# Energy-aware mapping and scheduling strategies for real-time workflows under reliability constraints

Zhiwei Wu, Li Han, Jing Liu, Yves Robert, Frédéric Vivien

# Highlights

**Energy-aware mapping and scheduling strategies for real-time workflows under reliability constraints**

Zhiwei Wu[a,b], Li Han[a], Jing Liu[a,d], Yves Robert[b,c], Frédéric Vivien[b]

- Research highlight 1: Tri-criteria mapping and scheduling problem (energy, deadline, reliability) for real-time workflows of arbitrary shape
- Research highlight 2: First general solution without any limitation on the number of replicas used to match the reliability threshold of each task
- Research highlight 3: Innovative solution based upon layers of the workflow graph outperforms competitors from the literature

# Energy-aware mapping and scheduling strategies for real-time workflows under reliability constraints

Zhiwei Wu[a,b], Li Han[a*], Jing Liu[a,d*], Yves Robert[b,c], Frédéric Vivien[b*]

[a] *East China Normal University, Shanghai, China*
[b] *Univ Lyon, ENS Lyon, UCBL, CNRS, Inria, LIP, F-69342, Lyon Cedex 07, France*
[c] *University of Tennessee, Knoxville, USA*
[d] *Shanghai Key Lab of Trustworthy Computing, Shanghai, China*

**Abstract**

This paper focuses on energy minimization for the mapping and scheduling of real-time workflows under reliability constraints. Workflow instances are input periodically to the system. Each instance is composed of several tasks and must complete execution before the arrival of the next instance, and with a prescribed reliability threshold. While the shape of the dependence graph is identical for each instance, task execution times are stochastic and vary from one instance to the next. The reliability threshold is met by executing several replicas for each task. The target platform consists of identical processors equipped with Dynamic Voltage and Frequency Scaling (DVFS) capabilities. A different frequency can be assigned to each task replica to save energy, but it may have negative effect on the deadline and reliability target.

This difficult tri-criteria mapping and scheduling problem (energy, deadline, reliability) has been studied only recently for workflows with arbitrary dependence constraints. We investigate new mapping and scheduling strategies based upon layers in the task graph. These strategies better balance replicas across processors, thereby decreasing the time overlap between different replicas of a given task, and saving energy. We compare these strategies with two state-of-the-art approaches and a reference baseline on a variety of benchmark workflows. Our best heuristics achieve an average energy gain of 60% over the competitors and of 82% over the baseline.

*Keywords:* real-time workflows, energy-aware, reliability, makespan, mapping, scheduling, tri-criteria optimization

## 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 the simpler version of the problem, the tasks are independent. However, many scientific applications from various disciplines are structured as workflows [1]. Informally, a workflow can be seen as the composition

*corresponding author

*Email addresses:* hanli@sei.ecnu.edu.cn (Li Han[a]), jliu@sei.ecnu.edu.cn (Jing Liu[a,d]), frederic.vivien@inria.fr (Frédéric Vivien[b])

of a set of basic operations that have to be performed on a given input data set to produce the expected scientific result. The development of complex middleware with workflow engines [2, 3, 4] has automated workflow management, providing even more appeal to a task-based approach in a variety of scenarios. This leads to scheduling real-time workflows: instances of the same workflow are input periodically to the system and must be completed by their deadline, which is the inter-arrival time of two consecutive instances. All dependence constraints, including communication costs, must be enforced among the tasks, which complicates the derivation of a schedule that meets the deadline. All workflow instances have the same dependence graph but operate on different data sets. Task execution times are stochastic and vary from one instance to the next; typically, it is assumed that they follow some probability distribution with bounded support. Scheduling and mapping decisions are taken based upon the *Worst Case Execution Time (WCET)* of each task, which is the largest value of the distribution support, i.e., the range of non-zero values of the probability density function. Mapping and static schedule tasks based on WCETs is the de-facto standard approach in real-time systems, which is pessimistic but mandatory, to guarantee that deadlines are met in every possible execution. Then at runtime the schedule is updated according to actual execution times of task instances, which obey the distribution and may be significantly smaller than their WCETs.

When scheduling a single workflow instance on a parallel platform, the traditional objective is to minimize the makespan, or total execution time: this is the time at which the last exit task of the workflow completes its execution. With real-time workflows, the objective is similar: one needs to match the deadline, i.e., to guarantee that the makespan does not exceed the deadline bound. Because the problem is already NP-complete with two processors, independent tasks (no dependence) and no communication cost[1], list-scheduling heuristics have been introduced; the main idea is to sort the tasks according to their bottom levels, and to greedily map them onto the first available processor when they become ready; here the bottom level of a task is the length of the longest path from that task to an exit task in the dependence graph of the workflow. The bottom level was originally computed without taking communication costs into account. However, for many scientific workflows, communication costs cannot be neglected, which has motivated the introduction of HEFT [7], a list-scheduling heuristic that computes bottom levels by (pessimistically) including all communication costs in the dependence paths. HEFT is able to deal with heterogeneous platforms by averaging task weights and communication costs, and it has become a reference approach since its inception 20 years ago.

The advent of green computing has motivated to revisit the scheduling of scientific workflows beyond pure time constraints, taking energy consumption into account. Modern processors are equipped with Dynamic Voltage and Frequency Scaling (DVFS) and can run at different frequencies. Running at a lower frequency decreases the dynamic energy that is consumed during the computation but also increases execution time, as well as the consumed static energy (these power components are detailed in Section 2.5). This leads to achieving complicated trade-offs. In addition, it has been widely proved that DVFS has negative effects on transient failure rates. As a result, it is important to take reliability degradation into consideration while managing energy by DVFS. Transient errors [8] are usually manifested as bit-flips and due to radiation, minimum voltage, or thermal cycling, etc. They cause a task execution to fail but without completely losing the data present in the processor memory, which makes them harder to detect than permanent faults. Several replicas of the same task instance must be executed in order to guarantee

[1] An easy reduction from 2-PARTITION [5]. See also [6] for more details on the complexity of multiprocessor scheduling.

a prescribed level of reliability. In fact, reliability has long been the main objective of real-time systems, because these systems deal with critical applications where erroneous or incomplete results could lead to catastrophic scenarios [9]. There are several approaches to enforce a prescribed reliability threshold. Re-execution after a failed execution attempt is one possibility, but it induces additional delays. Hence, the standard approach is to use *replication*: for each task, the execution of several replicas is scheduled, as many as needed to meet the reliability criterion. When DVFS is available, the problem gets more difficult, because one can choose to use, say, either two replicas at a high frequency, or three replicas at a lower one. Moreover, different frequencies can be assigned to different replicas of the same task. To further complicate matters, it is known that the transient error rate increases when scaling down frequency using DVFS (see [10] and the discussion in Section 2.3), so saving energy might lead to more errors. Similarly, the introduction of replicas, while mandatory to guarantee a reliability threshold, is also increasing energy consumption. Clearly, in order to limit this additional consumption, it is always beneficial to kill the other replicas of a task instance after one replica has succeeded; but several replicas may well have been executing concurrently until one of them succeeded, and the energy spent before interrupting them is wasted.

Altogether, the problem of mapping and scheduling real-time workflows on a parallel platform whose processors are equipped with DVFS, has become a challenging tri-criteria optimization problem: energy, deadline (or makespan bound) and reliability are conflicting objectives, and sophisticated algorithms must be designed. To the best of our knowledge, only two recent papers have tackled this problem [11, 12], albeit with additional restrictions (see Section 4.1 for details). We present a novel approach that outperforms all previously published solutions for an extended set of simulation scenarios. Given a deadline constraint and a reliability threshold, we first decide for each task how many replicas to use, and at which frequency to execute them. Next we use a new mapping heuristic based upon a notion of layers in the dependence graph; we map several (ready) tasks simultaneously rather than only the highest priority task; the motivation of using layers is to decrease the overlap between all the replicas of a given task, thereby providing more opportunities to save energy. Finally, we dynamically update the schedule to further reduce replica overlaps without violating deadline and reliability constraints.

The three main contributions of the paper are the following:

1. The formulation of the tri-criteria optimization problem, i.e., minimizing energy cost while ensuring deadline and reliability constraints, and the design of several mapping and scheduling heuristics to solve this problem without any restriction; to the best of our knowledge, this work is the first that can accommodate more than one or two replicas per task, hence that can deal with arbitrary reliability thresholds;

2. A set of heuristics which aggressively attempt to minimize the overlap between the replicas of the same task instance, relying upon the notion of task layers, upon the distinction between primary and secondary replicas, and upon the reclaim of the potential slack in the schedule;

3. An experimental evaluation based on a comprehensive set of simulation scenarios, showing that our best heuristics achieve a significant energy gain compared to the state-of-the-art methods 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. The mapping and scheduling heuristics are described in

Section 3. Section 4 is devoted to a comprehensive experimental comparison of the heuristics. Section 5 presents related work. Finally, Section 6 gives concluding remarks and hints for future work.

Table 1: Key Notations

| Notation | Explanation |
|---|---|
| $n$, $M$ | number of tasks and of processors |
| $T_i$ | $i$-th task of application $G$ |
| $w_i$ | WCET for task $T_i$ under $f_{max}$ |
| $w_{i,f_j}$ | WCET for task $T_i$ at frequency $f_j$ |
| $c_{i,j}$ | communication cost from $T_i$ to $T_j$ |
| $succ(T_i)$ | set of successor tasks of $T_i$ in $G$ |
| $prec(T_i)$ | set of tasks preceding $T_i$ in $G$ |
| $\mathcal{D}(G)$ | deadline for application $G$ |
| $\mathcal{D}(T_i)$ | deadline for task $T_i$ |
| $\mathcal{P}$ | set of processors |
| $rep\_num(T_i)$ | actual total number of replicas for $T_i$ |
| $r_i^\alpha$ | $\alpha$-th replica of $T_i$ |
| $f_j$ | the $j$-th available frequency |
| $f(r_i^\alpha)$ | frequency of replica $r_i^\alpha$ |
| $\mathcal{R}(G)$ | reliability threshold for application $G$ |
| $\mathcal{R}(T_i)$ | reliability threshold for $T_i$ |
| $R(r_i^\alpha, f_j)$ | reliability of $r_i^\alpha$ under $f_j$ |
| $R_i(f_j)$ | reliability of $T_i$ under $f_j$ |
| $k_{i,f_j}$ | minimal number of replicas required to run $T_i$ with $f(r_i^1) = f_j$ while meeting $\mathcal{R}(T_i)$ |
| $ST(r_i^\alpha)$ | start time of $r_i^\alpha$ |
| $FT(r_i^\alpha)$ | completion time of $r_i^\alpha$ |
| $FT(T_i)$ | completion time of $T_i$ |
| $FT(G)$ | completion time of application $G$ |
| $E(T_i, f_j, k_{i,f_j})$ | energy cost for $T_i$ with $k_{i,f_j}$ replicas under $f_j$ |

## 2. Model

The inputs to the optimization problem are a workflow (i.e., a task graph), a set of homogeneous processors, a reliability target and a global deadline. Key notations are summarized in Table 1. Our new scheduling algorithms apply to a very general framework, which we describe in this section. A particular instantiation of the model for the simulations is outlined in Section 4.2.

### *2.1. Application*

The workflow application is modeled as a task graph $G$, where $G = (V, E)$ is a Directed Acyclic Graph (DAG) composed of a set of $n$ tasks $V = \{T_1, T_2, ..., T_n\}$. An edge from task $T_i$

to task $T_j$ in set $E$ represents a task dependency. Each task $T_i$ has a weight $w_i$, which denotes its worst-case execution time (WCET) under the maximum available frequency $f_{max}$. Task instances (also called *jobs*) typically complete their execution earlier than their estimated WCETs: execution times are assumed to be data-dependent and randomly sampled from some probability distribution whose support is upper bounded by the WCET [13]. Note that without a WCET, i.e., with probability distributions that can take arbitrarily large values, it is not possible to guarantee any deadline. Each edge between two tasks $T_i$ and $T_j$ is also weighted by the size $c_{i,j}$ of the output data produced by $T_i$ that is needed as input to $T_j$. $prec(T_i)$ and $succ(T_i)$ represent the set of direct predecessors and successors of task $T_i$ respectively. A task $T_i$ can only start its execution when all of its direct predecessors have finished their execution and all intermediate data has been transferred.

