Improved energy-aware strategies for periodic real-time tasks under reliability constraints

Li Han*†, Louis-Claude Canon‡, Jing Liu*, Yves Robert†§ and Frédéric Vivien†
*East China Normal University, China
email: jliu@sei.ecnu.edu.cn
†Univ Lyon, CNRS, ENS de Lyon, Inria, Université Claude-Bernard Lyon 1, LIP UMR5668, F-69342, LYON Cedex 07, France
email: {li.han|yves.robert|frederic.vivien}@inria.fr
‡FEMTO-ST Institute, CNRS, Univ. Bourgogne Franche-Comté, France
email: louis-claude.canon@univ-fcomte.fr
§University of Tennessee Knoxville, USA

***Abstract*—This paper revisits the real-time scheduling problem recently introduced by Haque, Aydin and Zhu (2017). In this challenging problem, task redundancy ensures a given level of reliability while incurring a significant energy cost. By carefully setting processing frequencies, allocating tasks to processors and ordering task executions, we improve on the previous state-of-the-art approach with an average gain in energy of 20%. Furthermore, we establish the first complexity results for specific instances of the problem.**

I. Introduction

This work revisits recent results presented by Haque, Aydin and Zhu [1]. We call [1] *the reference paper* throughout the text. The reference paper deals with the following optimization problem: given a set of independent real-time tasks subject to (possibly different) periodic deadlines, how to execute them on a parallel platform and match all deadlines while minimizing the expected energy consumption? The problem is complicated by the need to enforce some reliability threshold, which is a standard constraint in real-time systems. In Section II below, we provide all details about this optimization problem. We do not provide any further motivation for this study: instead, we refer the reader to the reference paper. We simply point out that energy consumption and reliability have become extremely important and challenging problems in real-time systems.

The paper is organized as follows. Section II provides a detailed description of the optimization problem, and of the three-step approach used in the reference paper to solve it: (i) for each task, compute a set of replicas and their frequencies; (ii) map and statically schedule all replicas onto the platform; (iii) dynamically update the schedule based on actual completion times (instead of worst-case ones) and observed successful executions. These three steps are described in Sections II-B to II-D. Section III provides an example explaining the limitation of the scheduling strategy proposed in [1]. Then Section IV outlines new propositions for each step, as well as a new complexity result that establishes the combinatorial nature of the scheduling step when the mapping is given. Section IV-D summarizes new heuristics, while Section IV-E deals with complexity results. Section V is devoted to a comprehensive experimental comparison of the results of the reference paper against those obtained with our improved approach[1]. Section VI presents related work, with the aim of complementing the related work already covered in the reference paper: we focus on recent works that quote the reference paper and briefly discuss their contributions. Finally, Section VII gives concluding remarks and hints for future work.

II. Previous approach

In this section, we present the optimization problem in full details, and we describe the approach of the reference paper. Key notations are summarized in Table I.

A. Optimization problem

The inputs to the optimization problem are a set of real-time tasks, a set of processors and a reliability target:

Tasks – There are n periodic real-time tasks $\tau_1, \tau_2, ..., \tau_n$. Task τ_i has worst-case execution time (WCET) c_i under the maximum available frequency f_{max}. Tasks actually complete execution earlier than their estimated WCET: execution times are assumed to be data-dependent and non-deterministic, randomly sampled from some probability distribution whose support is upper bounded by the WCET. Task τ_i generates a sequence of *instances* with period p_i, which is equal to its deadline. The whole input pattern repeats every hyperperiod of length $L = \text{lcm}_{1 \leq i \leq n}\, p_i$. Each task τ_i has $\frac{L}{p_i}$ instances within the hyperperiod.

Processors – The platform consists of M homogeneous processors, with same set F of frequencies ranging from f_{min} to f_{max}. Without loss of generality, we normalize the frequencies to enforce $f_{max} = 1$. At frequency f_j, a processor needs up to c_{i,f_j} seconds, which is equal to $\frac{c_i}{f_j}$ in the reference paper, to complete an instance of task τ_i. The *utilization* u_{i,f_j} of task τ_i at frequency f_j is defined as $u_{i,f_j} = \frac{c_{i,f_j}}{p_i}$. The utilization of a processor is the sum of the utilizations of all tasks that are assigned to it.

[1]The algorithms are implemented in C++ and R and the related code, data and analysis are available at https://doi.org/10.6084/m9.figshare.9778319.v1

Fault model – One considers transient faults, modeled by an exponential distribution with average arrival rate λ. It has been widely proved that the fault rate λ increases when the frequency is scaled down to save energy using DVFS [2]. Let λ_0 denote the fault rate at frequency f_{max}. Then the fault rate at frequency f_i is $\lambda(f_i) = \lambda_0 \times \exp^{\frac{d(1-f_i)}{1-f_{min}}}$, 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. At the end of execution, there is an *acceptance test* to check the occurrence of soft errors induced by the transient faults. It is assumed that acceptance tests are 100% accurate. The duration of the test is included within the task WCET. The *reliability* of a task instance is the probability of executing it successfully, in the absence of faults. The reliability of a single instance of task τ_i running at frequency f_j is $R_i(f_j) = \exp^{-\lambda(f_j)c_{i,f_j}}$.

Reliability threshold – Let $\mathcal{R}_i$ denote the reliability threshold for each instance of task τ_i; $\mathcal{R}_i$ may be given as part of the input. The reference paper also deals with the case where a reliability threshold is given for the whole task system over the hyperperiod: then $\mathcal{R}_i$ is computed using the *Uniform Reliability Scaling* technique [3]. We have $\omega = \frac{\phi_{i,target}}{\hat{\phi_i}}$ for all i, where ω is the uniform probability of failure scaling factor (given as part of the input), $\phi_{i,target}$ is the failure probability of task τ_i ($\phi_{i,target} = 1 - \mathcal{R}_i{}^{h_i}$, where h_i is the number of instances of task τ_i in the hyperperiod) and $\hat{\phi_i}$ is the failure probability of task τ_i when a single replica at maximum frequency is executed ($\hat{\phi_i} = 1 - R_i(f_{max})^{h_i}$). This leads to $\mathcal{R}_i = \sqrt[h_i]{1 - \omega(1 - R_i(f_{max})^{h_i})}$. Now, given the reliability threshold $\mathcal{R}_i$, the question is to determine how many replicas to use, and at which frequency to execute them, so that $\mathcal{R}_i$ is enforced while energy consumption is kept minimal. Note that all 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.

