest
14                 else if SecondarySchedCriteria is "energy" then
15                     schedule the secondary replica of task τ_i that
                         consumes the maximum energy
```

(1) First we order tasks by criterion TaskSchedCriteria, for which we propose:
   - *deNR* (*inNR*): decreasing (increasing) number of replicas;
   - *deU* (*inU*): decreasing (increasing) total utilization (sum up the utilization of all replicas);
   - *random*: random ordering.

(2) Then we process the list of tasks in that order, and select a primary replica for each task, which we execute as soon as possible on its assigned processor, right after already scheduled primary replicas (if any). We use two different criteria PrimarySchedCriteria for selecting primary replicas:
   - *time:* choose the processor that can complete the execution of the replica the earliest (given already made scheduling decisions);
   - *energy:* choose the processor that can execute the replica with smallest dynamic energy.

(3) Once primary replicas have all been scheduled, we reverse the order of the list of tasks, and we schedule the remaining replicas (considered in a round-robin fashion in the reversed list) as late as possible on their assigned processor. The idea is to mimimize potential overlap between primary and secondaries for each task, hence to delay secondary replicas until the end of the period. The rationale for reverting the task list is that the primary replica of some task τ at the end of the list may have been scheduled after some other primary replica τ', hence the idea to process the secondary replica of τ' before that of τ and push it further away at the end of the period.

(4) Finally, there only remains to detail which secondary replica of a task is scheduled first (whenever the task has three replicas or more). We also have two criteria SecondarySchedCriteria for choosing secondary replicas:
   - *time:* choose the replica whose start-up time can be the latest (given already made scheduling decisions); the idea is to minimize overlap by maximizing slack;
   - *energy:* choose the replica whose energy is the highest; the idea is again to minimize overlap, thereby increasing the probability of this costly replica to be cancelled.

As we have two different criteria for both selecting primaries and secondaries, in total, we have four possible combinations, namely *time-time*, *time-energy*, *energy-time* and *energy-energy*. For the baseline scheduling (*randomShuffling*), we randomly order tasks on each processor and execute them in sequence and as early as possible (no idle time until the end of the period).

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.

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 ordering of all replicas. We first use Monte-Carlo simulations (see Section 6) and generate several sets of values for the actual execution time $w_{i,k}$ of τ_i on m_k. The values $w_{i,k}$ 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 $w_{i,k}$, 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 task 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. 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 (3). This leads to the energy consumption of the failure scenario. We average the values over all failure scenarios and obtain the expectation, denoted as $E(\{w_{i,k}\})$.

In addition, we also compute a lower bound $LB(\{w_{i,k}\})$ as follows. Our goal is to accurately estimate the energy consumption of an optimal solution. Because the static energy depends upon the subset of processors that are used in the solution (see Equation (3)), we need to try all possible subsets. Given a processor subset $\mathcal{S}$, we consider each task τ_i independently, and try all possible mappings of replicas of τ_i 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 τ_i on each processor of $\mathcal{T}$ meets the reliability criteria $\mathcal{R}_i$, 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 $w_{i,k}$, because this is part 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

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

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). Note that here we use execution times and failure probabilities based upon the actual execution times $w_{i,k}$ 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 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 $LB(\{w_{i,k}\})$ 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 code, data and analysis are publicly available in [8]. A companion research report with the comprehensive set of results is available in [10].

6.1 Experimental methodology

In the experiments, we have $M = 10$ processors and $N = 20$ tasks which have all the same period $p = 100$.[1] 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}). These two parameters vary between 0 and 1.[2] 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 also define a parameter *basicWork* as the estimated total utilization of the system with a single replica per task, in order to study the impact of system pressure:

$$basicWork = \frac{\sum_{i,k} c_{i,k}}{M^2 p} = \frac{\sum_{i,k} u_{i,k}}{M^2} \tag{5}$$

In Equation (5), we use the average WCETs on the M processors ($\frac{\sum_k c_{i,k}}{M}$) to estimate the execution time of task τ_i. We have M processors available during period p, hence *basicWork* represents an estimate of the fraction of time 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 of tasks from their 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 and its WCET. Therefore, the actual execution time 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, β_i, is task dependent: β_i describes whether the instance of a task is a small one or a large one. β_i is randomly drawn in $[0, 1]$. A value of $\beta_i = 0$ means that task τ_i has the shortest execution time possible, and $\beta_i = 1$ means that the actual execution is equal to its worst case execution time. Overall, the actual execution time of task τ_i on processor m_k is thus defined as: $w_{i,k} = (\beta_{\text{b/w}} + (1 - \beta_{\text{b/w}})\beta_i)c_{i,k}$.