### *2.2. Processors*

The platform consists of $M$ homogeneous processors, with the same set $F$ of frequencies ranging from $f_{min}$ to $f_{max}$. Without loss of generality, we normalize the frequencies to enforce $f_{max} = 1$. The interplay between task execution time and frequency is complicated [14]. To be fully general, we consider arbitrary WCETs that decrease when frequency increases. Specifically, for task $T_i$ at frequency $f_j$, the WCET is $w_{i,f_j}$, and we assume that $w_{i,f_j} > w_{i,f_k}$ if $f_j < f_k$. Then the actual execution time of a task instance of task $T_i$ at frequency $f_j$ is drawn from a distribution whose support is upper bounded by $w_{i,f_j}$. For the simulations in Section 4, we use four different set of frequencies taken from actual microprocessor values, we consider Amdahl speedup functions for the WCETs and we draw actual execution times from Uniform and Truncated Normal distributions.

### *2.3. Error model*

We consider transient errors [8], modeled by an Exponential distribution whose average arrival rate depends upon the frequency used. The interplay between error rate and frequency is complicated too. In fact, the impact of DVFS on the error rate is discussed extensively in the literature. On the one hand, lowering the voltage/frequency is believed to have an adverse effect on the system reliability [15, 16, 17, 18, 19]: this is because decreasing the nominal voltage will increase the vulnerability to longer delays in rippling logic along computational paths, hence lowering the circuit's critical charge (i.e., the minimum charge required to cause an error in the circuit). On the other hand, increasing the voltage beyond the nominal value (overclocking) will also increase the error rate due to overheating [20, 21, 22, 23]. In the general framework, we deal with frequency values in the range $[f_{min}, f_{max}]$, and can safely assume that the lower the frequency, the higher the error rate. In other words, $f_{max}$ is the nominal frequency recommended by the chip manufacturer, and we avoid overclocking to save energy. We let $\lambda(f_j)$ be the instantaneous error rate at frequency $f_j$ and assume that $\lambda(f_j) > \lambda(f_k)$ if $f_j < f_k$. For the simulations in Section 4, we instantiate the values of the $\lambda(f_j)$ from the formula used in [15, 16, 18, 19].

At the end of the execution of a task instance, there is an *acceptance test* [24] to check the occurrence of soft errors induced by the transient errors. If the execution result of a task instance satisfies its predefined acceptance criteria, then it is considered to be successfully executed [25, 26]. As in all relevant literature, it is assumed that acceptance tests are 100% accurate[2], and that the duration of the test is included within the task WCET (see [27] and references

[2]One could account for the possibility of the acceptance test to fail by lowering the reliability of a replica by some amount, but this goes beyond the standard model.

therein). The *reliability* of a task instance is the probability of executing it successfully, in the presence of errors. Multiple replicas of a task instance will be executed to mitigate the impact of transient errors. Note that two replicas of a given task instance will have the same execution time if run at the same frequency, because they operate on the same data. Because we do not know the actual execution time of tasks before they complete, to estimate the reliability of any task instance replica we conservatively assume that its actual execution time is equal to the task WCET. Therefore, a replica of an instance of task $T_i$ running at frequency $f_j$ has reliability $\exp^{-\lambda(f_j)w_{i,f_j}}$.

### *2.4. Reliability threshold and deadline*

There are two constraints for the workflow $G$ in the optimization problem. Firstly, $\mathcal{D}(G)$ represents the global deadline: the execution of exit tasks must be finished before $\mathcal{D}(G)$. Let $FT(T_i)$ denote the finish time of the task $T_i$. Then the total execution time of the application is $FT(G) = \max_{T_i \in V} FT(T_i)$. If $FT(G) \leq \mathcal{D}(G)$, the deadline is met. The calculation of $FT(T_i)$ will be detailed in Section 3.

Secondly, $\mathcal{R}(G)$ denotes the reliability threshold. Given that the reliability of the application is the product of the reliability of each task, we have $\mathcal{R}(G) = \Pi_{i=1}^{n} \mathcal{R}(T_i)$, where $\mathcal{R}(T_i)$ is the reliability threshold of task $T_i$. In some settings, the set of all $\mathcal{R}(T_i)$ values is given as input, while in some other settings these values are computed from $\mathcal{R}(G)$. Now, given the reliability threshold $\mathcal{R}(T_i)$ for each task $T_i$, the question is to determine the number of replicas to use, and at which frequency to execute them, so that the threshold $\mathcal{R}(T_i)$ is met. Following the literature again, we assume that the first replica of $T_i$ runs at some frequency $f_j$ to be determined by the mapping and scheduling process, and that all the other replicas run at frequency $f_{max}$. This simplifying assumption is not needed for our approach but it avoids the exponential cost of searching for all possible combinations. To fix notations, let $k_{i,f_j}$ be the total number of replicas needed for task $T_i$ when the first replica operates at frequency $f_j$, and let $f(i, j, \alpha)$ be the frequency of replica number $\alpha$ when the first replica runs at $f_j$, so that $f(i, j, 1) = f_j$ and $f(i, j, \alpha) = f_{max}$ for $2 \leq \alpha \leq k_{i,f_j}$. The reliability of replica number $\alpha$ of task $T_i$ is then $R(r_i^\alpha, f_j) = \exp^{-\lambda(f(i,j,\alpha))w_{i,f(i,j,\alpha)}}$.

Since a task instance fails only if all its replicas fail, we achieve a reliability $R_i(f_j)$ for task $T_i$ where

$$R_i(f_j) = 1 - \Pi_{\alpha=1}^{k_{i,j}} (1 - R(r_i^\alpha, f_j)) \tag{1}$$

and we need to enforce that $R_i(f_j) \geq \mathcal{R}(T_i)$. In the scheduling and mapping phase, we let $f(r_i^\alpha)$ denote the frequency actually used for replica number $\alpha$ of task $T_i$.

### *2.5. Energy model*

The energy cost of a replica of task $T_i$ at frequency $f_j$ is estimated as the power times its worst-case execution time $w_{i,f_j}$, which is an upper bound for the actual execution. As for the power $P(f_j)$ at frequency $f_j$, we use

$$P(f_j) = P_{static} + P_{dyn}(f_j) = P_{static} + (P_{indep} + C \times f_j^3)$$

where $P_{static}$ (the static power), $P_{indep}$ (the frequency independent part of dynamic power) and $C$ (the effective switching capacitance) are system-dependent constants. The energy cost $E(T_i, f_j, 1)$ for the first replica of task $T_i$ at $f_j$ is then: $E(T_i, f_j, 1) = P(f_j) \times w_{i,f_j}$. The final energy cost with $k_{i,f_j}$ replicas is estimated as:

$$E(T_i, f_j, k_{i,f_j}) = E(T_i, f_j, 1) + (1 - R(r_i^1, f_j)) \times \sum_{2}^{k_{i,f_j}} E(T_i, f_{max}, 1) \tag{2}$$

In Equation (2), we assume that there is no overlap between the primary replica (the first replica starting its execution) and the other replicas of a task, while secondary replicas fully overlap. Thus, the estimated energy consumption for a task is the energy consumed by the primary replica plus (in case the primary fails) the energy consumed by all the secondary replicas. Furthermore, each processor always consumes static power when idle (this consumption can be eliminated only by a complete shutdown). Hence, we account for static power whenever the scheduling strategies described below keep some processors idle.

### *2.6. Optimization Objective*

The objective is to determine a set of replicas for each task and their execution frequencies, and to build a static schedule, where the replicas of each task are mapped onto the processors, so that the expected energy consumption is minimized. The constraints are: enforcing all dependencies in the task graph, matching the workflow deadline $\mathcal{D}(G)$, and meeting the reliability threshold $\mathcal{R}(T_i)$ for each task $T_i$. As already mentioned in the introduction, 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 as described in Equation (2); more precisely, we use WCETs instead of (yet unknown) actual execution times, and we estimate the energy of a task as the sum of the energy of its primary replica plus all its secondary replicas in case the primary replica failed.
- To further complicate matters, the static schedule is dynamically modified on the fly to take into account the actual execution times rather than the WCETs. Also, as soon as one replica of a given task instance completes its execution successfully, all the other replicas of that task instance become redundant and are terminated instantaneously. The scheduler receives a signal from the successful replica and is responsible for both interrupting the other replicas that are currently executing and discarding any additional replica that has not started yet [27].

## 3. Scheduling heuristics

In this section, we describe the heuristics we introduce to solve the tri-criteria optimization problem. We start by outlining some design guidelines, namely, how to prioritize tasks and compute a deadline for each of them (Section 3.1), how to set frequencies to task replicas (Section 3.2), how to map tasks (Section 3.3) and how to partition them into layers (Section 3.4). Then we introduce our basic static scheduling algorithm in Section 3.5, followed by several energy-saving heuristics using DVFS in Section 3.6. Finally the runtime optimization is described in Section 3.7.

### *3.1. Task deadlines and ordering*

From the workflow deadline $\mathcal{D}(G)$ we derive a deadline $\mathcal{D}(T_i)$ for each task $T_i$. When deriving task deadlines, we conservatively assume that all communications take place (which is

similar to HEFT assumptions [7]). Consequently, if each task satisfies its deadline, the schedule is valid. We then have:

$$\mathcal{D}(T_i) = \begin{cases} \mathcal{D}(G) \text{ if } succ(T_i) = \emptyset \\ \min\limits_{T_j \in succ(T_i)} \mathcal{D}(T_j) - w_j - c_{i,j} \text{ otherwise} \end{cases} \quad (3)$$

List-based schedules require tasks to be prioritized. We order tasks by non-increasing bottom-levels. The bottom-level $bl(T_i)$ of a task $T_i$ is the length of a critical path starting from $T_i$ if all communications take place:

$$bl(T_i) = w_i + \max_{T_j \in succ(T_i)} (c_{i,j} + bl(T_j)) \quad (4)$$

### *3.2. Replicas*

For saving energy, we use DVFS and decrease the frequency of replicas. We decide that only the frequency of primary replicas may be modified (and, thus, may be lower than $f_{max}$), while all secondary replicas are executed at $f_{max}$. Running all secondary replicas at $f_{max}$ decreases the pressure on the schedule. This assumption should not significantly impact the energy consumption because we expect that most secondary replicas will not be executed in actual failure-free execution scenarios.

As we decrease the frequency of a primary replica, we may need to add an additional secondary replica to satisfy the reliability threshold. However, even if we decrease the frequency of the primary replica from $f_{max}$ to $f_{min}$, at most one additional replica will be required. Indeed, let us assume that we need $r$ replicas to satisfy the reliability threshold for task $T_i$ when all replicas (including the primary) are executed at $f_{max}$. Then a primary replica at $f_{min}$ and $r$ secondary replicas at $f_{max}$ satisfy the reliability threshold as $\mathcal{R}(T_i) \leq 1 - \Pi_{\alpha=1}^{r}(1 - R(r_i^{\alpha}, f_{max})) < 1 - (1 - R(r_i^{1}, f_{min})) \times \Pi_{\alpha=1}^{r}(1 - R(r_i^{\alpha}, f_{max}))$.

### *3.3. Task Mapping*

To schedule replicas we follow an earliest starting time policy, i.e., we map each replica on a processor that will start its execution at the earliest (breaking ties arbitrarily). In the static scheduling phase, each task should wait for the completion of all the replicas of all of its predecessors before starting its execution. This is because we do not know which replicas of the predecessors will be successful, and we must consider the worst case.

We now describe how to compute the start time $ST$ and the finish time $FT$ of each replica. Let $EST(r, p)$ denote the earliest time at which the replica $r$ could start on processor $p$, and let $P(r)$ denote the processor that will execute $r$. By definition of the earliest starting time policy, we have

$$P(r) = \arg\min_{p \in \mathcal{P}} EST(r, p) \quad (5)$$

and

$$ST(r) = EST(r, P(r)), \quad (6)$$

where

$$EST(r_i^{\alpha}, p) = \max \left\{ \begin{array}{c} ready(p), \\ \max\limits_{\substack{T_j \in prec(T_i) \\ \beta \in [1, rep\_num(T_j)]}} FT(r_j^{\beta}) + c(r_i^{\alpha}, r_j^{\beta}) \end{array} \right\}$$

with

$$c(r_i^\alpha, r_j^\beta) = \begin{cases} 0 \text{ if } P(r_i^\alpha) = P(r_j^\beta) \\ c_{j,i} \text{ otherwise} \end{cases}$$

Here $ready(p)$ denotes the time at which processor $p$ is available, according to the mapping and scheduling decisions previously taken. $rep\_num(T)$ denotes the number of replicas for task $T$. We will later detail its computation.