Optimization objective – The objective is to determine a set of replicas for each task and their execution frequencies, and to build a static schedule of length the hyperperiod, where the replicas of each instance of each task are mapped onto the processors, so that energy consumption is minimized, while matching the deadline p_i and reliability threshold $\mathcal{R}_i$ for each instance of each task τ_i. We detail below how energy consumption is estimated. To further complicate matters, the static schedule is dynamically modified on the fly to take actual execution times rather than WCET into account. Also, as soon as one replica of a given task instance completes its execution successfully, all its other replicas become redundant and are terminated instantaneously.

B. Replica sets

In the reference paper, the first step is to construct a table with all information needed. For each task τ_i and each possible frequency we compute the number of replicas needed, the corresponding energy cost and the CPU time.

Table I: Key Notations

Notation	Explanation
c_i	WCET for task τ_i under max. available frequency
p_i	period (deadline) for each task instance of task τ_i
h_i	number of instances of task τ_i in the hyperperiod
f_j	frequency $f_j \in F = \{f_{min}, \dots, f_{max} = 1\}$
c_{i,f_j}	WCET for task τ_i at frequency f_j
u_{i,f_j}	utilization of task τ_i at frequency f_j
$f_{opt(i)}$	most energy-efficient frequency for task τ_i
$R_i(f_j)$	reliability of one instance of task τ_i under f_j
$\mathcal{R}_i$	target reliability threshold for one instance of task τ_i
w	uniform probability of failure scaling factor
k_{i,f_j}	min. replica number for task τ_i under f_j to meet $\mathcal{R}_i$
$E(\tau_i, f_j, k_{i,f_j})$	energy cost for task τ_i with k_{i,f_j} replicas under f_j

Given a frequency f_j, we start by computing the number k_{i,f_j} of copies that are needed for (each instance of) τ_i. The reliability using a single task instance (no replica) is $R_i(f_j) = \exp^{-\lambda(f_j)c_{i,f_j}}$. If $R_i(f_j) \geq \mathcal{R}_i$, the reliability threshold is enforced, and we need no replica, hence $k_{i,f_j} = 1$. Otherwise, using r additional replicas, the reliability increases to $R_i^r(f_j) = 1 - (1 - R_i(f_j))^{r+1}$ (the task fails only if all $r+1$ copies fail), and we take the minimum value r such that $R_i^r(f_j) \geq \mathcal{R}_i$. This leads to $k_{i,f_j} = r + 1 = \left\lceil \frac{\log(1-\mathcal{R}_i)}{\log(1-R_i(f_j))} \right\rceil$ with $\mathcal{R}_i = \sqrt[h_i]{1 - \omega(1 - R_i(f_{max})^{h_i})}$ (see Section II-A). The reference paper maps different copies onto different processors, so necessarily $k_{i,f_j} \leq M$. If no value of k_{i,f_j} can be found, frequency f_j cannot be used, and a higher frequency must be selected.

Once we have determined the number k_{i,f_j} of copies of τ_i at frequency f_j (with $1 \leq k_{i,f_j} \leq M$), we compute the corresponding energy cost. The reference paper adopts a conservative strategy and sums up the energy cost of all copies. This is pessimistic because as soon as a copy is successful, the remaining copies are interrupted (if already started) or simply cancelled (if not started). The energy cost of a copy is estimated as the power times the execution time $c_{i,f_j} = \frac{c_i}{f_j}$, which is an upper bound. 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 $E(\tau_i, f_j, 1)$ for one copy of task τ_i at f_j is then: $E(\tau_i, f_j, 1) = P(f_j) \times c_{i,f_j}$. The final energy cost with k_{i,f_j} copies is estimated as $E(\tau_i, f_j, k_{i,f_j}) = k_{i,f_j} \times E(\tau_i, f_j, 1)$. The total CPU time is then estimated as $S(\tau_i, f_j, k_{i,f_j}) = k_{i,f_j} \times c_{i,f_j}$.

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 mapping and scheduling phases described below leave processors idle.

C. Mapping and static schedule

The first step provides an initial *configuration* as input to the second step, the mapping and static scheduling onto processors. The initial configuration consists of an assigned frequency and number of replicas for each task instance. Given a configuration, the mapping builds a schedule for an hyperperiod of length $L = \mathrm{lcm}(p_i)$ as follows:
• sort the tasks by decreasing total CPU time; renumber them to have

$$S(\tau_1, f_{j(1)}, k_{1,f_{opt(1)}}) \geq S(\tau_2, f_{j(2)}, k_{2,f_{opt(2)}}) \geq \dots$$

where $f_{opt(i)}$ is the frequency leading to the lowest energy consumption to meet the reliability threshold;
• for i ranging from 1 to n, successively map all $k_{i,f_{opt(i)}}$ copies of τ_i onto $k_{i,f_{opt(i)}}$ different processors, using the First-Fit Decreasing (FFD) bin packing heuristic [4]. When mapping all the $\frac{L}{p_i}$ instances of a given task copy on a processor (bin), we use the standard *Earliest Deadline First (EDF)* scheduling heuristic [5]. EDF tells us that a given processor (bin) is a fit for that copy if and only if the utilization of that processor does not exceed 1. Recall that the utilization of a processor is the sum of the utilizations of all task instances assigned to it.

Hence, for a given task, the mapping finds the first processor whose utilization makes it a fit for the first task copy (and all its instances). Then it finds the first next processor whose utilization makes it a fit for the second task copy (and all its instances), and so on. If the mapping succeeds, we have built a static schedule for the hyperperiod. But it may well be the case that it is impossible to find a processor onto which to map a given task copy in the procedure, because all processor utilizations are too high to accommodate that copy. Then the reference paper proposes to enter an iterative procedure as follows:

- Change the initial configuration into the one where every task copy executes at maximal speed $f_{max} = 1$. This requires fetch the values $k_{i,f_{max}}$ from the table and reordering the tasks by decreasing total CPU time $S(\tau_i, f_{max}, k_{i,f_{max}})$.
- Apply the mapping heuristic (FFD mapping and EDF schedule) to the new configuration.