For a processor m_k in the platform, we fix the static power $P_{k,s}$ at 0.001 as in previous papers [22, 23, 26]. But for the dynamic power and the failure rate, we have two sets of parameters. The first set also follows values similar to those of the previous papers [22, 23, 26]. For this set, we have a relatively large power and a very small failure rate. Therefore, the replicas using this first set of parameters succeed in almost all cases. Thus, to evaluate our heuristics in the context when failures occur more frequently, we

[1] One execution is around 0.150 milliseconds with these parameters on a basic laptop.
[2] We ignored the case when $cor_{\text{task}} = 0$ and $cor_{\text{proc}} = 1$ for the parameter set with big failure rate, because when tasks are completely unrelated, there (very likely) is a task with very long execution time on all processors ($cor_{\text{proc}} = 1$). The number of replicas needed to meet its reliability goal will exceed the number of available processors.

introduce a second set of parameters where the replicas have a smaller power and a larger 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 always 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, 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 [10]. The general trends about the relative performance of heuristics is the same in both sets, and we make specific comments when it not the case.

We vary the local reliability target $\mathcal{R}_i$ between 0.9 and 0.98 for the first set and between 0.8 and 0.95 for the second set. This is to give the system a reasonable freedom while mapping and scheduling. The reliability target is relatively high, implying that tasks need plural replicas to reach it. Therefore, we give more tolerance (smaller relialbility threshold) to the second set with a larger failure rate, because otherwise we may not be able to find feasible mappings.

Table 3: Ratio of energy consumption to the baseline of different mapping and scheduling ordering tasks criteria

map \ sch	*deNR*	*inNR*	*deU*	*inU*	*random*
deW	0.5655	0.5662	0.5655	0.5660	0.5663
inW	0.5631	0.5635	0.5630	0.5635	0.5635
deMinW	0.5658	0.5662	0.5657	0.5661	0.5665
inMinW	0.5637	0.5642	0.5637	0.5642	0.5641
deMaxW	0.5658	0.5664	0.5657	0.5663	0.5665
inMaxW	0.5629	0.5633	0.5629	0.5633	0.5633
random	0.5633	0.5639	0.5633	0.5638	0.5639

6.2 Results

In this section, we analyze the impact of the different parameters on the performance of the heuristics. Due to lack of space, the comprehensive set of all results is available in the extended version [10]. We choose as default values $\beta_{\text{b/w}} = 1$, $basicWork = 0.3$, $\mathcal{R}_i = 0.95$ for the set with big failure rate, and $\mathcal{R}_i = 0.98$ for the set with small failure rate. This set of parameters is chosen to constrain the solution so that we can observe the most interesting results. For cor_{task} and cor_{proc}, we fix them at 0.5 as default value. Each experiment is the average of 10 sets of WCET values. For each set, 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 of 10 executions. Overall, we run 1,000 randomly generated experiments for each set of $\beta_{\text{b/w}}$, $basicWork$, $\mathcal{R}_i$, cor_{task} and cor_{proc} values. The total number of experiments ran is 3,075,000 for each heuristic. Each result is represented as a ratio to the random baseline method which is defined as follows: for each task, we add replicas randomly on available processors until reaching its 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 also compare different strategies with the lower bound proposed in Section 5. We report on these figures the average number of replicas needed in total for 20 tasks (on the left side) and of failures that occur for the 1,000 random trials for each setting (on the right side). These numbers are reported in black above the horizontal axis in each figure.

6.2.1 Ordering tasks for mapping and scheduling. In Table 3, we calculated the ratio of combinations of different mapping and scheduling methods to the baseline method, when considering different criteria for ordering tasks. We can see that, in the whole set of experiments, all criteria for ordering tasks perform equally well (around 56%). The difference between the best and the worst performance is only around 0.36%. Hence these criteria do not critically influence energy consumption. In the following results, for the task ordering, we only consider the decreasing average WCET (*deW*) for the mapping, and the decreasing utilization (*deU*) for the scheduling, which give priority to the tasks that putting more pressure to the system. We then focus on selecting processors during the mapping phase, and on choosing primary and secondary replicas during the scheduling phase.