We map each replica while ensuring that no two replicas of the same task are assigned on the same processor. This is implicit in Equation (5): the arg min is computed over all processors on which a replica of the same task as $r$ has not already been mapped. Finally, we derive the completion time of each replica as

$$FT(r_i^\alpha) = ST(r_i^\alpha) + w_{i,f(r_i^\alpha)} \tag{7}$$

### *3.4. Task layers*

Tasks are partitioned into *layers* and we will use a layer by layer approach to map and schedule replicas. A task layer is a set of tasks which are mutually independent. Because there is no dependence between any two tasks belonging to the same layer, we can schedule all their replicas in any order. For any task $T$, $L(T)$ is the integer index denoting its layer. The layer index of exit tasks is equal to 1, and layers are numbered in reverse topological order. Formally:

$$L(T_i) = \begin{cases} 1 + \max\limits_{T_j \in succ(T_i)} L(T_j) \text{ if } succ(T_i) \neq \emptyset, \\ 1 \text{ if } succ(T_i) = \emptyset \end{cases} \tag{8}$$

With the layer by layer approach, we first map one replica of each task in the layer $L$ considered, and we tag them as the *primary* replicas of the tasks in $L$. All (possibly) remaining replicas for the tasks in $L$ will be *secondary* replicas. We then map a first secondary replica of each task in $L$, then a second secondary replica, and so on. The aim of this approach is to reduce the overlapping of replicas of the same task.

### *3.5. The* BasicLayer *algorithm*

Initially, the task graph $G$ is scheduled with the BasicLayer algorithm (Algorithm 1). The aim of BasicLayer is only to build a valid schedule, which will later be optimized. Therefore, BasicLayer does not use DVFS, which means all tasks are mapped with frequency $f_{max}$. BasicLayer first computes the bottom-level, deadline, and layer index of each task (Lines 2 to 4). Then BasicLayer sorts tasks in list $TaskList$, which includes all tasks in the task graph $G$ sorted by non-decreasing bottom-level (Line 5). Then it builds a set of tasks that are in the same layer (Lines 7 to 10). We get the layer identifier from the top task of *TaskList* (Line 7), then move the top task and other tasks with the same layer identifier to *TasksInLayer* (Lines 8 to 10). In line 11, BasicLayer tries to map tasks in this set first using a layer by layer approach (Algorithm 2). If Algorithm 2 fails, BasicLayer maps those tasks using a task by task approach (Algorithm 3 called on line 13). If Algorithm 3 succeeds, BasicLayer moves to the scheduling of the next layer. Otherwise, when even the task-by-task approach failed, the whole schedule is cleared (Line 15), and the whole graph is rescheduled using only the task-by-task approach (Line 16). If this last attempt also fails, we report a failure of the BasicLayer algorithm, meaning that we are not able to find a feasible schedule of $G$.

Algorithm 2 is an algorithm to map and schedule the replicas of a given layer in a layer-by-layer approach. In Line 2, we first compute the number of replicas for each task in *TasksInLayer*

**Algorithm 1:** The BasicLayer algorithm

**Input:** A graph $G = (V, E)$, the set $F$ of possible frequencies, the reliability threshold $\mathcal{R}(T_i)$ for each task $T_i$, the deadline $\mathcal{D}(G)$

```
for T_i ∈ V do
    Compute bl(T_i) using Equation (4)
    Compute D(T_i) using Equation (3)
    Compute L(T_i) using Equation (8)
Populate a list TaskList containing all tasks sorted by non-increasing bottom-level
while TaskList ≠ ∅ do
    LayerIdentifier ← L(Head(TaskList))
    TasksInLayer ← {RemoveHead(TaskList)}
    while L(Head(TaskList)) == LayerIdentifier do
        TasksInLayer ← TasksInLayer ∪ {RemoveHead(TaskList)}
    map the tasks in TasksInLayer using Algorithm 2
    if not feasible then
        map the tasks in TasksInLayer using Algorithm 3
        if not feasible then
            clear the whole schedule
            reschedule application G, while only using Algorithm 3 to map each task
            if not feasible then
                return failure to build a schedule
            else
                return schedule
return schedule
```

**Algorithm 2:** Replica mapping (layer by layer)

**Input:** The set of tasks *TasksInLayer*, the reliability threshold $\mathcal{R}(T_i)$ and the deadline $\mathcal{D}(T_i)$ for each task $T_i$ in *TasksInLayer*, the schedule built so far

1 **for** $T_i \in TasksInLayer$ **do**
2 $\quad rep\_num(T_i) \leftarrow k_{i,f_{max}}$
3 $\quad$ Compute $ST(r_i^1)$ using Equation (6)
4 $max\_rep\_num \leftarrow \max_{T_i \in TasksInLayer} rep\_num(T_i)$
5 **for** $k = 2$ *to* $max\_rep\_num$ **do**
6 $\quad$ **for** $T_i \in TasksInLayer$ **do**
7 $\quad\quad$ **if** $rep\_num(T_i) \leq k$ **then** Compute $ST(r_i^k)$ using Equation (6)
8 **for** $T_i \in TasksInLayer$ **do**
9 $\quad FT(T_i) \leftarrow \max_{1 \leq k \leq rep\_num(T_i)} FT(r_i^k)$
10 $\quad$ **if** $FT(T_i) > \mathcal{D}(T_i)$ **then** return *not feasible*
11 return the schedule built so far

**Algorithm 3:** Replica mapping (task by task)

**Input:** The set of tasks *TasksInLayer*, the reliability threshold $\mathcal{R}(T_i)$ and the deadline $\mathcal{D}(T_i)$ for each task $T_i$ in *TasksInLayer*, the schedule built so far

1 **for** $T_i \in TasksInLayer$ **do**
2 $\quad rep\_num(T_i) \leftarrow k_{i,f_{max}}$
3 $\quad$ **for** $1 \leq k \leq rep\_num(T_i)$ **do** Compute $ST(r_i^k)$ using Equation (6)
4 $\quad FT(T_i) \leftarrow \max_{1 \leq k \leq rep\_num(T_i)} FT(r_i^k)$
5 $\quad$ **if** $FT(T_i) > \mathcal{D}(T_i)$ **then** return *not feasible*
6 return the schedule built so far

under the assumption that the frequency of each primary replica is $f_{max}$ (recall that secondary replicas are always executed at $f_{max}$). Then we map the primary replica of each task (Line 3). Note that we use $r_i^1$ to denote the primary replica of task $T_i$, and $r_i^\alpha$ to represent any secondary replica (when $\alpha > 1$). After mapping all primary replicas in the layer, we map all secondary replicas in a layer-by-layer manner, which means we map a first secondary replica for each task, then a second secondary replica, and so on (Lines 5 to 7). Finally, we compute the completion time of each task (Line 9) and check whether any task exceeds its deadline (Line 10). If this is the case, we return *not feasible*, otherwise we return the partial schedule built so far.

For the task-by-task mapping of replicas (Algorithm 3), we iteratively schedule each task in *TasksInLayer* according to the task priority order. For any given task, we first compute its number of replicas with the same assumption as for Algorithm 2 (Line 2). Then we map its primary replica and each of its secondary replicas (Line 3). Finally, we compute its completion time (Line 4), and check whether it exceeds its deadline (Line 5). If this is the case, we return *not feasible*, otherwise, we return the partial schedule built so far.

*3.6. Energy-saving heuristics*

Recall that BasicLayer does not use DVFS to save energy. Therefore, after successfully building an initial solution where all replicas are run at $f_{max}$, we use some heuristics to lower the frequency of primary replicas in order to save energy. However, as already stated in Section 3.2, when the frequency of a primary replica decreases, we may need to add one additional replica. Adding some extra replicas may prevent the workflow to be scheduled under the deadline and reliability constraints. Hence, it is very important to choose which tasks can have an additional replica, and which cannot. We propose several heuristics to choose which tasks will be granted an additional replica while ensuring the application can still be scheduled:

- **MinRepNumber**: No task is allowed to have an additional replica (this is the baseline).
- **TaskSize**: In each layer, tasks are sorted by non-increasing WCETs. This is because tasks with longer execution time consume more energy. Higher priority will give more freedom to those tasks to decrease their frequencies as low as possible. Following this order, we consider the tasks one by one. Each task can have an additional replica provided that a valid schedule can still be built.
- **LayerSize**: We define the WCET of a layer as the sum of the WCETs of its constituting tasks. LayerSize sorts layers by non-increasing WCETs and processes them in this order. When processing a layer $L$, it checks whether all tasks in $L$ can be granted an additional replica. If so, one replica is added for each task in $L$, otherwise none of the tasks in $L$ is allowed an additional replica.
- **TopoLayerSize**: This heuristic works as LayerSize except that after processing a layer $L$, it goes directly to processing the next layer in the sorted list whose layer index is larger than that of $L$. In other words, LayerSize does not process all layers but only a topologically ordered subset of them.

The next heuristic proceeds differently: rather than starting with BasicLayer, it tries to use DVFS from the start in an aggressive way:

- **OptFrequency**: For each layer, OptFrequency maps and schedules each primary replica using the frequency minimizing the energy consumption as expressed by Equation (2). If the global deadline is not satisfied, the solution of BasicLayer is used as is.

After deciding which tasks can have an additional replica, we use Algorithm 4 to adjust the frequency of primary replicas and the scheduling of all replicas. The aim of the latter type of modification is to reduce the overlap between the execution of a primary replica and that of the secondary replicas of the same task. In the BasicLayer algorithm, we map each replica as early as possible. Hence, it may contain some slack with respect to deadlines at the end of the schedule. We use this slack to postpone the execution of some replicas or to adjust the frequency of some primary replicas. Firstly, we consider the tasks in an order reverse from that of BasicLayer (Line 1), which means considering tasks starting from the end of the schedule. Then, we build at Line 5 the layer containing the first task, exactly as we did in Lines 7–10 of Algorithm 1. We delay the start of each secondary replica as much as possible. For a secondary replica (Lines 7 to 14), we compute its latest possible completion time according to its deadline and to the earliest start time of its successors (Line 8). We then optimize the schedule to minimize the overlap between the different secondary replicas of the same task (Line 9), while forbidding any of them

**Algorithm 4:** Schedule optimization

**Input:** The base schedule S, the Graph G = (V,E)

1 Sort the tasks in a list *TaskList* by non-decreasing bottom-levels
2 $has\_changed \leftarrow true$
3 **while** *has_changed* **do**
4 $\quad has\_changed \leftarrow false$
5 $\quad TasksInLayer \leftarrow ExtractFirstLayer(TaskList)$
6 $\quad$ **for** $T_i \in TasksInLayer$ **do**
7 $\quad\quad$ **for** $2 \leq k \leq rep\_num(T_i)$ **do**
8 $\quad\quad\quad newFT(r_i^k) \leftarrow \min\left\{\mathcal{D}(T_i), \min_{T_j \in succ(T_i)} ST(T_j) - c_{i,j}\right\}$
9 $\quad\quad$ Minimize the overlap between replicas of $T_i$
10 $\quad\quad$ **for** $2 \leq k \leq rep\_num(T_i)$ **do**
11 $\quad\quad\quad$ **if** $newFT(r_i^k) > FT(r_i^k)$ **then**
12 $\quad\quad\quad\quad FT(r_i^k) \leftarrow newFT(r_i^k)$
13 $\quad\quad\quad\quad has\_changed \leftarrow true$
14 $\quad\quad\quad\quad ST(r_i^k) \leftarrow FT(r_i^k) - w_{i,f_{max}}$
15 $\quad$ **for** $T_i \in TasksInLayer$ **do**
16 $\quad\quad newFT(r_i^1) \leftarrow \min\left\{\mathcal{D}(T_i), \min_{2 \leq k \leq rep\_num(T_i)} ST(r_i^k)\right\}$
17 $\quad\quad$ **if** $FT(r_i^1) < newFT(r_i^1)$ **then**
18 $\quad\quad\quad$ decrease $r_i^1$ frequency as much as possible while satisfying $FT(r_i^1) \leq newFT(r_i^1)$ and $k_{i,f(r_i^1)} \leq rep\_num(T_i)$
19 $\quad\quad\quad FT(r_i^1) \leftarrow newFT(r_i^1)$
20 $\quad\quad\quad ST(r_i^1) \leftarrow FT(r_i^1) - w_{i,f(r_i^1)}$
21 $\quad\quad\quad has\_changed \leftarrow true$
22 $\quad\quad$ **if** $rep\_num(T_i) > k_{i,f(r_i^1)}$ **then** delete one secondary replica of $T_i$

to start earlier than in the original schedule. Then we delay the start times of secondary replicas according to the new computed completion times (Lines 11–14). To be specific, we compare the new completion times with original ones (Line 11): set completion time to the larger value (Line 12), and if this value is the new computed one, then set signal *has_changed* (Line 13) and delay the start time of secondary replicas (Line 14). For a primary replica, we also compute its latest possible completion time (Line 16). Then we try to reduce its frequency as much as possible as allowed by the available number of replicas (Line 18). Then we delay its start time according to the new computed completion time and frequency (Lines 19 and 20). Note that some tasks have already been granted an additional replica by some of the heuristics described above. Hence, these tasks can decrease the frequency of their primary replica to $f_{min}$ (the slack permitting). If a task $T_i$ has received an additional replica, but the frequency of its primary replica frequency does not require one, we delete the additional replica (Line 22). We repeat the whole process as long as we can improve the schedule.