Now, if the latter mapping fails again, there is no solution, resource utilization is too high. However, if the mapping succeeds, its energy cost may be very high. The reference paper proposes a refinement scheme where some tasks are *relaxed*, meaning that their frequency is decreased down to its predecessor (going from their current value f_j down to f_{j-1}). Initially, all tasks have frequency f_{max}, and some tasks are greedily selected for relaxation. If relaxing a task τ_i fails to lead to a successful mapping, or if we have reached its energy-optimal frequency $f_{opt(i)}$, then τ_i is marked as *ineligible*. The scheme stops when all tasks become ineligible. The reference paper uses three different greedy criteria to pick up the next task to be relaxed among eligible tasks, *Largest Energy First (LEF)*, *Largest Power First (LPF)*, and *Largest Utilization First (LUF)*. We refer to the reference paper for details. We have implemented the first two variants, LEF and LPF, because they are shown to outperform LUF in the reference paper.

D. Dynamic schedule

The static schedule, also called *canonical* schedule, is based upon the WCET of each task and EDF. It is never executed exactly as such, because the actual execution time of a task instance will be shorter than its WCET. Still, it is used as the baseline to guide dynamic updates. From the canonical schedule, each processor has an *assignment list* made up with all task instances that it has to execute during the hyperperiod.

Let us follow the operation of a given processor P. For simplicity, assume that P computes the full EDF schedule for all tasks in its assignment list, during the entire hyperperiod. The reference paper uses a data structure, called Canonical Execution Queue (CEQ), to avoid the high cost of computing the static schedule, while preserving the same outcome. Recall that the EDF schedule uses the WCET of each task, and preemption, so that a given task instance may well be split into several chunks. Hence P has an ordered list of chunks together with their starting and finish times in its EDF schedule.

Let t be the starting time of the next chunk ch, from task instance τ, to be processed by P in its EDF schedule (initially, $t = 0$). In the canonical schedule, this chunk executes in the interval $[t_s, t_f]$, where t_s is the starting time and t_f the finish time on P. Let k be the number of copies of τ in the canonical schedule, distributed over k different processors. A major idea of the reference paper is to differentiate the action of P depending upon whether its own copy of τ is the first copy to start execution among the k processors. So if the chunk ch is indeed the first chunk of any copy of τ to start execution, then P promotes its copy of τ to the status of *primary* replica, and all the chunks of τ as primary chunks. P signals the other $k-1$ processors that their copies (and their copy chunks) are *secondary* replicas (or chunks). Otherwise, the chunk ch has already been marked either as primary (if it is not the first chunk of τ, but τ has been marked primary on P previously) or as secondary (if some other processor has signalled P previously). The action of P is the following:

- if ch is a primary chunk, then P starts its execution immediately, using the frequency given by the canonical schedule. This execution will last for a duration of $t_f - t_s$ seconds in the worst case;
- if ch is a secondary replica, then P executes it at frequency f_{max}, and using ALAP (As Late As Possible) scheduling, as further detailed below.

The rationale for executing secondary chunks at highest frequency f_{max} is that it allows for a minimal execution time, hence a maximal delay for ALAP scheduling. When delaying secondary chunks, we hope that the primary copy will complete before secondary copies actually start, hence will be cancelled whenever the primary copy succeeds. At least, this ALAP strategy should minimize overlap between primary and secondary copies, hence minimize redundant work. We point out that the choice for primary/secondary replicas is done dynamically, by the first processor to start a task instance in the actual execution of the hyperperiod. It may well be the case that two different instances of the same task have not the same primary processor.

There remains to explain in full details how secondary chunks are scheduled on P. In fact, all secondary chunks are delayed according to the finish time of τ in the canonical schedule. Recall that the canonical schedule provides an execution interval $[t_s, t_f]$ for every chunk of τ. Assume there are m chunks. For notational convenience, let $[t_s(i), t_f(i)]$, $1 \leq i \leq m$ denote the m execution intervals in the canonical schedule. The sum of these m interval lengths $t_f(i) - t_s(i)$, $1 \leq i \leq m$ is equal to $\frac{c}{f}$, where c is the WCET of τ and f is its frequency from the canonical schedule. Since we have decided to execute τ at frequency $f_{max} = 1$, we only need c seconds, in the worst-case, to execute τ. For instance, if $f = 0.5$, we only need (at most) half the time planned in the canonical schedule. The reference paper uses backfilling and reserves a total of c seconds in the dynamic schedule, starting from the last interval and going backwards to the first interval, allocating slots until c seconds are reserved. Then P will execute τ greedily using these intervals from the beginning until completion of τ. Finally, all chunks of τ will be scheduled across n new intervals $[t'_s(i), t'_f(i)]$, where $1 \leq i \leq n$ and $n \leq m$, because very likely τ will finish before its WCET. Here are two examples with $c = 20$, $f = 0.5$ and $m = 3$:

- the intervals in the canonical schedule are $[5, 35]$, $[40, 46]$ and $[50, 54]$. We reserve 20 seconds out of the 40 available in these three intervals by keeping the third interval entirely (4 seconds), then keeping the second interval entirely (6 seconds) and then keeping the last 10 seconds of the first interval. Then P will use these reserved slots ($[25, 35]$, $[40, 46]$ and $[50, 54]$) to execute its copy of τ at frequency f_{max} starting from time 25.
- the intervals in the canonical schedule are $[10, 18]$, $[40, 46]$ and $[50, 76]$. We reserve 20 seconds out of the 40 available in the three intervals by just keeping the last fraction $[56, 76]$ of the third interval (20 seconds), and leaving the first and second intervals empty.

In both cases, P will consume the slots from the beginning of the first reserved interval, until consuming all time units needed to finish τ, say at time t' ($t' \leq t'_f(n)$). Once the copy of τ on the processor P is successfully executed, all other copies (chunks) of τ are removed from the EDF list on other processors. Then P will start processing the next chunk ch' in its list right at time t' if ch' is primary, otherwise, the chunks are delayed using the same mechanism.

To summarize, consecutive primary replicas are scheduled ASAP (As Soon As Possible), while secondary replicas are scheduled ALAP, which aims at reducing the overlap between copies over different processors, so as to minimize energy consumption.