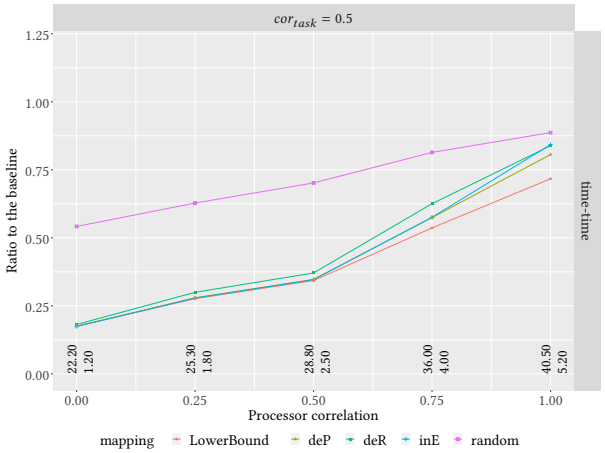
6.2.2 Processor correlation. Figure 1 shows results when processor correlation varies. We found that our strategies consume less than 25% of the energy needed by the baseline strategy when $cor_{\text{proc}} = 0$, and the result is close to the lower bound. But we can observe that this percentage increases with cor_{proc}.

For the mapping phase, we can see from Figure 1(a) that, when cor_{proc} is not equal to 1, *inE* performs better than *deR*. But when cor_{proc} increases to 1, the performance of *deR* catches up. And in all cases, *deP* performs better than, or similarly to the best strategy between *deR* and *inE*.

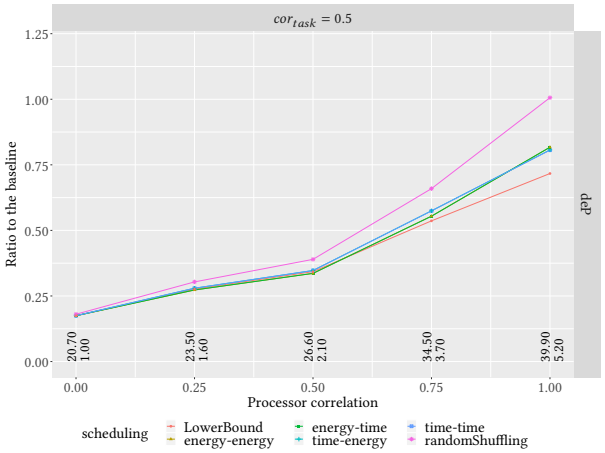
For the scheduling strategies, Figure 1(b) shows that there is little difference between our different criteria, the random one excepted. But we can still observe that, with $cor_{\text{proc}} \neq 1$, we have *energy* criteria for primary replica choosing slightly better than *time*, and when $cor_{\text{proc}} = 1$, the *energy* becomes worse than the *time*.

This is because, for most of the cases, the reliability of our replica is high, so that we can simply choose the replica which costs the least energy as primary, and delete all secondary when it finishes successfully. But in the case of $cor_{\text{proc}} = 1$, the WCETs of each task on different processors are the same, so the order of the processors for any task is the same, and is relative to the power and reliability parameters ($P_{k,d}$ and λ_k). This can result in a few fully used processors, with the other processors being empty. Also, for *time* criteria, primary replicas will be randomly balanced on different fully used processors, because every replica of a task has the same WCET. But for *energy*, the processors which cost less energy are the same for all tasks. Then these processors will execute all mapped replicas as primary, and others will execute all mapped replicas as secondary, which increases the overlap. This is why, when $cor_{\text{proc}} = 1$, we cannot save as much energy as in other cases, and this is why the *energy* criteria performs worse than the *time* criteria.