We now detail the complexity of our approach. Recall that $n$ is the number of tasks, and

$M = |\mathcal{P}|$ is the number of processors. In Algorithm 1, Line 5 sorts the $n$ tasks in time $O(n \log(n))$; the maximum number of iterations of the loop Line 6 is $n$, and each task has at most $M$ replicas, hence the complexity of the loop is $O(nM)$. Thus the time complexity of Algorithm 1 is $O(n \max(n, M))$. In Algorithm 4, each task has an additional replica in the worst case, hence the number of iterations for the loops of Lines 6 and 15 is at most $n$; the number of iterations of the loop in Line 7 is at most $M$; then in, for each of the $O(nM)$ iterations overall, only one additional replica is deleted in the worst case, so we are going to repeat this big loop at most $n$ times. Therefore the time complexity of Algorithm 4 is $O(n^2 M)$. This low complexity shows that our approach has a great potential for dealing with a large number of tasks (see also reports on actual execution times in Section 4.4).

### *3.7. Runtime optimization*

In all executions of the heuristics, the successful execution of a replica leads to the immediate cancellation of all other replicas of the same task instance. This is a crucial source of energy saving. Furthermore, the actual execution times of replicas are usually shorter than their WCETs. Therefore, when one replica completes, we have some additional slack to adjust the schedule of other (unfinished) replicas, in order to save energy.

In the actual execution, when one replica completes, we adjust the scheduling of the first unfinished replica of each processor. For each unfinished replica, if there exists another replica of the same task instance that has successfully completed, it is immediately canceled. Otherwise, this replica is processed as early as possible, while satisfying precedence constraints and while not increasing its overlap with other unfinished replicas of the same task instance.

## 4. Performance evaluation

In this section, we present simulation results to evaluate the performance of our strategies compared to two state-of-the-art competitor approaches, OEA and MILP. In Section 4.1, we introduce both competitors and explain how we have extended them; we also present a baseline approach, QFEC, which we use for reference. In Section 4.2, we describe the parameters and settings used for the experimental campaign and in Section 4.3 we discuss the limitations of MILP. Finally, we present the results in Section 4.4.

### *4.1. Comparing Methods*

We compare our five heuristics (MinRepNumber, TaskSize, LayerSize, TopoLayerSize, and OptFrequency) with one baseline reference, QFEC [28], and two state-of-the-art approaches, OEA [11] and MILP [12].

*Baseline* QFEC. Some related heuristics have been designed in the book [28]. Chapter 2 of [28] introduces heuristics for the bi-criteria problem (energy, makespan) while chapter 3 deals with another bi-criteria problem (reliability, makespan). Both chapters target heterogeneous platforms. In this work, we have extended the heuristic QFEC of chapter 3 in [28] to address the tri-criteria problem and used it as a baseline for performance comparison. In our extension of QFEC, all replicas are assigned the highest frequency, and their number is computed so as to match the reliability threshold. The scheduling and mapping obey the HEFT priority and assignment rules [7]. This QFEC extension is used as a reference to report the energy savings of the other heuristics.

OEA. The OEA heuristic [11] minimizes the energy consumption while satisfying the reliability requirement, but only considers low reliability thresholds for which additional replicas are not needed: using only one instance of each task is assumed to be enough to match the reliability threshold. The general OEA heuristic targets heterogeneous platforms; for homogeneous platforms, the same frequency is assigned to all tasks. A major difference is that the mapping of tasks to processors is assumed to be given in OEA; computing a mapping is achieved by the ODS heuristic [11], which differs from HEFT in that priority is given to tasks with high out-degree. The intuition is that the scheduling of a task is more urgent if it has many successors. We have extended OEA with replicas to match arbitrary reliability thresholds as follows: all primary replicas are assigned the same frequency $f$, and all secondary replicas are assigned $f_{max}$; we iteratively find the smallest primary replica frequency $f$ (and the corresponding replica number) that can meet the reliability and deadline constraints using HEFT to allocate each replica: this leads to the HEFT-OEA heuristic. We have also designed a similar extension to deal with replicas, but keeping the original ODS scheduling instead of using HEFT: this leads to the ODS-OEA heuristic. After these phases, we use Algorithm 4 to decrease the overlap between primary and secondary replicas. Altogether, we have designed two optimized extended version of OEA, one with traditional HEFT scheduling (HEFT-OEA) and one with the ODS priority scheduling (ODS-OEA).

MILP. The MILP heuristic [12] presents a Mixed Integer Linear Program (MILP) to deal with the same tri-criteria problem as in this paper (in the variant which considers task-level DVFS giving rise to the RAFTM-TL linear program; the other variants being less general). Hence, this is the closest related work to our approach. However, MILP does not take communication costs into account and is limited by design to have at most two replicas per task; hence, it cannot match high reliability thresholds that require three or more replicas.

We first comment on the objective function of MILP. Then we show how to modify MILP to obtain a new version, called EXTMILP, which does not have its limitations. The objective of MILP is to minimize the energy consumption. The formula used to estimate the energy consumption (Equation (10) of [12]) assumes that if a task has two replicas, then both are fully executed. This is a conservative, and thus pessimistic, assumption: if the primary replica completes successfully, and the two replicas do not overlap, the secondary replica does not need to be launched and its energy consumption is then null. Our own objective function in Equation (2) accounts for the possibility that a primary replica completes successfully, and is optimistic since it assumes that the secondary replicas of a task do not overlap with the primary replica. However, we fix the frequency of secondary replicas at $f_{max}$ (to simplify the problem) while MILP is free to chose any frequency.

To take communications into account in MILP, we need to modify Equation (8) of [12] which enforces task dependencies. To the right-hand side of this equation we add the term

$$c_{i,j}(1 - w_{ij} - w_{ji}).$$

where $w_{ij}$ is a binary variable equal to 1 if and only if the $i$-th task is executed before the $j$-th task and on the same processor. Indeed, if tasks $i$ and $j$ are both mapped on the same processor no communication cost should be paid and exactly one of the two binary variables $w_{ij}$ and $w_{ji}$ is equal to 1 while the other one is 0. Otherwise $w_{ij} = w_{ji} = 0$ and the communication cost must be paid.

We now show how to extend Equation (2) of [12] to enable a number $N$ of secondary replicas for each task. Let $\sigma_{kN+i}$ be the binary variable indicating whether there is a need for $k+1$ replicas

Table 2: Comparison of the capabilities of the different heuristics: maximum allowed number of replicas per task; whether DVFS is used or all replicas are executed at $f_{max}$; whether communication costs are taken into account.

| | MILP | ExtMILP | OEA | Heft-OEA and ODS-OEA | QFEC | Our methods |
|---|---|---|---|---|---|---|
| Number of replicas | $\leq 2$ | arbitrary | 1 | arbitrary | arbitrary | arbitrary |
| DVFS | Yes | Yes | Yes | Yes | No | Yes |
| Communication cost | No | Yes | Yes | Yes | Yes | Yes |

for task $i$ (hence, $k$ secondary replicas). We have the constraint: $\sigma_{(k+1)N+i} \leq \sigma_{kN+i}$, because there is no need for a $(k+1)$-th secondary replica if there is no need for a $k$-th one. Let $R_i^k$ be the reliability obtained using the primary replica and the first $k$ secondary replicas for task $i$. There is a need for a $(k+1)$-th secondary replica only if $k$ secondary replicas do not provide the required level of reliability. Hence, $R_i^{th} > R_i^k$ implies $\sigma_{(k+1)N+i} = 1$. Therefore, the equation generalizing Equation (2) of [12] is:

$$\delta_i - (1+\delta_i)\sigma_{(k+1)N+i} \leq R_i^k - R_i^{th} \leq 1 - \sigma_{(k+1)N+i}.$$

These inequalities are not linear because of the definition of $R_i^k$:

$$R_i^k = (1-\sigma_{kN+i})R_i^{k-1} + \sigma_{kN+i}\left(1 - \prod_{j=0}^{k}(1-r_{jN+i})\right)$$

where $r_{jN+i}$ is the reliability of the $j$-th replica, i.e., using the notations of [12]:

$$r_{jN+i} = \sum_{l \in \mathcal{L}} s_{jN+i} l e^{-\phi_i(f_i)}.$$

Hence, the formula of $R_i^k$ is a polynomial of degree $k+1$ in the binary variables $s_{jN+i}$, which can easily be linearized using classical methods [29].

*Comparison fairness.* We point out that all energy-aware heuristics discussed in this section (Heft-OEA, ODS-OEA, ExtMILP) were originally designed to solve problem instances with low reliability thresholds: there is a single replica per task in OEA, and only two replicas per task in MILP. The heuristics introduced in this work are the first ones that can solve arbitrary instances of the tri-criteria problem. We have extended OEA to deal with replicas and MILP to use an arbitrary number of replicas and to take communication costs into account. We have optimized their runtime executions in the same way as our own heuristics. Table 2 details the different capabilities of the methods. We will see in Section 4.4 that solving ExtMILP with only two replicas is already very slow. Therefore we report results for ExtMILP for low reliability scenarios requiring, most of the times, only two replicas.

### *4.2. Experimental methodology*

We validate our methods on a comprehensive set of parameters that we detail below. In the simulations, we set $M = 8$ cores as default. Interested readers could run experiments with different settings using our discrete simulator which is publicly available online [30].

*Fault rates.* Let $\lambda_0$ denote the fault rate at frequency $f_{max}$. Then the fault rate at frequency $f_i$ is given by the formula used in [15, 16, 18, 19]: $\lambda(f_i) = \lambda_0 \times \exp^{\frac{d(1-f_i)}{1-f_{min}}}$, where $d$ is the sensitivity

factor; $d$ is a measure of how quickly the transient fault rate increases when the system supply voltage and frequency are scaled. Following [27], we assume that the transient error arrival rate at $f_{max}$ is $\lambda_0 = 10^{-6}$ and the system sensitivity factor is $d = 4$.

*Frequency sets.* We have 4 sets of frequencies. Due to space limitation, the plots in Section 4.4 are for a 5-level frequency set which is taken from a real microprocessor [31] and whose (normalized) values are $f_1 = \{1.0, 0.8, 0.6, 0.4, 0.15\}$. We also validate our methods on two other real frequency sets, $f_3 = \{1.0, 0.86, 0.71, 0.57, 0.46, 0.32, 0.21\}$ and $f_4 = \{1.0, 0.844, 0.750, 0.633, 0.562, 0.50, 0.421, 0.375, 0.316, 0.281, 0.250, 0.211\}$ [32, 14], and one synthetic set ranging from 0.1 to 1 by step of 0.1: $f_2 = \{1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1\}$. See [33] for the complete set of experiments.

*WCETs and task execution times.* To be close to real application cases, we let WCETs obey a randomized speedup function similar to Amdahl's law: specifically for task $T_i$ at frequency $f_j$, the WCET is $w_{i,f_j} = s_i w_i + (1 - s_i)\frac{w_i}{f_j}$, where the sequential fraction $s_i$ is drawn uniformly and randomly in the interval $[0.1, 0.3]$. Actual task execution times for each task may have different probability distributions. In the experiments, we consider that the actual execution time is between a best case (BC) and a worst case (WC) execution time and is determined by a normal distribution or by a uniform distribution. The mean and the standard deviation of the normal distribution are set to $(BC + WC)/2$ and $(BC - WC)/6$, respectively, as in [27]. Moreover, we varied the BC/WC ratio from 0.1 to 0.9 by step of 0.1 to model different workload variability settings.

*Power.* The static power and the frequency-independent part of the dynamic power are set to 5% and 15% respectively, of the maximum frequency-dependent power consumption with $C = 1$.