III. Motivational Example

We illustrate one limitation of the strategy in [1] with a simple example with two tasks τ_1 and τ_2 and two processors P_1 and P_2. Assume that: (i) for WCET, $c_1 = c_2 = 0.5$ at maximum frequency $f_{max} = 1$; (ii) for periods $p_1 = 2$ and $p_2 = 3$; (iii) τ_1 and τ_2 have been scheduled to run at frequency $f = 0.5$, which takes 1 time unit; and (iv) the reliability threshold is met with one replica per processor for each task. For simplicity, also assume that: (i) each instance of τ_1 is a primary replica on P_1 and secondary on P_2, while each instance of τ_2 is a primary replica on P_2 and secondary on P_1; and (ii) there is no difference between actual execution times and the WCETs.

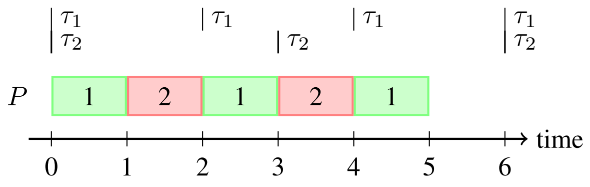

Figure 1: Canonical Execution Queue (CEQ) with two tasks τ_1 and τ_2 ($c_1 = c_2 = 0.5$, $p_1 = 2$ and $p_2 = 3$) on each processor, at frequency $f = 0.5$.

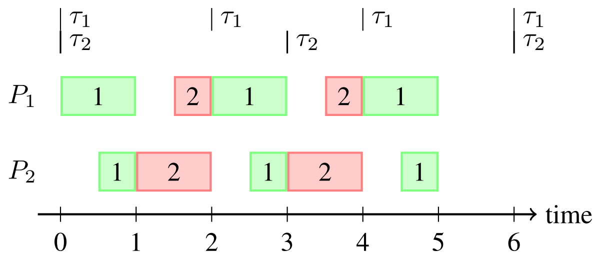

Figure 2: Executions when prioritizing primaries while delaying secondaries at maximum frequency as in the Canonical Execution Queue (CEQ).

Figure 1 shows the CEQ (same for both processors) for the example. Figure 2 shows the execution on both processors with a CEQ-based strategy as described in Section II-D. Recall that for secondary replicas, the strategy delays their executions according to CEQ and runs them at f_{max}. In the example, there is a complete overlap between each primary and each secondary replica due to the constraint given by the CEQ. This is wasteful because the secondary replica can never be canceled whenever the primary replica succeeds. However, it would be possible to avoid any overlapping by advancing each primary replica while delaying each secondary replica. In the example, this simply means interchanging τ_1 and τ_2 on P_2. Without overlap, the executions of all secondary replicas at maximum frequency will be cancelled most of the time, leading to a substantial energy gain.

IV. New strategies

We identify several possible reasons why the approach in the reference paper may be sub-optimal:

- All optimizations in the dynamic schedule aim at reducing overlap among replicas, so as to avoid redundant work. In their final schedule, the primary replica of task τ_i is executed at assigned frequency f_j, but the frequency of all $k_{i,f_j}-1$ secondary replicas is increased to f_{max}. However, energy consumption is estimated with all replicas at frequency f_j. We revisit the estimate for energy consumption and use a different formula, with one copy executing at f_j, and the rest at f_{max}: this is expected to be closer to actual execution scenarios;

- The mapping uses the *First-Fit Decreasing (FFD)* bin-packing heuristic, which is likely to create imbalance across processors. Instead, we use the *Worst-Fit Decreasing (WFD)* bin-packing heuristic [6], which selects the least-loaded processor that is a fit for the current task copy. WFD has been shown to reduce imbalance in a related framework [7]; we use it with a similar motive;
- The mapping maps all copies of a task before proceeding to the next task. Instead, we map the first copy of each task, and then the second copy of each task (whenever it exists), and so on. This *layered* approach is expected to: (1) evenly map primary replicas onto processors; (2) decrease the overlap among copies of the same task as long as EDF priority constraints do not call for a full reordering of the tasks during (or after) the mapping. For instance, assume for simplicity that all tasks have the same period, so that the schedule is not constrained by EDF. Then, the mapping of the reference paper will place all replicas of the first task at the beginning of the assignment list of the processors, with possibly more primary replicas on the first processors, while it is better to delay all copies but one and insert first copies of other tasks instead.
- For the dynamic schedule, we keep the idea of running one primary replica of an instance of task τ_i at assigned frequency f_j and to execute all other replicas at frequency f_{max}, but we implement several novel aggressive strategies to reorder and delay chunks, as described in Section IV-C.

We outline these modifications step by step below, in Sections IV-A, IV-B and IV-C, which are the respective counterpart, with the same title, of Sections II-B, II-C and II-D. Section IV-D summarizes our new heuristics. Note that we have both online and offline scheduling heuristics. For online scheduling strategies, which are able to determine dynamically the primary copy and secondary copies on the fly, all copies are mapped onto processors using frequency f_j: this is because we must reserve enough room for each copy in the mapping. In other words, we assume that each copy is a primary copy in the mapping, and some of them become secondary only during execution. On the contrary, for offline scheduling strategies, we have decided primary copy and secondary copies during the mapping phase, which means we map one replica at f_j and the rest at f_{max}. In both scenarios, recall that we use different formulas to estimate the energy cost in both cases. Finally, Section IV-E is devoted to complexity results.

A. Replica sets

For task τ_i at frequency f_j, the reference paper determines the number of copies k_{i,f_j} needed to match the reliability threshold $\mathcal{R}_i$ and estimates the energy cost as $k_{i,f_j}E(\tau_i, f_j, 1)$ where $E(\tau_i, f_j, 1)$ is the energy cost for a single copy. Instead, we propose to estimate the energy cost as:

$$E(\tau_i, f_j, k_{i,f_j}) = E(\tau_i, f_j, 1) + (k_{i,f_j}-1)E(\tau_i, f_{max}, 1) \quad (1)$$

because secondary replicas are actually executed at f_{max} in the reference paper. Equation (1) is an accordance with the pessimistic scenario where no replica is cancelled.