6.2.3 Task variability. Figure 2 presents the results when $\beta_{\text{b/w}}$ varies. We observe that, for almost all the mapping and scheduling

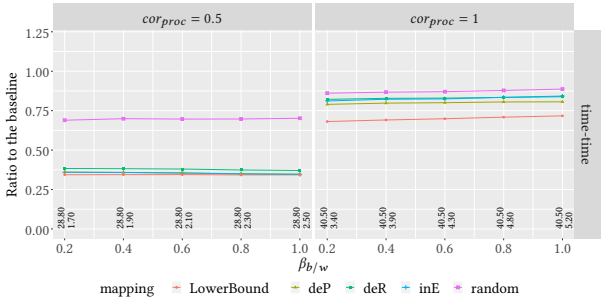


(a) Comparing mapping strategies when using *time-time* as scheduling strategy

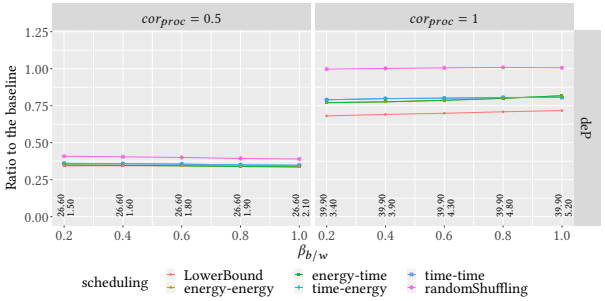


(b) Comparing scheduling strategies when using *deP* as mapping strategy

Figure 1: Ratio of energy consumption using different mapping and scheduling strategies under big failure rate, when varying cor_{proc}, with *basicWork* = 0.3, $\beta_{\text{b/w}} = 1$, $\mathcal{R}_i = 0.95$ and $cor_{\text{task}} = 0.5$.



(a) Comparing mapping strategies when using *time-time* as scheduling strategy



(b) Comparing scheduling strategies when using *deP* as mapping strategy

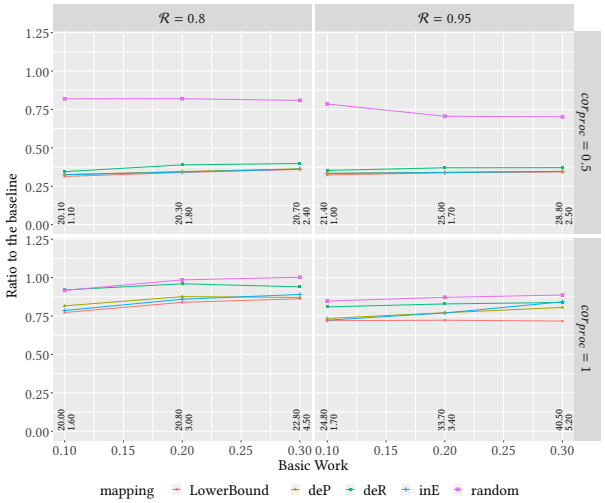
Figure 2: Ratio of energy consumption using different mapping and scheduling strategies under big failure rate when varying $\beta_{\text{b/w}}$, with *basicWork* = 0.3, $\mathcal{R}_i = 0.95$ and $cor_{\text{task}} = 0.5$.

criteria, the results are similar whatever the value of $\beta_{\text{b/w}}$. This is because we map and schedule tasks based on their WCETs, so the mapping and scheduling results are independent of the value of $\beta_{\text{b/w}}$. Furthermore, each task i has the same β_i on the different processors. Therefore the energy consumption ratios tend to be similar. But in the case of $cor_{\text{proc}} \approx 1$, we can see that the ratio of *energy* scheduling criteria increases with $\beta_{\text{b/w}}$. In fact, when we have a larger value of $\beta_{\text{b/w}}$, the actual execution time is closer to the WCET. So that, although the mappings and schedulings are the same for different $\beta_{\text{b/w}}$, replicas will take longer time during the actual execution. At the same time, as explained in the previous paragraph, we have a more serious overlap in the case of $cor_{\text{proc}} = 1$ and for *energy* criteria. This is why $cor_{\text{proc}} = 1$ performs differently when varying $\beta_{\text{b/w}}$. However this phenomenon is not obvious ($\approx 5\%$), so we can always conclude that the result is independent of $\beta_{\text{b/w}}$. When $\beta_{\text{b/w}}$ is small, actual execution times can greatly differ from the WCETs used for mapping and scheduling. However, in this case, our heuristics have a performance similar to that of the lower bound, which shows that they are very robust.