*Workflows.* We consider 12 workflows, including 9 workflows (with 300 tasks) from real-world applications, namely Genome, BLAST, BWA, Cycles, Epigenomics, Montage, Seismology, SoyKB and SRASearch generated by the Pegasus Workflow Generator (PWG) [34, 35, 36], as well as the 3 most classical factorization algorithms of a $k \times k$ tiled matrix (LU, QR and Cholesky) [37]. For each factorization, we perform experiments with $k = 15$ with up to 1240 tasks. The number of vertices in the DAG depends on $k$ as follows: the Cholesky DAG has $\frac{1}{3}k^3 + O(k^2)$ tasks, while the LU and QR DAGs have $\frac{2}{3}k^3 + O(k^2)$ tasks. The task weights (WCETs) and file sizes of Pegasus workflows are generated by PWG, and those for the numerical kernels correspond to the number of floating-point operations and communication volumes according to [38]. The actual execution time (at $f_{max}$) of each task instance of a task $T_i$ is randomly generated from a uniform distribution or a normal distribution see details in Section 4.4.

*Reliability threshold.* We first compute the reliability of the whole workflow when there is a single replica per task running at $f_{max}$: $R(G) = \prod_{i=1}^{n} \mathcal{R}_i(f_{max})$. This will give us a corresponding probability of failure $F = 1 - R(G)$. Then we consider three reliability thresholds $\mathcal{R}(G)$ chosen as $1 - F$, $1 - \frac{F}{10}$ and $1 - \frac{F}{100}$. Thus, the reliability thresholds vary from one workflow to another. To compute the reliability threshold $\mathcal{R}(T_i)$ for each task $T_i$, we let $\mathcal{R}(T_i) = \sqrt[n]{\mathcal{R}(G)}$, where $n$ is the number of tasks in the workflow.

*Communication-to-Computation Ratio (CCR).* An important factor that influences the performance of scheduling strategies is the data-intensiveness of the application. We define the Communication-to-Computation Ratio (CCR) as the time $\tau$ needed to store all the files handled by a workflow (input, output, and intermediate files) divided by the time $T$ needed to perform all the computations of that workflow on a single processor. Here $\tau$ is simply the sum $S$ of all communication sizes (in bytes) divided by the bandwidth $b$, and $T$ is the sum of all WCETs (in

seconds). In other words, the CCR is

$$c = \frac{S}{b}\frac{1}{T} \tag{9}$$

We consider 3 CCR values namely $c = 1$, $c = 0.1$ and $c = 0.01$. Given the value $c$ of the CCR, we generate the value of the communication time $c_{i,j}$ as the size $s_{i,j}$ of the file communicated from $T_i$ to $T_j$ divided by the bandwidth $b = \frac{S}{cT}$ computed from Equation (9). When not specified otherwise, we report performance for the case $c = 1$. Again, the complete set of experimental results is available at [33].

*Deadlines.* We set five deadlines for each workflow: tightest, tight, medium, relatively loose, and loose. To calculate the tightest deadline $d_1$, we take the makespan when scheduling the workflow with HEFT with a single replica per task running at $f_{max}$. Then the loose deadline is $d_5 = 10 \times d_1$ and the other deadlines are at $\frac{1}{4}$, $\frac{1}{2}$, and $\frac{3}{4}$ of the interval $(d_1, d_5)$ respectively.

### *4.3. Limitations of* MILP

The time complexity of the MILP method is quite large: in [12], the authors of MILP report execution times for MILP ranging from 3 seconds to more than 300 seconds, with a median at 150 seconds, for a task graph only containing a total of 20 replicas (hence, for a workflow containing at most 10 tasks when each task has a second replica). They do not report execution times for workflows containing a total of 30 tasks and replicas (the largest workflows considered in [12]). We have extended MILP to ExtMILP as detailed in Section 4.1. In the simulations involving ExtMILP, there are three differences from the general setting: (i) we only consider the 8 workflows from the Pegasus suite that could generate graphs with 10 to 70 tasks (for instance, the Montage workflow has a minimal task number of 133 and is not considered); (ii) we consider three relatively low reliability thresholds 0.9, 0.93, and 0.96, that is, reliability thresholds which are chosen sufficiently large to have a good chance to be achievable with only two replicas per task; and (iii) we set a timeout of 600 seconds for ExtMILP (if, after 600 seconds ExtMILP has not been able to find an optimal solution, it will report the best feasible solution found so far, if it was able to find at least one). By construction of ExtMILP, the maximum number of replica per task is hard-coded in the linear program. However, the larger the number of replicas, the larger the running time of ExtMILP. Therefore, in the evaluation we first run ExtMILP with a maximum of two replicas per task. If ExtMILP answers that no solution exists, it is then run with a maximum of three replicas per task, and so on, until it finds a solution or the allowed time is exceeded.

### *4.4. Results*

We designed a discrete event simulator which is publicly available online [30] with all experimental data. Interested readers could run experiments with different settings, e.g., different numbers of cores, different workflow sizes, etc. Due to space limitations we only present a subset of our results: all results are available in the companion research report [33]. The performance reported on Figure 1 and on Figures 4 through 7 is based on the first frequency set and a generation of actual execution times that uses a uniform distribution and a $BC/WC$ ratio equal to 0.8. On Figure 1 and on Figures 4 through 7, the x-axis is the deadline of the application (from tightest to loosest), and the y-axis displays the ratio of the energy cost of the studied heuristic with respect to the energy cost of QFEC; hence, the lower the better. All energy costs are evaluated using the actual execution times of tasks (and not their WCETs).

Table 3: Type of the solution found by MILP with a reliability threshold of 0.96.

| | | deadline | | | | |
|---|---|---|---|---|---|---|
| | number of tasks | tightest | tight | medium | relatively loose | loose |
| BLAST | 20 | Suboptimal | Suboptimal | Optimal | Optimal | Optimal |
| | 40 | No Result | No Result | Suboptimal | Suboptimal | Suboptimal |
| | 60 | No Result | Suboptimal | Suboptimal | Suboptimal | Suboptimal |
| BWA | 20 | Suboptimal | Suboptimal | Optimal | Optimal | Optimal |
| | 40 | Suboptimal | Suboptimal | Suboptimal | Suboptimal | Suboptimal |
| | 60 | No Result | Suboptimal | Suboptimal | Suboptimal | Suboptimal |
| Cycles | 20 | Optimal | Optimal | Optimal | Optimal | Optimal |
| | 40 | Suboptimal | Suboptimal | Suboptimal | Suboptimal | Suboptimal |
| | 60 | No Result | Suboptimal | Suboptimal | Suboptimal | Suboptimal |
| Epigenomics | 20 | Optimal | Optimal | Optimal | Optimal | Optimal |
| | 40 | Optimal | Optimal | Optimal | Optimal | Optimal |
| | 60 | No Result | Suboptimal | No Result | Suboptimal | Suboptimal |
| Genome | 20 | Suboptimal | Suboptimal | Suboptimal | Optimal | Optimal |
| | 40 | No Result | Suboptimal | Suboptimal | Suboptimal | Suboptimal |
| | 60 | No Result | Suboptimal | No Result | No Result | Suboptimal |
| Seismology | 20 | Suboptimal | Suboptimal | Suboptimal | Suboptimal | Optimal |
| | 40 | Suboptimal | Suboptimal | Suboptimal | Suboptimal | Suboptimal |
| | 60 | Suboptimal | Suboptimal | Suboptimal | Suboptimal | Suboptimal |
| SoyKB | 20 | Optimal | Optimal | Optimal | Optimal | Optimal |
| | 40 | Optimal | Optimal | Optimal | Optimal | Optimal |
| | 60 (3 replicas per task) | No Result | No Result | No Result | No Result | No Result |
| SRASearch | 20 | Suboptimal | Optimal | Optimal | Optimal | Optimal |
| | 40 | No Result | Suboptimal | Suboptimal | Suboptimal | Suboptimal |
| | 60 | No Result | No Result | No Result | No Result | Suboptimal |

On graphs which display box plots, each box ranges from the 25th percentile to the 75th, and the horizontal line inside a box marks the average performance. The whiskers explicit the whole range of achieved performance from the best to the worst one.

ExtMILP *experiments.* In Table 3 we report the type of the solution found by ExtMILP: "Optimal" means that ExtMILP found an optimal solution; "Suboptimal" means that ExtMILP only found a feasible solution within the allocated 600 seconds, but not an optimal one; "No Result" means that ExtMILP was not able to find a single feasible solution in 600 seconds. Table 3 shows that as the number of tasks increases, ExtMILP requires more time to obtain the optimal solution. Hence, the quality of the solution found by ExtMILP decreases as the number of tasks increases, moving from an optimal solution, to a feasible solution (without any guarantee on its quality), to no solution at all. Therefore, in our simulations including ExtMILP we set the task size of the considered workflows from 10 to 70.

Figure 1 displays the relative performance of ExtMILP and of the other heuristics for the SRASearch workflow with 20 tasks. This figure shows that when the deadline becomes larger (and thus looser), our methods (e.g., TaskSize, LayerSize) perform (generally) better than ExtMILP. Indeed, with larger deadlines, there is more scheduling freedom and it becomes possible to decrease the overlap between the replicas of the same job. Such a strategy leads to a significant reduction of the energy consumption and explains why our heuristics outperform ExtMILP. Furthermore, when the reliability increases our heuristics outperform ExtMILP for tighter deadlines.

One can check in Figure 2 that this conclusion about the impact of the tightness of deadlines is general. For the tightest deadlines, ExtMILP achieves a performance similar to those of the best competitors. However, for looser deadlines our methods are significantly outperforming it.

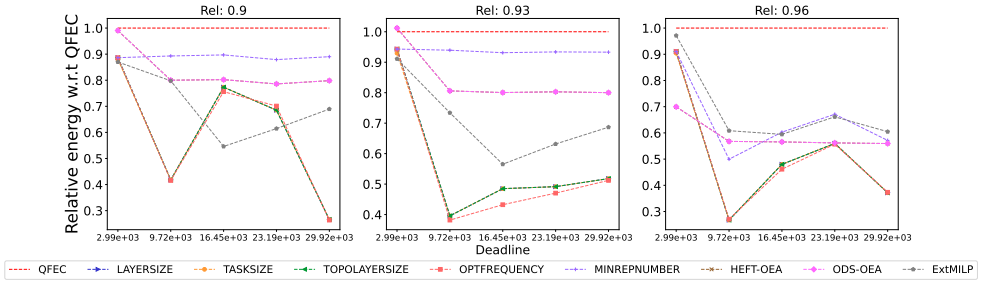


Figure 1: Assessing the performance of ExtMILP on the SRASearch workflow (with 20 tasks) under the frequency set $f_1$, a CCR of 1.0, and when actual execution times are generated with a uniform distribution and a $BC/WC$ ratio of 0.8.

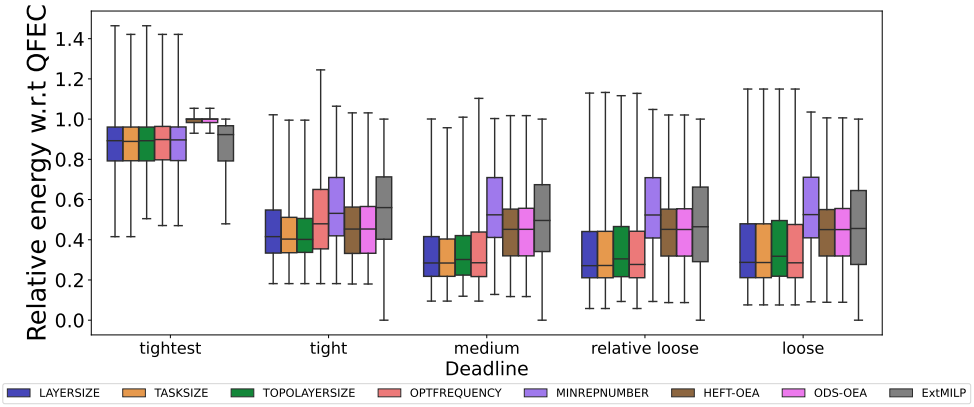


Figure 2: Comparing the performance of ExtMILP and of the other heuristics over all parameter settings as a function of the deadline tightness.