Note that since we do not use the same estimation formula for the energy cost as in the reference paper, we may find a different frequency $f_{opt(i)}$ to be used for the mapping step. Again, only one copy (the primary copy) will actually be executed at frequency $f_{opt(i)}$, while all remaining copies (the secondaries) will be executed at frequency f_{max}. For each task τ_i and a given primary frequency f_j, we determine the minimum number of replicas r_i such that the reliability threshold $\mathcal{R}_i$ for each instance is met. Recall that the reliability using a single task instance (no replica) is $R_i(f_j) = \exp^{-\lambda(f_j)c_{i,f_j}}$. If $R_i(f_j) \geq \mathcal{R}_i$, the reliability threshold is enforced, and we need no replica, hence $k_{i,f_j}=1$. Otherwise, using r additional replicas at f_{max}, the reliability increases to $1-(1-R_i(f_j))(1-R_i(f_{max}))^r$, instead of $1-(1-R_i(f_j))^{r+1}$, and we take the minimal value of r such that this reliability exceeds $\mathcal{R}_i = \sqrt[h_i]{1-\omega(1-R_i(f_{max})^{h_i})}$. This leads to $r = \left\lceil \frac{\log(\frac{1-\mathcal{R}_i}{1-R_i(f_j)})}{\log(1-R_i(f_{max}))} \right\rceil$. The new value of r may be smaller than before, because each replica is more reliable. Of course, the new value of r leads to a new value of $k_{i,f_j} = r+1$. We use this new value of k_{i,f_j} to compute the energy cost. Note that the reference paper assumed that task WCETs are inversely proportional to the assigned frequency where $c_{i,f_j} = \frac{c_i}{f_i}$, but in reality, the interplay between task execution time and frequency is complicated [8]. To be closer to real cases, we let WCETs obey a randomized speedup function similar to Amdhal's law: specifically for task τ_i at frequency f_j, $c_{i,f_j} = s_i c_i + (1-s_i)\frac{c_i}{f_j}$, where the sequential fraction s_i is drawn uniformly and randomly in the interval $[0.1, 0.3]$.

We retain the frequency $f_{opt(i)}$ that minimizes Equation (1), and we use $k_{i,f_{opt(i)}}$ copies of task τ_i all running at frequency $f_{opt(i)}$ as input to the mapping phase for all online scheduling strategies. On the contrary, for offline scheduling strategies, we use one copy at frequency $f_{opt(i)}$ and $k_{i,f_{opt(i)}}-1$ copies at f_{max} as we explained before. Note that we tried through all possible frequencies between f_{min} and $f_{max}=1$, but the final number of valid frequency levels may be smaller than the number of available frequency levels. There are several possibilities: 1) We should not consider lower frequency levels that makes the task τ_i miss its deadline. 2) As we decrease the frequency level, the number of required replicas may increase or remain the same, the energy consumption is not strictly decreasing. 3) A lower frequency may introduce more overlap that can not be avoided than a higher frequency, so we enforce that $c_{i,f_j} + c_{i,f_{\max}} \le p_i$.

B. Mapping and static schedule

We map each replica to a processor while ensuring that no two replicas of the same task are assigned to the same processor. The mapping is done for a whole hyperperiod, with the following constraint for each task: when the first iteration (in case the period differs from the hyperperiod) of a replica is assigned to a given processor, all the other iterations of the same replica will be assigned to the same processor. It allows a simple feasibility check based upon the utilization of all the replicas assigned to the processor.

Recall that any success of a primary replica leads to the immediate cancellation of the secondary replicas, a crucial source of energy saving. The objective of the proposed mapping is thus to avoid overlapping between the execution of the primary and secondary replicas for each task: the primary must be terminated as soon as possible, while the secondaries must be delayed as much as possible. To this end, the mapping strategy ventilates primaries on all processors, in order to minimize conflict among several primary executions. Moreover, it allocates the secondaries while leaving idle time on all processors. This slack can then be used at execution time to delay the execution of the secondaries.

Algorithm 1: Mapping (WFD layer by layer)

```
Input: The WCET $c_i$, the period $p_i$, the assigned frequency $f_i$, the
    number of secondary replicas $r_i$, the number of instances $h_i$
Output: An allocation of all replicas on the processors $\sigma_m$
1  begin
2      execute WFD on the primaries considering non-increasing order
         of $c_{i,f_i} \times h_i$
3      if will do online scheduling afterwards then
4          $f_i \leftarrow f_i$
5      else
6          $f_i \leftarrow f_{\max}$
7      execute WFD on the secondaries layer-by-layer considering
         non-increasing order of $c_{i,f_i} \times h_i$ while ensuring that no
         processor executes more than one replica of each task
8      if $\exists m, \sum_{\tau_i \in \sigma_m} \frac{c_{i,f_i}}{p_i} > 1$ then
9          return not feasible
10     else
11         return $\sigma_m$
```

As shown in Algorithm 1, given a list of primary and secondary replicas for each task, and their execution times, we first execute the Worst-Fit Decreasing (WFD) allocation on the primaries ordered by their total execution time (Line 2). For all secondaries, we assign the same frequency as the primary if preparing for online scheduling afterwards. We assign them the maximum frequency f_{max} otherwise (Lines 3-6). Then we execute WFD on the secondaries layer-by-layer and ordered by their total execution time (Line 7), which means we consider the first secondary of all tasks, then the second and so on (as long as it exists). If we successfully map all replicas onto the processors, and the sum of the utilization of each replica on each processor is less than or equal to one (see Section II-C), then we return this allocation. Otherwise, we could not find a feasible mapping with this replica setting (Lines 8-11). We observed that the competitor strategy FFD in the reference paper tends to compact all replicas onto the minimum number of processors, while WFD spreads replicas among all available processors, which may give a higher static energy cost. To reduce the influence of static power, we first run WFD with the number of processors used by FFD. If WFD is not able to find a feasible mapping, then we increase the number of processors by one up to the total available number.