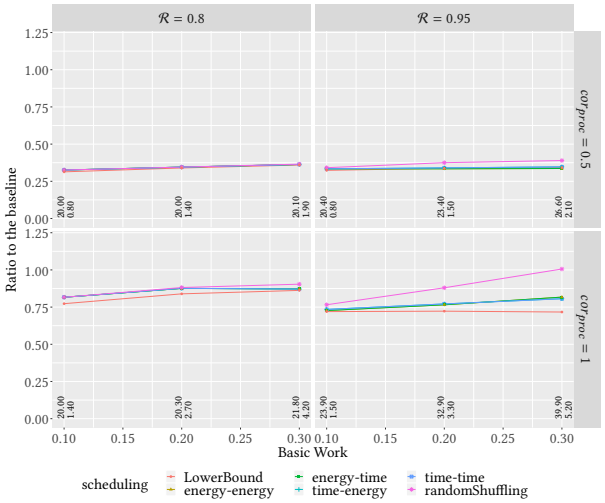
6.2.4 Utilization and reliability threshold. From Figure 3, we observe the performance of different mapping and scheduling criteria when varying *basicWork* and $\mathcal{R}_i$.

We can see the case $cor_{\text{proc}} = 0.5$ in the first row of Figures 3. During the mapping phase, *deR* has a slightly worse performance than *inE* and *deP*, which perform similarly to the lower bound. During the scheduling phase, all criteria have similar performance, including the random strategy and the lower bound.

With $cor_{\text{proc}} \approx 1$, during the mapping phase, the second row of Figure 3(a) shows that the difference between different criteria becomes smaller when *basicWork* and $\mathcal{R}_i$ increase. Inversely, the difference with the lower bound becomes larger, but it is still less than 10% (except random), even in the worst case. For small values of *basicWork* and $\mathcal{R}_i$, *deR* performs worse, but when *basicWork* and $\mathcal{R}_i$ increase, *deR* becomes similar to *inE*, or even better. *deP* has



(a) Comparing mapping strategies when using *time-time* as scheduling strategy



(b) Comparing scheduling strategies when using *deP* as mapping strategy

Figure 3: Ratio of energy consumption using different mapping and scheduling strategies when varying *basicWork* and $\mathcal{R}_i$, under big failure rate, with $\beta_{\mathbf{b/w}} = 1$ and $cor_{\mathbf{task}} = 0.5$.

always better or similar performance than *deR* and *inE*. For the scheduling phase, from the second row of Figure 3(b), we can still find similar performance on all our criteria, but their difference with the lower bound increases to 10% as in the mapping phase. The reason is that with the increase of system load, it becomes harder to map all replicas to their best processors, while the lower bound is calculated without considering utilization constraints.

6.2.5 Number of failures. 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 0.44% failed replicas. Thus, in most of the cases, it is enough to have a single replica mapped on the processor that costs the least energy, because failures are scarce. On the contrary, in the set with big failure rate, the average rate of failed replicas increases to 7.57%. We can observe that the *deP* mapping method used in conjunction with either the *time* or *energy* scheduling criteria achieves the best performance, or a performance similar to the best observed one, in both cases. This confirms that the performance of our best heuristics is not affected by the failure rate.

6.2.6 Success rate. All the tested heuristics were able to find a valid solution in all tested configurations with small failure rate. And in the big failure rate cases, heuristics were able to build valid solutions for more than 99.94% of the instances. The very high success rate of our experiments shows the robustness of our approach.

6.2.7 Summary. In conclusion, our strategies can save more than 40% of the energy consumed by the baseline, except in the high processor correlation case. The ratio to the baseline can be as low as 20% in the best case. As for the different criteria used in the heuristics, we find that the *deP* method is the best processor ordering during the mapping phase. For scheduling, we can find from the result that, all our primary-secondary choosing criteria have a similar result as the lower bound, except a difference of 10% in the case of $cor_{\text{proc}} \approx 1$. This means that our primary-secondary choosing heuristic performs well. On the other hand, we point out that strategies with the same method for choosing the primary replica but different methods for choosing secondary replicas, perform similarly. Hence the strategy for choosing the primary replica has much more impact than the one for choosing secondary replicas. We can find that, *time-time* and *time-energy* criteria performs better when $cor_{\text{proc}} \approx 1$, and *energy-energy* and *energy-time* have better performance in other cases.