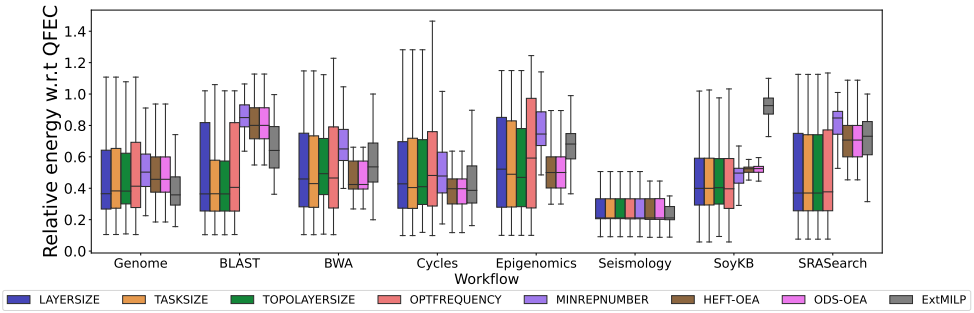


Figure 3: Comparing the performance of ExtMILP and of the other heuristics over all parameter settings as a function of the workflow.

Figure 3 aggregates the statistics over the performance of all heuristics over all parameter settings as a function of the considered workflow. ExtMILP achieves good to very good performance for three of these workflows (Genome, Cycles and Seismology), passable performance for one of them (BWA), and very bad performance for the remaining 4 workflows (BLAST, Epigenomics, SoyKB, SRASearch). This confirms that even for small workflows ExtMILP is not the heuristic of choice.

In fact, ExtMILP achieves the best performance only when the deadline is very tight or the reliability threshold is very low. This may surprise the reader as ExtMILP is designed to obtain an *optimal* solution. This is due to the choice of the objective function of ExtMILP. As we explained in Section 4.1, ExtMILP uses a pessimistic approximation to evaluate the energy consumption of a solution while our heuristics work with an optimistic one. The fundamental problem is the complexity of the evaluation of the energy consumption of a solution, mainly because of the potential overlapping of the different replicas of a single task.

*General experiments: study of particular instances.* We start by considering a set of well-chosen experiments (Figures 4 to 7) to assess some of the features of the proposed solutions. We move to the whole set of experiments right after this.

Figure 4 reports performance when the reliability is low enough so that each task only requires a single replica. Heft-OEA and ODS-OEA, which were designed for this peculiar case, are nonetheless outperformed by our heuristics, and especially by LayerSize and TaskSize (in this graph Heft-OEA is hidden behind ODS-OEA most of the time). Hence, while our new heuristics were specifically designed to work with any number of replicas, they are already very efficient with a single replica per task.

A major feature of our heuristics is their layer-by-layer approach. We assess the pertinence of this design choice with Figure 5. There, we compare the performance of heuristic TaskSize in its layer-by-layer approach and in a task-by-task version. This figure shows that a layer-by-layer approach reduces energy consumption when the deadline is tight. When the deadline is loose, there is enough scheduling freedom for even a task-by-task approach to avoid the overlapping of primary and secondary replicas. Hence, a layer-by-layer approach cannot lead to additional gains in such a case.

At runtime, we implement the optimization described in Section 3.7. We assess with Figure 6 the impact of this optimization. Once again, when the deadline is tight, this type of optimization can lead to additional gains. The runtime optimization never leads to worse performance, whatever the workflow and the parameter settings [33]; therefore, it should always be used.

Figure 7 reports the performance of the heuristics when the number of cores $M$ varies. We observe on this figure that the number of cores has no significant impact on the relative performance of our heuristics with respect to the competitors, Heft-OEA and ODS-OEA. Again, interested readers can run additional experiments with various settings using [30].

*General experiments: synthesis over all parameter settings.* Figure 8 presents a synthesis of the performance achieved by each heuristic for each workflow, over the whole set of experiments (for the different frequency sets, reliability thresholds, deadlines, communication-to-computation ratio, maximum $BC/WC$ ratio and the two different distributions used to generate the actual execution times).

It is obvious that Heft-OEA and ODS-OEA achieve rather poor performance overall and for each single workflow, even though they already lead to significant gains with respect to QFEC.

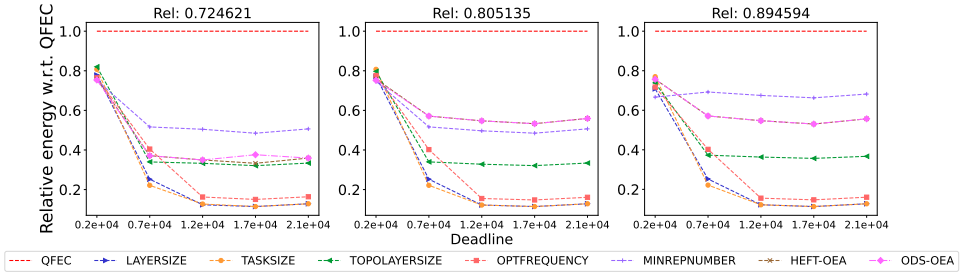


Figure 4: Assessing performance with the Montage workflow when the reliability threshold is very low (a single replica per task is needed) under the frequency set $f_1$, a CCR of 1.0, and when actual execution times are generated with a uniform distribution and a $BC/WC$ ratio of 0.8.

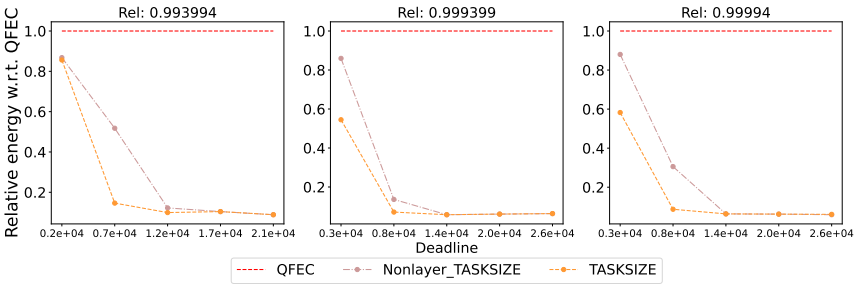


Figure 5: Assessing the impact of the layer-by-layer approach on the Montage workflow under the frequency set $f_1$, a CCR of 1.0, and when actual execution times are generated with a uniform distribution and a $BC/WC$ ratio of 0.8.

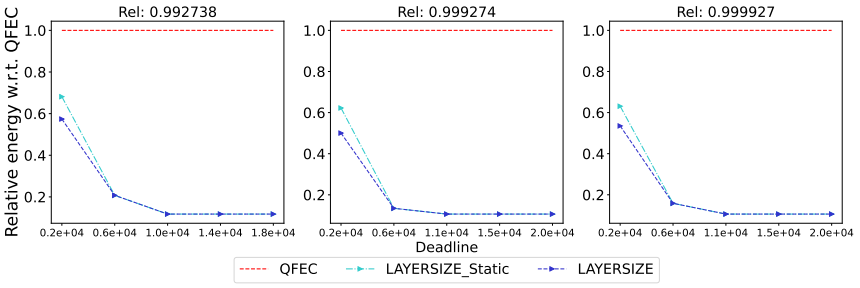


Figure 6: Assessing the impact of the runtime optimization on the Cycles workflow under the frequency set $f_1$, a CCR of 1.0, and when actual execution times are generated with a uniform distribution and a $BC/WC$ ratio of 0.8.

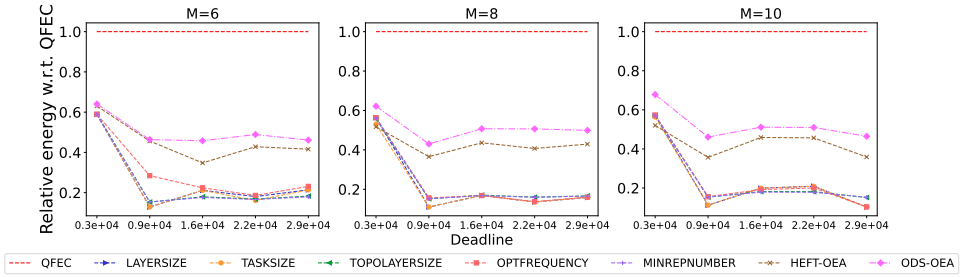


Figure 7: Assessing the impact of the number of processors $M$ on the Montage workflow under the frequency set $f_1$, a CCR of 1.0, a reliability threshold of 0.999399, and when actual execution times are generated with a uniform distribution and a $BC/WC$ ratio of 0.8.

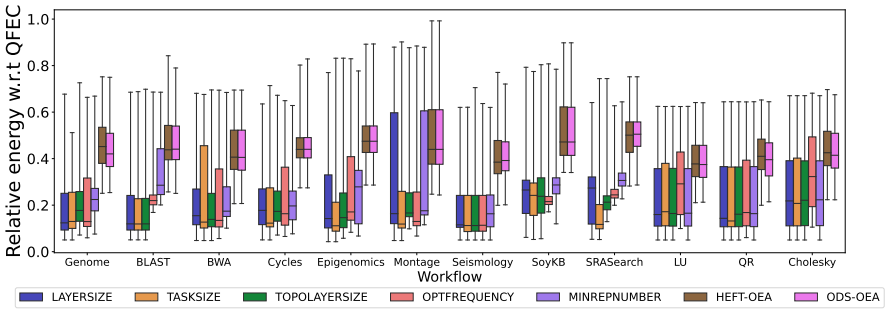


Figure 8: Average performance of the different heuristics for each workflow over all the parameter settings.

This proves the benefit of using DVFS for energy minimization, even if lowering the processor voltage leads to an increase in the fault rate.

MinRepNumber achieves some rather good performance for some of the workflows (typically the three factorization algorithms: LU, QR and Cholesky) and poor performance for others (BLAST and SRASearch being the worst workflows for MinRepNumber).

OptFrequency achieves better performance than MinRepNumber except for the three factorization algorithms. It even achieves the best performance overall for SoyKB but also has difficulties with the BLAST and SRASearch workflows.

Among the three remaining heuristics, TaskSize achieves slightly lower average performance than TaskSize and TopoLayerSize for workflows Cycles, Epigenomics, Montage, SRASearch and, less significantly, for BWA, QR and Cholesky. Its average is slightly outperformed for Genome and LU. Even if it achieves worse performance than OptFrequency for SoyKB, the difference of average performance is quite small. On the other hand, TaskSize achieves significantly better average performance than any other heuristic for SRASearch: its 75-th percentile is below the average of any other heuristic. Altogether, TaskSize achieves robust performance overall with very few weak points, and appears as a heuristic of choice. A peculiar case is that of workflow BWA: the 25th percentile and average performance of TaskSize is better than any other heuristics but its 75th percentile is extremely large; therefore, even if TaskSize achieves

Table 4: Statistics over all parameter settings on the amount of energy used by each heuristic expressed as a ratio over the energy used by QFEC (hence, the lower the better). Best, worst and geometric average ratios are reported.

| | LayerSize | TaskSize | TopoLayerSize | OptFrequency | MinRepNumber | Heft-OEA | ODS-OEA |
|---|---|---|---|---|---|---|---|
| Best | 0.0430 | 0.0429 | 0.0478 | 0.0502 | 0.0502 | 0.1994 | 0.2015 |
| Worst | 0.8794 | 0.9018 | 0.8770 | 0.8843 | 0.8786 | 0.9923 | 0.9923 |
| Mean | 0.1948 | **0.1721** | 0.1962 | 0.2181 | 0.2373 | 0.4410 | 0.4390 |

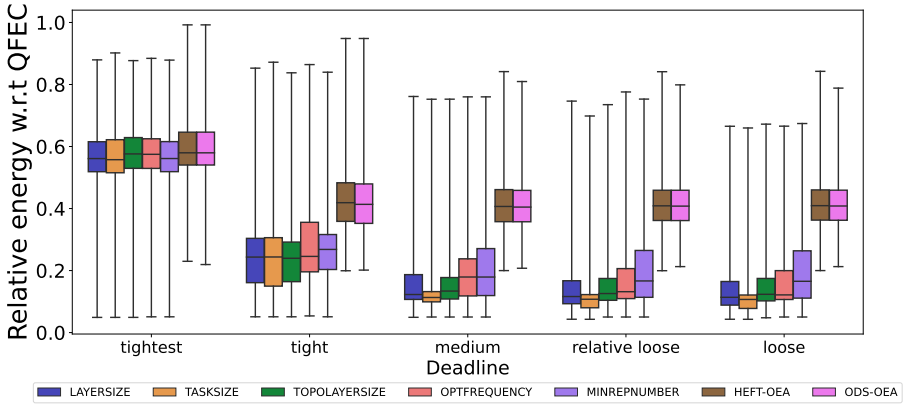


Figure 9: Average performance of the different heuristics for different deadlines over all the parameter settings.

very good average performance on that workflow it often achieves bad performance on it.