Algorithm 2: Replication setting

```
Input: A set of tasks with cost $c_i$ and reliability requirement $\mathcal{R}_i$
Output: A set of minimum frequency $f_i$ and number of secondary
    replicas $r_i$
1  begin
       /* start with all primaries at
          energy-optimal frequency                    */
2      for $i \in [1, \dots, n]$ do
3          $f_i \leftarrow f_{opt(i)}$
           /* reliability requirement                 */
4          $r_i \leftarrow \left\lceil \frac{\log(\frac{1-\mathcal{R}_i}{1-R_i(f_i)})}{\log(1-R_i(f_{max}))} \right\rceil$
5      map the tasks to the processors with Algorithm 1
6      if feasible then
7          return $\{f_i, r_i\}$
       /* reset all primaries at $f_{\max}$             */
8      for $i \in [1, \dots, n]$ do
9          $f_i \leftarrow f_{\max}$
           /* reliability requirement                 */
10         $r_i \leftarrow \left\lceil \frac{\log(\frac{1-\mathcal{R}_i}{1-R_i(f_i)})}{\log(1-R_i(f_{max}))} \right\rceil$
11     map the tasks to the processors with Algorithm 1
12     if not feasible then
13         return does not exist a feasible mapping
       /* enter the relaxing phase                    */
14     while any primary frequency can be decreased do
15         select task $i$ with LEF or LPF criteria
16         $f_i \leftarrow f_{i-1}$
17         $r_i \leftarrow \left\lceil \frac{\log(\frac{1-\mathcal{R}_i}{1-R_i(f_i)})}{\log(1-R_i(f_{max}))} \right\rceil$
18         map the tasks to the processors with Algorithm 1
19         restore $f_i$ and mark task $i$ can not be further decreased if
             mapping is not feasible
20     return $\{f_i, r_i\}$
```

We use this allocation mechanism to determine the number of replicas and their frequencies for each task (see Algorithm 2). As described in Section IV-A, we already know for each task, at each frequency level, how many replicas are needed to meet the reliability threshold. If we can find a feasible mapping with each task at its energy-optimal frequency $f_{opt(i)}$, then we return this optimal setting (Lines 2-7). Otherwise, we check the other end, all tasks run at f_{max} that takes the shortest time possible. If it is still impossible to map all replicas, then there does not exist a feasible mapping (Lines 8-13). If there exists a feasible mapping, we will enter the relaxing phase that decreases each primary frequency f_i iteratively until it is no longer possible (Lines 14-19). Finally, we return the solution with a frequency level and number of secondary replicas for each task (Line 20).

C. Dynamic schedule

For the scheduling phase, it is important to start primary replicas as soon as possible, and to delay secondary replicas as much as possible to minimize the overlapping, while still meeting all deadlines. As explained in Section II-D, the reference paper uses the canonical schedule to compute the maximum delay for secondary replicas. Our improvements rely on the following techniques:

- Consider a *scheduling interval* defined by two consecutive deadlines in the global schedule. Inside the interval, task chunks to be executed are ordered by the EDF policy. We observe that we can freely reorder the chunks without missing any deadline, by definition of an interval. It means that in each interval, we should reorder to execute all primary replicas first, and then secondary replicas.
- It is possible to use only a fraction α of each scheduling interval, where α is the utilization. Here is why: at the mapping phase, as long as the total utilization of replicas that are mapped onto the processor is less than or equal to one, then we are able to find a valid scheduling using EDF. Assume we have mapped three tasks t_i, t_j, t_k onto processor p, and that the utilization is $\alpha = \frac{c_{i,f_i}}{p_i} + \frac{c_{j,f_j}}{p_j} + \frac{c_{k,f_k}}{p_k}$. Either we keep the mapping and have a fraction $1-\alpha$ of the interval where p is idle, or we slow down the execution time of all three tasks by a factor α, then we will have a new utilization $\beta = \frac{c_{i,f_i}}{p_i\alpha} + \frac{c_{j,f_j}}{p_j\alpha} + \frac{c_{k,f_k}}{p_k\alpha} = 1$, which also gives us a feasible mapping without any idle time. This idea can be used in two ways:
  1) Schedule while keeping a fraction $1-\alpha$ of idleness in each interval. Then, each primary replica is pushed to be beginning of the interval, while secondaries are pushed back to the end of the interval, with idleness in between.
  2) Scale the WCET of all tasks by $\frac{1}{\alpha}$, which also gives a valid canonical schedule, but with longer worst case expected execution time for all tasks. This gives a better reference to further delay the start time of secondary replicas.
- Because we have delayed the start time of secondary replicas, there are some idle slots in the schedule. We take advantage of these idle slots by pre-fetching other primary replica chunks in the availability list: those primaries have been released but were scheduled later because they have lower EDF priority than the current secondary replicas.

D. Heuristics

Based on the above ideas, we propose several new scheduling heuristics which improve upon EDF_PAPER, the adaptive dynamic scheduling (and the most efficient) heuristic of the reference paper.

EDF_PAPER_PF is an adaptive online scheduling that simply adds the pre-fetching mechanism to EDF_PAPER.

EDF_PAPER_PF_UTILITY is an online scheduling heuristic where we refine EDF_PAPER_PF by using the utilization of each processor. We scale the worst case execution time of all replicas of a given processor by a factor $\frac{1}{\alpha}$, where α is the utilization of that processor.

EDF_IDLE_CEQ is an offline scheduling that builds the EDF schedule for the whole hyperperiod. In each interval defined by two consecutive deadlines, we only use (on each processor) a fraction of the interval defined by the static utilization. It consists in the following steps:

1) consider at each interval the EDF schedule with the constraint of keeping a fraction $1-\alpha$ of idle time. For secondaries, we refer to the canonical schedule to delay its start time without missing any deadline;
2) start the primary replicas and put aside the secondary replicas in a waiting list to be executed at the end of the interval. Note that for each secondary, as it is impossible to know its actual execution time before its execution, we need to reserve the space for its WCET and to finish execution within the interval;
3) fill in the idle period by inserting other primary replicas that are available;
4) finish the execution of the interval with the secondary replicas in the waiting list, with their actual execution time.

EDF_IDLE_CEQ_ONLINE is the online version of EDF_IDLE_CEQ. It has two major advantages compared to the offline version: (1) we can dynamically decide the primary copy of each task instance, which gives us the flexibility to speed up replicas on the fly; (2) as long as we finish one replica successfully, we can safely cancel other replicas of the same task instance earlier than in static schedules, which gives us more flexibility to adjust the schedule afterwards. Moreover, all algorithms have to reserve for the secondaries some time slots corresponding to their WCET. As their actual execution times are usually shorter, this dynamically frees some time slots that the online schedule uses to prefetch available primary replica chunks.