The performance of these best heuristics is only 17.0% higher than the lower bound in the worst case. Furthermore, we report a median value only 3.5% higher than that of the lower bound; and the average value is only 4.3% higher. We can confidently conclude that our best strategies perform remarkably well over the whole experimental setting.

7 RELATED WORK

7.1 Scheduling for heterogeneous platforms

There is a huge literature on scheduling for heterogeneous platforms, and even dedicated workshops. Here, we only refer some very recent work closely related to our problem but dealing with non-periodic tasks. [22] maximizes the reliability of an energy-constrained DAG executed on a heterogeneous platform while using DVFS. Conversely, [26] minimizes the energy consumption of a reliability-constrained DAG executed on a heterogeneous platform while using or not DVFS. A group of authors published a book [24] and several articles on the problem of DAG scheduling on heterogeneous platforms. In Chapter 2 of book [24] and in [25] these authors consider the energy minimization when scheduling a DAG with or without DVFS. In [23] they considered the same

problem while satisfying some reliability goal. However, these two results do not consider reliability.

Overall, these studies do not consider real-time applications. The deadlines constraining real-time tasks and applications make problems significantly harder to tackle.

7.2 Scheduling of real-time applications on homogeneous platforms

There is 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 [18] where only 9 of the 78 references deal with heterogeneous platforms. [12] minimizes the energy when scheduling independent tasks with different deadlines on a homogeneous platform while satisfying some threshold on reliability. The study [9] improved the solution from [12] in particular by carefully avoiding overlaps between primary and secondary replicas. [11] considers the same problem; however, it uses checkpointing to cope with failures when all other work consider replication.

We refer the interested reader to [3, 9, 12, 18] for a comprehensive overview of the related work for homogeneous 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. Heuristics have thus to perform complicated trade-offs in our three-criteria settings.

7.3 Scheduling of real-time applications on heterogeneous platforms

Some related work targets the scheduling of real-time applications on heterogeneous platforms, but without considering fault tolerance. For instance, [29] 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). [7] 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. [14], [17] and [28] minimize energy consumption by using DVFS, [14] when scheduling independent tasks, [17] a DAG, and [28] a moldable application. [21] considers the scheduling of independent tasks and DAGs under an energy constraint, while [20] considers the scheduling of independent tasks under a thermal constraint. [27] 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 of the 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. [15] 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. [16] 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 as at most one of them will need to be executed. [13] 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. [6] 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. [19] 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). [5] 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 increase energy consumption. As a consequence, [5] produces energy greedy schedules (see subplots (a-1), (b-1), and (c-1) of Figure 1 in [5]). In Chapter 3 of the already mentioned book [24], the authors consider cost minimization (which can be energy minimization) when scheduling a DAG under deadline and reliability constraints. Therefore, we consider the same problem but for a set of independent tasks rather than for a DAG. Because of the dependence between tasks and the chosen as-soon-as-possible scheduling of [24], this solution tends to schedule simultaneously the different replicas of a single task. As already pointed out in the studies [9, 12] that this can lead to a significant waste of energy. Therefore, it would have been unfair to compare our solution to that of [24] applied on independent tasks.

From what precedes, we have only identified 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 the overlapping between replicas of a same task, what have been previously proved to be crucial [9] and what we specifically targeted (cf. Section 4).

8 CONCLUSION

In this work, we have studied the problem of executing periodic real-time tasks on an heterogeneous platform, with several objectives: minimizing the energy consumption, guaranteeing some reliability thresholds, and meeting all deadlines. For each task, we decide how many replicas should be launched, and on which processors to map them. We tagged one replica per task as “primary” replica and the other ones as “secondary” replicas. To obtain an absolute measure

for the evaluation of our heuristics, we have computed a theoretical lower bound on energy consumption. Extensive simulations show that our best heuristic always achieve very good performance, very close to the lower bound (on average only 4% higher than this lower bound). 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), by executing primary replicas as soon as possible and secondary ones as late as possible, and by tagging replicas as "primary" using an earliest completion time criterion when processors are highly correlated, using an smallest energy criterion otherwise. Furthermore, while all decisions are taken with the worst-case execution times (WCETs) of tasks as only input, the simulations used the actual execution times; the best heuristic always achieved excellent performance even when the actual execution times were far smaller than the WCETs, showing the robustness of our approach.

Future work will 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.