On average, LayerSize and TaskSize have an average energy gain of 60% over the competitors (Heft-OEA and ODS-OEA) and save 82% of the energy consumed by QFEC. Figure 9 presents a synthesis of the performance achieved by each heuristic, over the whole set of experiments, depending on the tightness of the deadline. It shows that when the deadline is the tightest, all heuristics achieve similar performance, even if the performance of ODS-OEA and Heft-OEA is slightly worse than that of the others. But as deadlines grow larger, the differences between heuristics grow larger too. Our heuristics perform far better, and they can save up to 95% of QFEC cost.

Table 4 presents statistics on the energy performance of the different heuristics over all the parameter settings. The performance reported is the energy used by each heuristic expressed as a fraction of the energy used by QFEC. We report the geometric mean (rather than the classical arithmetic mean) because we aggregate ratios and not raw values. Table 4 shows that TaskSize achieves the minimum mean energy cost. Its average performance is slightly better than those of LayerSize and TopoLayerSize. OptFrequency achieves slightly worse performance, and then MinRepNumber. The performance of both competitors (Heft-OEA and ODS-OEA) is significantly worse. These overall statistics confirm the previous analyses.

Because QFEC does not use DVFS, the fact that there are heuristics which achieve lower energy usage than QFEC shows that using DVFS enables to decrease energy consumption even if lowering processor voltage increases the fault rate which may lead to additional replicas.

Finally, we timed heuristics TaskSize and LayerSize on a subset of the real-world applica-

tions workflows (BWA, Cycles, Epigenomics, Genome), and on the QR factorization algorithm. For the real-world workflows, the running time of both TaskSize and LayerSize was under 0.5 second for workflows comprising 1,000 tasks, and under 31 seconds for 10,000 tasks (with an average at 11 seconds). For QR, their running time was under 0.9s for 1240 tasks, and under 5.2 seconds for 2870 tasks. Hence, TaskSize and LayerSize are both able to process large workflows in a fairly reasonable time. It should be noted that the fastest heuristic depends both upon the workflow and upon its size.

## 5. Related work

A lot of works focus on this tri-criteria optimization problem (reliability, deadlines, energy), but for independent task sets. Reference [27] proposes both static and dynamic methods to reduce overlapping between replicas, thus minimizing energy consumption. Reference [39] improves the approach of [27] through three steps, and proposes several new scheduling heuristics, which have the best performance under a wide range of scenarios. Reference [40] considers an energy-budget-aware reliability management (enBudRM) method for multi-core embedded systems featuring hybrid energy source, while [41] proposes heuristic-based scheduling techniques to minimize the energy consumption of task executions when errors are absent, and preserves feasibility under the worst case of error occurrences. Reference [42] considers deadline-partition scheduling algorithm for periodic tasks on heterogeneous platforms. Reference [43] presents a low-overhead strategy to reduce energy consumption while enforcing deadlines on heterogeneous platforms. References [44, 45, 46, 47] introduce low-overhead multi-hierarchical heuristics to reduce energy consumption while satisfying temperature constraints for real-time systems.

There are other works that consider managing energy using DVFS for real-time workflows. Reference [48] considers precise scheduling of tasks in a mixed-criticality model. Reference [49] aims to schedule sporadic parallel tasks with deadline constraints while saving energy and proposes a real-time scheduler to solve this problem. Reference [50] considers energy-aware duplication scheduling algorithms for a parallel application on homogeneous systems. Reference [51] presents energy-conscious scheduling to implement joint reduction between the schedule length and energy consumption of a parallel application on heterogeneous systems. Reference [52] investigates the problem of reducing energy consumption with a schedule length constraint for a parallel application on heterogeneous systems by reclaiming the slack time for each task on its fixed assigned processor. Reference [53] also examines the same problem as that reported in [52] by switching off inefficient processors to reduce static energy consumption based on slack time reclamation. The MaxRe [54] and RR [55] algorithms aim at reducing resource consumption while satisfying the reliability goal of a parallel application on heterogeneous systems. However, they only aim to reduce resource consumption cost, which refers to the resource usage of processors when tasks are running.

The works listed above do not consider reliability, but it has been widely proved that DVFS has negative effects on transient failure rates. As a result, it is important to take reliability degradation into consideration while managing energy by DVFS. Reference [56] proposes reliability-aware power management schemes to save energy while guaranteeing a certain level of system reliability on homogeneous multiprocessors. Reference [57] introduces a power-efficient reliability management method through dynamic redundancy and voltage scaling under variations on homogeneous manycore processors. Reference [58] presents an N-modular redundancy (NMR) technique to achieve high reliability with low energy overhead for hard real-time applications on homogeneous multicore processors.

The tri-criteria problem is also studied in [59] but again with major differences: mapping of the tasks onto the processors is assumed to be given, as in [11], and reliability is achieved through re-execution, not replication. Similarly, reference [60] studies the tri-criteria optimization problem with a given mapping on heterogeneous architectures, assuming that the user specifies the maximum number of failures per processor tolerated to satisfy the reliability constraint. Reference [61] addresses the tri-criteria optimization problem by choosing some tasks that have to be re-executed to match the reliability constraint. However, they restrict to the scheduling problem on one single processor, and they consider only the energy consumption of the first execution of a task (best-case scenario) when re-execution is done. Reference [56] studied energy minimization problem for real-time workflows on homogeneous platforms, while reference [62] worked on the same problem on heterogeneous platforms. Both works save energy by scaling down frequencies of selected tasks, and preserve reliability by shared recovery. But they can only guarantee the original reliability level, which means all tasks running at $f_{max}$ with no replica. Finally, reference [63] has proposed an off-line tri-criteria scheduling heuristic (TSH), which uses active replication to minimize the makespan, with a threshold on the global failure rate and the maximum power consumption. TSH is an improved critical-path list scheduling heuristic that takes into account power and reliability before deciding which task to assign and to duplicate onto the next free processors. The complexity of this heuristic is unfortunately exponential in the number of processors. To the best of our knowledge, the only existing solutions based on replication that applies to workflows of arbitrary shape are OEA [11] and MILP [12], which we described in Section 4.1. As already stated, MILP allows at most two replicas per task and the original version of OEA has only one, which limits their solutions to low reliability thresholds only.

Finally, in all the works above, the mapping procedure always proceeds as a list-scheduling heuristic, where ready tasks are sorted according to some priority in a waiting queue and mapped in this order. The idea of mapping a chunk of ready tasks rather than only the task with the highest priority is used in [64] to achieve a better load-balancing at each decision step when targeting an heterogeneous platform. We re-use the idea but focusing on graph layers rather than arbitrary chunks, and with a different objective, namely to avoid overlap between replicas.

## 6. Conclusion

In this paper, we have introduced scheduling and mapping heuristics for real-time workflows, with a general tri-criteria objective (deadline, reliability, energy). A key design element is our novel layer-by-layer approach, which allows to considerably limit the overlap between the replicas of a task, thereby dramatically saving energy. To the best of our knowledge, these heuristics are the first to solve the general problem with an arbitrary number of replicas. We have assessed the performance of our heuristics via an extensive set of experimental scenarios, and we have shown that they significantly outperform the best available competitors from the literature. Specifically, our best heuristics LayerSize and TaskSize have an energy gain of 60% over the Heft-OEA and ODS-OEA competitors and save 82% of the energy consumed by QFEC. We also report partial comparisons with ExtMILP, which does not take communication times into account. On the contrary, our heuristics can deal with much larger workflows, account for communication times, and include as many replicas as needed to meet high reliability thresholds.

Future work will investigate whether complementary resilience techniques such as introducing intermediate error detectors and checkpointing can help decrease the number of replicas, and thereby the total energy consumption.

*Acknowledgments*

The authors would like to thank the reviewers and editors for their comments and suggestions, which greatly helped improve the final version of the paper. This work was supported in part 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. Corresponding authors: Li Han(*hanli@sei.ecnu.edu.cn*) from the School of Software Engineering, East China Normal University and Jing Liu(*jliu@sei.ecnu.edu.cn*) from Shanghai Key Lab of Trustworthy Computing, East China Normal University and Frédéric Vivien($frederic.vivien@inria.fr$) from ENS Lyon.

## References

[1] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, K. Vahi, Characterization of scientific workflows, in: SC'08 Workshop: The 3rd Workshop on Workflows in Support of Large-scale Science (WORKS08) web site, ACM/IEEE, Austin, TX, 2008.

[2] E. Caron, F. Desprez, T. Glatard, M. Ketan, J. Montagnat, D. Reimert, Workflow-based comparison of two distributed computing infrastructures, in: Workflows in Support of Large-Scale Science (WORKS10), In Conjunction with Supercomputing 10 (SC'10), IEEE, New Orleans, 2010, hal-00677820.
URL `https://hal.inria.fr/hal-00677820`

[3] P. Couvares, T. Kosar, A. Roy, J. Weber, K. Wenger, Workflow Management in Condor, in: I. Taylor, E. Deelman, D. Gannon, M. Shields (Eds.), Workflows for e-Science, Springer, 2007, pp. 357–375.

[4] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. Jacob, D. Katz, Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems, Scientific Programming Journal 13 (3) (2005) 219–237.

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

[6] H. Casanova, A. Legrand, Y. Robert, Parallel Algorithms, Chapman and Hall/CRC Press, 2008.

[7] H. Topcuoglu, S. Hariri, M. Y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Trans. Parallel Distributed Systems 13 (3) (2002) 260–274.

[8] C. Weaver, J. Emer, S. S. Mukherjee, S. K. Reinhardt, Techniques to reduce the soft error rate of a high-performance microprocessor, ACM SIGARCH Computer Architecture News 32 (2) (2004) 264.

[9] A. Burns, A. Wellings, Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX, 4th Edition, Addison-Wesley, 2009.

[10] A. Dixit, A. Wood, The impact of new technology on soft error rates, in: 2011 International Reliability Physics Symposium, IEEE, 2011, pp. 5B–4.

[11] J. Huang, R. Li, X. Jiao, Y. Jiang, W. Chang, Dynamic DAG Scheduling on Multiprocessor Systems: Reliability, Energy, and Makespan, IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems 39 (11) (2020) 3336–3347.

[12] M. Cui, A. Kritikakou, L. Mo, E. Casseau, Fault-tolerant mapping of real-time parallel applications under multiple dvfs schemes, in: IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS), 2021, pp. 387–399.

[13] H. Chen, X. Zhu, D. Qiu, L. Liu, Uncertainty-aware real-time workflow scheduling in the cloud, in: IEEE 9th Int. Conf. Cloud Computing (CLOUD), 2016, pp. 577–584.

[14] D. C. Snowdon, G. Van Der Linden, S. M. Petters, G. Heiser, Accurate run-time prediction of performance degradation under frequency scaling, in: Workshop on Operating Systems Platforms for Embedded Real-Time applications, 2007, p. 58.

[15] D. Zhu, R. Melhem, D. Mosse, The effects of energy management on reliability in real-time embedded systems, in: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2004, pp. 35–40.

[16] B. Zhao, H. Aydin, D. Zhu, Reliability-aware dynamic voltage scaling for energy-constrained real-time embedded systems, in: Proceedings of the IEEE International Conference on Computer Design (ICCD), 2008, pp. 633–639.

[17] A. Dixit, A. Wood, The impact of new technology on soft error rates, in: IEEE International on Reliability Physics Symposium (IRPS), 2011, pp. 5B.4.1–5B.4.7.

[18] G. Aupy, A. Benoit, Y. Robert, Energy-aware scheduling under reliability and makespan constraints, in: Proceedings of the International Conference on High Performance Computing (HiPC), 2012, pp. 1–10.