E. Complexity analysis

This section is devoted to the proof of several complexity results for the scheduling phase. The global optimization problem is obviously NP-hard, since it is a generalization of the makespan minimization problem with a fixed number of parallel processors [9]. However, the complexity of the sole scheduling phase is open: if the number or replicas has already been decided for each task, and if the frequency and assigned processor of each replica has also been decided, the scheduling phase aims at minimizing the expected energy consumption. We state a lower bound for this scheduling problem in Section IV-E1, and we assess the complexity of achieving this lower bound: in Section IV-E2, we show that the instance with identical WCETs is polynomial, while in Section IV-E3, we show the instance with different WCETs is NP-complete in the strong sense.

1) Lower bound: Consider the following instance of the scheduling phase:

- All tasks have the same period p, hence there is a single instance of each task in the hyperperiod of length $L = p$. hence EDF constraints do not apply, and each task is scheduled without preemption (as a single chunk)
- $F = 1$: there is a unique frequency $f_{max} = 1$
- For $1 \leq i \leq n$, task τ_i has k_i replicas, including itself. The j-th replica, with $1 \leq j \leq k_i$, is mapped onto processor $P_{alloc(i,j)}$, where$1 \leq alloc(i,j) \leq M$. For each task τ_i, replicas are mapped onto different processors: $alloc(i, j_1) \neq alloc(i, j_2)$ for $1 \leq j_1 < j_2 \leq k_i$.
- The WCET of any replica of task τ_i is c_i, its reliability is $R_i = R_i(f_{max})$, and its consumed energy is $E_i = E(\tau_i, f_{max}, 1)$.
- $P_{static} = 0$, meaning that no energy is spent when a processor is idle

Thus, each of the M processors has a list of assigned replicas to execute. It can choose any ordering because all tasks have the same period, hence all deadlines will be enforced, regardless of the ordering. We further assume that the mapping is valid, which translates on each processor P_q, $1 \leq q \leq M$, by the condition:

$$\sum_{1 \leq i \leq n, 1 \leq j \leq k_i, alloc(i,j)=q} WCET_i \leq p$$

Proposition 1. *A lower bound on the total expected energy consumed is*

$$E_{LB} = \sum_{i=1}^{n} \left(E_i \sum_{j=1}^{k_i} (1 - R_i)^{j-1} \right) \tag{2}$$

The bound E_{LB} is met if and only if the scheduling achieves no overlap between any two replicas of the same task.

Proof. For each task, we need to execute the replica which is scheduled in first position. If this replica fails, with probability $1 - R_i$, we need to execute the replica which is scheduled in second position. If both replicas fail, we need to execute the replica which is scheduled in third position, and so on. This directly leads to the lower bound E_{LB}.

Now if any two replicas of the same task, say τ_i, do overlap, then with some non-zero probability, both replicas will execute, and the consumed energy will be strictly higher than the contribution of τ_i to Equation (2). This concludes the proof. □

2) Identical costs:

Proposition 2. *When all tasks have the same WCET ($c_i = c$ for $1 \leq i \leq n$), one can build a schedule meeting the lower bound of Equation (2) in polynomial time.*

Proof. We construct a bipartite graph with tasks on the left (with n vertices) and processors on the right (with M vertices). Task τ_i has k_i edges, connecting each of its replicas to the assigned processor. Hence we have $K = \sum_{i=1}^{n} k_i$ edges. Let δ be the maximum degree of a vertex in the graph. According to Konig's edge coloring theorem, one can find a collection of δ perfect matchings that cover all edges in the graph, in time $O(\delta K)$ [10]. Since $\delta K \leq (n+M)^3$, this is indeed polynomisal in the problem size. These mappings directly lead to a schedule with minimal makespan δc. By construction, this schedule is guaranteed overlap-free. □

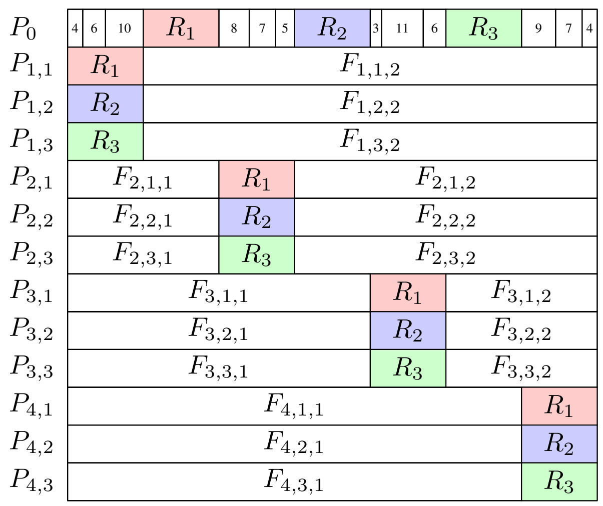

Figure 3: Scheduling for a solution of $\mathcal{I}$ of 3-Partition, with $m = 4$, $B = 20$, and $(a_1, ..., a_{12}) = (4, 3, 8, 9, 7, 6, 11, 6, 7, 10, 4, 5)$. On processor P_0 the digits are the sizes of the A_i's.

3) Arbitrary costs:

Proposition 3. *When tasks have different WCET, determining whether the lower bound of Equation* (2) *can be met, is a problem which is NP-complete in the strong sense.*

Proof. Let NoOverlap denote the problem with different WCETs. NoOverlap clearly belongs to the class NP: a certificate can be the description of the schedule with start and end times for each replica, and one can check in quadratic time that no two replicas of the same task overlap. We establish completeness in the strong sense through a reduction from 3-Partition [9]. Let $\mathcal{I}$ be an instance of 3-Partition. $\mathcal{I}$ comprises $3m$ integers, $a_1, ..., a_{3m}$ such that $\sum_{i=1}^{3m} a_i = mB$ and $\frac{B}{4} < a_i < \frac{B}{2}$ for all i. The question is: can we partition the a_i's into m subsets $S_1, ..., S_m$ such that each subset has total size B: $\sum_{j \in S_i} a_j = B$? The size of $\mathcal{I}$ is $O(m + B)$. From the instance $\mathcal{I}$ of 3-Partition, we build an instance $\mathcal{J}$ of NoOverlap: this instance contains three types of tasks: some replicated tasks (the R_i's), some filling tasks which constrain the replicated tasks (the $F_{i,j}$'s), and the tasks corresponding to the integers in instance $\mathcal{I}$ (the A_i's). Specifically:

- There are $1 + m(m - 1)$ processors denoted P_0 and $P_{i,j}$ with $1 \leq i \leq m$ and $1 \leq j \leq m - 1$.
- All tasks have the same period $p = (2m - 1)B$.
- There are $m - 1$ replicated tasks of size B, $R_1, ..., R_{m-1}$. For any i, $1 \leq i \leq m - 1$, there is one replica of R_i on processor P_0, and 1 on each of the processors $P_{k,i}$, $1 \leq k \leq m$. Thus, each R_i's is replicated $m + 1$ times.