ACKNOWLEDGMENTS

This paper is partially supported by the National Key Research and Development under Project 2019YFA0706404, NSFC 61972150, 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). First two authors contributed equally to this work.

REFERENCES

[1] Louis-Claude Canon, Mohamad El Sayah, and Pierre-Cyrille Héam. 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, 460–467.

[2] H. Chen, J. Wen, W. Pedrycz, and G. Wu. 2020. Big Data Processing Workflows Oriented Real-Time Scheduling Algorithm using Task-Duplication in Geo-Distributed Clouds. *IEEE Trans. Big Data* 6, 1 (2020), 131–144.

[3] Robert I. Davis and Alan Burns. 2011. A Survey of Hard Real-time Scheduling for Multiprocessor Systems. *ACM Comput. Surv.* 43, 4, Article 35 (Oct. 2011), 44 pages. https://doi.org/10.1145/1978802.1978814

[4] M. R. Garey and D. S. Johnson. 1979. *Computers and Intractability, a Guide to the Theory of NP-Completeness.* W.H. Freeman and Company.

[5] Tian Guo, Jing Liu, Wei Hu, and Mengxue Wei. 2018. Energy-Aware Fault-Tolerant Scheduling Under Reliability and Time Constraints in Heterogeneous Systems. In *Intelligent Computing Methodologies*, De-Shuang Huang, M. Michael Gromiha, Kyungsook Han, and Abir Hussain (Eds.). Springer, 36–46.

[6] Yifeng Guo, Dakai Zhu, Hakan Aydin, Jian-Jun Han, and Laurence T. Yang. 2017. Exploiting primary/backup mechanism for energy efficiency in dependable real-time systems. *Journal of Systems Architecture* 78 (2017), 68 – 80. https://doi.org/10.1016/j.sysarc.2017.06.008

[7] Jian-Jun Han, Wen Cai, and Dakai Zhu. 2018. Resource-aware Partitioned Scheduling for Heterogeneous Multicore Real-time Systems. In *Proceedings of the 55th Annual Design Automation Conference* (San Francisco, California) *(DAC '18)*. ACM, New York, NY, USA, Article 124, 6 pages. https://doi.org/10.1145/3195970.3196103

[8] Li HAN. 2020. Heterogeneous real-time systems. (3 2020). https://doi.org/10.6084/m9.figshare.11925423.v1

[9] Li Han, Louis-Claude Canon, Jing Liu, Yves Robert, and Frédéric Vivien. 2020. Improved energy-aware strategies for periodic real-time tasks under reliability constraints. In *40th IEEE Real-Time Systems Symposium (RTSS)*.

[10] Li Han, Yiqin Gao, Jing Liu, Yves Robert, and Frédéric Vivien. 2020. *Energy-aware strategies for reliability-oriented real-time task allocation on heterogeneous platforms.* Research report 92324. INRIA.

[11] Qiushi Han. 2015. *Energy-aware Fault-tolerant Scheduling for Hard Real-time Systems.* Ph.D. Dissertation. Florida International University. https://doi.org/10.25148/etd.FIDC000077

[12] Mohammad A Haque, Hakan Aydin, and Dakai Zhu. 2017. On reliability management of energy-aware real-time systems through task replication. *IEEE Transactions on Parallel and Distributed Systems* 28, 3 (2017), 813–825.

[13] K. Huang, X. Jiang, X. Zhang, R. Yan, K. Wang, D. Xiong, and X. Yan. 2018. Energy-Efficient Fault-Tolerant Mapping and Scheduling on Heterogeneous Multiprocessor Real-Time Systems. *IEEE Access* 6 (2018), 57614–57630. https://doi.org/10.1109/ACCESS.2018.2873641

[14] Sanjay Moulik, Rishabh Chaudhary, and Zinea Das. 2020. HEARS: A heterogeneous energy-aware real-time scheduler. *Microprocessors and Microsystems* 72 (2020), 102939. https://doi.org/10.1016/j.micpro.2019.102939

[15] Xiao Qin and Hong Jiang. 2006. A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems. *Parallel Comput.* 32, 5-6 (2006), 331–356.