[19] A. Das, A. Kumar, B. Veeravalli, C. Bolchini, A. Miele, Combined DVFS and mapping exploration for lifetime and soft-error susceptibility improvement in MPSoCs, in: Proceedings of the Conference on Design, Automation & Test in Europe (DATE), 2014, pp. 61:1–61:6.
[20] M. Patterson, The effect of data center temperature on energy efficiency, in: Proceedings of 11th Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems, 2008, pp. 1167–1174.
[21] W.-C. Feng, Making a case for efficient supercomputing, Queue 1 (7) (2003) 54–64.
[22] C.-H. Hsu, W.-C. Feng, A power-aware run-time system for high-performance computing, in: Proceedings of the ACM/IEEE Supercomputing Conference (SC), 2005, pp. 1–9.
[23] O. Sarood, E. Meneses, L. V. Kale, A 'cool' way of improving the reliability of HPC machines, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC), 2013, pp. 58:1–58:12.
[24] H. K. N. Leung, P. W. L. Wong, A study of user acceptance tests, Softw. Qual. J. 6 (2) (1997) 137–149. doi:10.1023/A:1018503800709.
URL https://doi.org/10.1023/A:1018503800709
[25] G. Manimaran, C. S. R. Murthy, A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis, IEEE Trans. Parallel Distributed Syst. 9 (11) (1998) 1137–1152. doi:10.1109/71.735960.
URL https://doi.org/10.1109/71.735960
[26] G. Yao, Y. Ding, K. Hao, Using imbalance characteristic for fault-tolerant workflow scheduling in cloud systems, IEEE Trans. Parallel Distributed Syst. 28 (12) (2017) 3671–3683. doi:10.1109/TPDS.2017.2687923.
URL https://doi.org/10.1109/TPDS.2017.2687923
[27] M. A. Haque, H. Aydin, D. Zhu, On reliability management of energy-aware real-time systems through task replication, IEEE Transactions on Parallel and Distributed Systems 28 (3) (2017) 813–825.
[28] G. Xie, G. Zeng, R. Li, K. Li, Scheduling Parallel Applications on Heterogeneous Distributed Systems, Springer, 2019.
[29] F. Glover, E. Woolsey, Technical note: Converting the 0-1 polynomial programming problem to a 0-1 linear program, Operations Research 22 (1) (1974) 180–182.
[30] Simulator, https://github.com/wuzhui1995/Energy-aware-mapping-and-scheduling-strategies.
[31] N. B. Rizvandi, A. Y. Zomaya, Y. C. Lee, A. J. Boloori, J. Taheri, Multiple frequency selection in dvfs-enabled processors to minimize energy consumption, Energy-Efficient Distributed Computing Systems (2012) 443–463.
[32] Frequency behavior-intel, https://en.wikichip.org/wiki/intel/frequency_behavior.
[33] Z. Wu, L. Han, J. Liu, Y. Robert, F. Vivien, Energy-aware mapping and scheduling strategies for real-time workflows under reliability constraints, Research Report RR-9469, INRIA (Jul. 2022).
URL https://hal.inria.fr/hal-03641039v2
[34] R. F. da Silva, W. Chen, G. Juve, K. Vahi, E. Deelman, Community resources for enabling research in distributed scientific workflows, in: e-Science (e-Science), 2014 IEEE 10th International Conference on, Vol. 1, IEEE, 2014, pp. 177–184.
[35] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, K. Vahi, Characterization of scientific workflows, in: Workflows in Support of Large-Scale Science (WORKS), IEEE, 2008, pp. 1–10.
[36] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, K. Vahi, Characterizing and profiling scientific workflows, Future Generation Computer Systems 29 (3) (2013) 682–692.
[37] J. Choi, J. J. Dongarra, L. S. Ostrouchov, A. P. Petitet, D. W. Walker, R. C. Whaley, Design and implementation of the scalapack lu, qr, and cholesky factorization routines, Scientific Programming 5 (3) (1996) 173–184.
[38] G. H. Golub, C. F. V. Loan, Matrix Computations, Fourth Edition, 4th Edition, Johns Hopkins University Press, 2013.
[39] L. Han, L.-C. Canon, J. Liu, Y. Robert, F. Vivien, Improved energy-aware strategies for periodic real-time tasks under reliability constraints, in: RTSS'2019, the 40th IEEE Real-Time Systems Symposium, IEEE Press, 2019.
[40] S. Safari, M. Ansari, M. Salehi, A. Ejlali, Energy-budget-aware reliability management in multi-core embedded systems with hybrid energy source, CSI Journal on Computer Science and Engineering 15 (2018) 31–43.
[41] Y. Liu, H. Liang, K. Wu, Scheduling for energy efficiency and fault tolerance in hard real-time systems, in: Design, Automation and Test in Europe, DATE 2010, Dresden, Germany, March 8-12, 2010, 2010, pp. 1444–1449. doi:10.1109/DATE.2010.5457039.
URL https://doi.org/10.1109/DATE.2010.5457039
[42] S. Moulik, R. Devaraj, A. Sarkar, A. Shaw, A deadline-partition oriented heterogeneous multi-core scheduler for periodic tasks, in: S. Horng (Ed.), 18th International Conference on Parallel and Distributed Computing, Applications and Technologies, PDCAT 2017, Taipei, Taiwan, December 18-20, 2017, IEEE Computer Society, 2017, pp. 204–210. doi:10.1109/PDCAT.2017.00041.
URL https://doi.org/10.1109/PDCAT.2017.00041
[43] S. Moulik, Z. Das, R. Devaraj, S. Chakraborty, SEAMERS: A semi-partitioned energy-aware scheduler for heterogeneous multicorereal-time systems, J. Syst. Archit. 114 (2021) 101953. doi:10.1016/j.sysarc.2020.

101953.
URL https://doi.org/10.1016/j.sysarc.2020.101953

[44] Y. Sharma, Z. Das, S. Moulik, TEFRED: A temperature and energy cognizant fault-tolerant real-time scheduler based on deadline partitioning for heterogeneous platforms, in: H. Shen, Y. Sang, Y. Zhang, N. Xiao, H. R. Arabnia, G. C. Fox, A. Gupta, M. Malek (Eds.), Parallel and Distributed Computing, Applications and Technologies - 22nd International Conference, PDCAT 2021, Guangzhou, China, December 17-19, 2021, Proceedings, Vol. 13148 of Lecture Notes in Computer Science, Springer, 2021, pp. 358–366. doi:10.1007/978-3-030-96772-7_33.
URL https://doi.org/10.1007/978-3-030-96772-7_33

[45] S. Moulik, Z. Das, TASOR: A temperature-aware semi-partitioned real-time scheduler, in: TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON), Kochi, India, October 17-20, 2019, IEEE, 2019, pp. 1578–1583. doi:10.1109/TENCON.2019.8929501.
URL https://doi.org/10.1109/TENCON.2019.8929501

[46] Y. Sharma, S. Chakraborty, S. Moulik, ETA-HP: an energy and temperature-aware real-time scheduler for heterogeneous platforms, J. Supercomput. 78 (8) (2022) 1–25. doi:10.1007/s11227-021-04257-7.
URL https://doi.org/10.1007/s11227-021-04257-7

[47] S. Moulik, RESET: A real-time scheduler for energy and temperature aware heterogeneous multi-core systems, Integr. 77 (2021) 59–69. doi:10.1016/j.vlsi.2020.11.012.
URL https://doi.org/10.1016/j.vlsi.2020.11.012

[48] A. Bhuiyan, S. Sruti, Z. Guo, K. Yang, Precise scheduling of mixed-criticality tasks by varying processor speed, in: J. Ermont, Y. Song, C. D. Gill (Eds.), Proceedings of the 27th International Conference on Real-Time Networks and Systems, RTNS 2019, Toulouse, France, November 06-08, 2019, ACM, 2019, pp. 123–132. doi:10.1145/3356401.3356410.
URL https://doi.org/10.1145/3356401.3356410

[49] Z. Guo, A. Bhuiyan, D. Liu, A. Khan, A. Saifullah, N. Guan, Energy-efficient real-time scheduling of DAGs on clustered multi-core platforms, in: B. B. Brandenburg (Ed.), 25th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2019, Montreal, QC, Canada, April 16-18, 2019, IEEE, 2019, pp. 156–168. doi:10.1109/RTAS.2019.00021.
URL https://doi.org/10.1109/RTAS.2019.00021

[50] Z. Zong, A. Manzanares, X. Ruan, X. Qin, EAD and PEBD: two energy-aware duplication scheduling algorithms for parallel tasks on homogeneous clusters, IEEE Trans. Computers 60 (3) (2011) 360–374. doi:10.1109/TC.2010.216.
URL https://doi.org/10.1109/TC.2010.216

[51] Y. C. Lee, A. Y. Zomaya, Energy conscious scheduling for distributed computing systems under different operating conditions, IEEE Trans. Parallel Distributed Syst. 22 (8) (2011) 1374–1381. doi:10.1109/TPDS.2010.208.
URL https://doi.org/10.1109/TPDS.2010.208

[52] Q. Huang, S. Su, J. Li, P. Xu, K. Shuang, X. Huang, Enhanced energy-efficient scheduling for parallel applications in cloud, in: 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2012, Ottawa, Canada, May 13-16, 2012, 2012, pp. 781–786. doi:10.1109/CCGrid.2012.49.
URL https://doi.org/10.1109/CCGrid.2012.49

[53] Z. Tang, L. Qi, Z. Cheng, K. Li, S. U. Khan, K. Li, An energy-efficient task scheduling algorithm in dvfs-enabled cloud environment, J. Grid Comput. 14 (1) (2016) 55–74. doi:10.1007/s10723-015-9334-y.
URL https://doi.org/10.1007/s10723-015-9334-y

[54] L. Zhao, Y. Ren, Y. Xiang, K. Sakurai, Fault tolerant scheduling with dynamic number of replicas in heterogeneous system, in: 12th IEEE International Conference on High Performance Computing and Communications, HPCC 2010, 1-3 September 2010, Melbourne, Australia, 2010, pp. 434–441. doi:10.1109/HPCC.2010.72.
URL https://doi.org/10.1109/HPCC.2010.72

[55] L. Zhao, Y. Ren, K. Sakurai, Reliable workflow scheduling with less resource redundancy, Parallel Comput. 39 (10) (2013) 567–585. doi:10.1016/j.parco.2013.06.003.
URL https://doi.org/10.1016/j.parco.2013.06.003

[56] Y. Guo, D. Zhu, H. Aydin, Reliability-aware power management for parallel real-time applications with precedence constraints, in: Proceedings of the 2011 International Green Computing Conference and Workshops, IGCC '11, IEEE Computer Society, USA, 2011, p. 1–8. doi:10.1109/IGCC.2011.6008562.
URL https://doi.org/10.1109/IGCC.2011.6008562

[57] M. Salehi, M. K. Tavana, S. Rehman, F. Kriebel, M. Shafique, A. Ejlali, J. Henkel, DRVS: power-efficient reliability management through dynamic redundancy and voltage scaling under variations, in: IEEE/ACM International Symposium on Low Power Electronics and Design, ISLPED 2015, Rome, Italy, July 22-24, 2015, 2015, pp. 225–230. doi:10.1109/ISLPED.2015.7273518.
URL https://doi.org/10.1109/ISLPED.2015.7273518

[58] M. Salehi, A. Ejlali, B. M. Al-Hashimi, Two-phase low-energy n-modular redundancy for hard real-time multi-core

systems, IEEE Trans. Parallel Distributed Syst. 27 (5) (2016) 1497–1510. doi:10.1109/TPDS.2015.2444402.
URL https://doi.org/10.1109/TPDS.2015.2444402

[59] G. Aupy, A. Benoit, Y. Robert, Energy-aware scheduling under reliability and makespan constraints, in: International Conference on High Performance Computing (HiPC'2012), IEEE Computer Society Press, 2012.

[60] P. Pop, K. H. Poulsen, V. Izosimov, P. Eles, Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems, in: Proc. of IEEE/ACM Int. Conf. on Hardware/software codesign and system synthesis (CODES+ISSS), 2007, pp. 233–238.

[61] D. Zhu, H. Aydin, Energy management for real-time embedded systems with reliability requirements, in: Proceedings of the IEEE/ACM International Conference on Computer-aided Design (ICCAD), 2006, pp. 528–534.

[62] L. Zhang, K. Li, K. Li, Y. Xu, Joint optimization of energy efficiency and system reliability for precedence constrained tasks in heterogeneous systems, International Journal of Electrical Power and Energy Systems 78 (2016) 499–512. doi:https://doi.org/10.1016/j.ijepes.2015.11.102.
URL https://www.sciencedirect.com/science/article/pii/S0142061515005335

[63] I. Assayad, A. Girault, H. Kalla, Tradeoff exploration between reliability, power consumption, and execution time for embedded systems, International Journal on Software Tools for Technology Transfer 15 (3) (2013) 229–245.

[64] O. Beaumont, V. Boudet, Y. Robert, The iso-level scheduling heuristic for heterogeneous processors, in: PDP'2002, 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing, IEEE Computer Society Press, 2002.