[16] W. Qiu, Z. Zheng, X. Wang, and X. Yang. 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)*. 1–7. https://doi.org/10.1109/ISORC.2013.6913213

[17] Monire Safari and Reihaneh Khorsand. 2018. Energy-aware scheduling algorithm for time-constrained workflow tasks in DVFS-enabled cloud environment. *Simulation Modelling Practice and Theory* 87 (2018), 311 – 326. https://doi.org/10.1016/j.simpat.2018.07.006

[18] Saad Zia Sheikh and Muhammad Adeel Pasha. 2018. Energy-Efficient Multicore Scheduling for Hard Real-Time Systems: A Survey. *ACM Trans. Embed. Comput. Syst.* 17, 6, Article 94 (Dec. 2018), 26 pages. https://doi.org/10.1145/3291387

[19] Ranjani Sridharan and Rabi Mahapatra. 2010. Reliability Aware Power Management for Dual-processor Real-time Embedded Systems. In *Proceedings of the 47th Design Automation Conference* (Anaheim, California) *(DAC '10)*. ACM, New York, NY, USA, 819–824. https://doi.org/10.1145/1837274.1837480

[20] T. Tsai, Y. Chen, X. He, and C. Li. 2018. STEM: A Thermal-Constrained Real-Time Scheduling for 3D Heterogeneous-ISA Multicore Processors. *IEEE Trans. Comput.* 67, 6 (June 2018), 874–889. https://doi.org/10.1109/TC.2017.2783941

[21] Eduardo Bezerra Valentin. 2017. *Scheduling hard real-time tasks in heterogeneous multiprocessor platforms subject to energy and temperature constraints.* Ph.D. Dissertation. Universidade Federal do Amazonas.

[22] Xiongren Xiao, Guoqi Xie, Cheng Xu, Chunnian Fan, Renfa Li, and Keqin Li. 2018. Maximizing reliability of energy constrained parallel applications on heterogeneous distributed systems. *Journal of Computational Science* 26 (2018), 344 – 353. https://doi.org/10.1016/j.jocs.2017.05.002

[23] Guoqi Xie, Yuekun Chen, Xiongren Xiao, Cheng Xu, Renfa Li, and Keqin Li. 2018. Energy-efficient fault-tolerant scheduling of reliable parallel applications on heterogeneous distributed embedded systems. *IEEE Transactions on Sustainable Computing* 3, 3 (2018), 167–181.

[24] Guoqi Xie, Gang Zeng, Renfa Li, and Keqin Li. 2019. *Scheduling Parallel Applications on Heterogeneous Distributed Systems.* Springer Singapore.

[25] G. Xie, G. Zeng, X. Xiao, R. Li, and K. Li. 2017. Energy-Efficient Scheduling Algorithms for Real-Time Parallel Applications on Heterogeneous Distributed Embedded Systems. *IEEE Transactions on Parallel and Distributed Systems* 28, 12 (Dec 2017), 3426–3442. https://doi.org/10.1109/TPDS.2017.2730876

[26] Hongzhi Xu, Renfa Li, Chen Pan, and Keqin Li. 2019. Minimizing energy consumption with reliability goal on heterogeneous embedded systems. *J. Parallel and Distrib. Comput.* 127 (2019), 44–57.

[27] C. Yang, J. Chen, T. Kuo, and L. Thiele. 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*. 694–699. https://doi.org/10.1109/DATE.2009.5090754

[28] Houssam-Eddine Zahaf, Abou El Hassen Benyamina, Richard Olejnik, and Giuseppe Lipari. 2017. Energy-efficient scheduling for moldable real-time tasks on heterogeneous computing platforms. *Journal of Systems Architecture* 74 (2017), 46 – 60. https://doi.org/10.1016/j.sysarc.2017.01.002

[29] Houssam-Eddine Zahaf, Nicola Capodieci, Roberto Cavicchioli, Marko Bertogna, and Giuseppe Lipari. 2019. A C-DAG task model for scheduling complex real-time tasks on heterogeneous platforms: preemption matters. (Jan. 2019). https://hal.archives-ouvertes.fr/hal-01971594 working paper or preprint.

[30] X. Zhu, J. Wang, H. Guo, D. Zhu, L. T. Yang, and L. Liu. 2016. Fault-Tolerant Scheduling for Real-Time Scientific Workflows with Elastic Resource Provisioning in Virtualized Clouds. *IEEE Trans.Parallel and Distributed Systems* 27, 12 (2016), 3501–3517.