- Two tasks $F_{i,j,1}$ and $F_{i,j,2}$ are mapped on each processor $P_{i,j}$, $1 \leq i \leq m-1$, $1 \leq j \leq m$. $F_{i,j,1}$ is of size $2(i-1)B$ and $F_{i,j,2}$ is of size $2(m-i)B$. The total load of processor $P_{i,j}$ is $B + 2(i-1)B + 2(m-i)B = (2m-1)B$ and there is no slack on that processor. Note that, to ease the writing, we have kept in our description a null size task on processors $P_{i,1}$ and $P_{i,m}$.
- In addition to one replica of each of the tasks R_1, R_{m-1}, $3m$ tasks $A_1, ..., A_{3m}$ are mapped to processor P_0, where task A_i has size a_i. Therefore, the total load of processor P_0 is $(m-1)B + \sum_{i=1}^{3m} a_i = (2m-1)B$ and there is no slack on that processor either.

Instance $\mathcal{J}$ contains $1+m(m-1)$ processors and $(m-1)(m+1) + 2m(m-1) + 3m = 3m^2 + 3m - 1$ replicas. All tasks have size $O(mB)$. Hence, the size of $\mathcal{J}$ is polynomial in the size of $\mathcal{I}$. We now prove that if $\mathcal{I}$ has a solution, then $\mathcal{J}$ has a solution. Let $S_1, ..., S_m$ be the solution of $\mathcal{I}$. Then we schedule the tasks of $\mathcal{J}$ as follows and as illustrated by Figure 3:

- For any i, $1 \leq i \leq m-1$, and any j, $1 \leq j \leq m$, on processor $P_{i,j}$, task $F_{i,j,1}$ is executed during the interval $[0, 2(i-1)B]$, a replica of R_j during the interval $[2(i-1)B, (2i-1)B]$, and $F_{i,j,2}$ during the interval $[(2i-1)B, (2m-1)B]$.
- For any i, $1 \leq i \leq m-1$, a replica of R_i is executed on P_0 during the interval $[(2i-1)B, 2iB]$.
- For any j, $1 \leq j \leq m$, the tasks corresponding to the j-th partition of $\mathcal{I}$, i.e., the tasks A_k such that $k \in S_j$, are executed on P_0 during the time interval $[2(j-1)B, (2j-1)B]$.

One can easily check that this schedule is valid and that two replicas of a task R_i are never executed simultaneously. Therefore there exists a schedule without overlap for $\mathcal{J}$ if there exists a solution for $\mathcal{I}$.

Let us now assume that there exists a valid schedule for $\mathcal{J}$, i.e., a schedule without any overlap. Let us consider any i, $1 \leq i \leq m-1$. Processor $P_{i,1}$ (respectively $P_{i,m}$) contains a replica R_i and a task of size $(2m-2)B$ (we do not care about the null-size task). Therefore, the replica R_i is executed on $P_{i,1}$ (resp. $P_{i,m}$) either during the interval $[0, B]$ or during $[(2m-2)N, (2m-1)B]$, Then, for any j, $1 < j \leq \frac{m}{2}$, processor $P_{i,j}$ (respectively $P_{i,m-j+1}$) contains a replica R_i, a task of size $(2i-1)B$ and one of task $2(m-i)B$. From what precedes, there is already a replica of R_i executed during the time interval $[0, B]$ and one during $[(2m-2)B, (2m-1)B]$. Therefore, the replica R_i is executed on $P_{i,j}$ (resp. $P_{i,m-j+1}$) either during the interval $[(2i-1)B, 2iB]$ or during $[(2m-i)B, (2m-i+1)B]$. Overall, on processor P_0, the replica R_i must be executed during one of the intervals $[(2j-1)B, (2j)B]$, for $1 \leq j \leq m-1$, because at all the other instants, there is already one R_i replica being executed on one other processor, and because the schedule is without any overlap. However, there are $m-1$ such intervals and $m-1$ such replicas. Therefore, the tasks A_i's must be executed during the intervals $[(2j-2)B, (2j-1)B]$, for $1 \leq j \leq m$. This is a set of m intervals each of size B. Because the schedule is valid, all the A_i's are executed during these intervals. Let S_j, $1 \leq j \leq m$, be the set of the indices of the A_l's executed during $[(2j-2)B, (2j-1)B]$. Then the subsets S_j define a solution to $\mathcal{I}$. $\square$

V. Performance evaluation

In this section, we present the simulation results to evaluate the performance of our whole strategy compared to the best solution proposed in the reference paper [1]. In Section V-A, we describe the parameters and settings used during the experimental campaign. We present the results in Section V-B.

A. Experimental methodology

We designed a discrete event simulator, which is publicly available at [11]. For each data point, we considered 2000 data sets with 20 tasks. Task periods are randomly generated between 10ms and 100ms. The utilization of each task is generated randomly using the UUnifast scheme [12], with the total utilization U_{tot} as input. The set of frequencies is taken from a real microprocessors [13] which has $F = 5$ frequency levels including 1.0, 0.8, 0.6, 0.4 and 0.15. We also validate our methods on two other real frequencies sets [14], [8]. Following [1], in the rest of the section, we assume that the transient fault arrival rate at f_{max} is $\lambda_0 = 10^{-6}$ and the system sensitivity factor $d = 4$. The static power and the frequency-independent are set to 5% and 15% respectively of the maximum frequency-dependent power consumption with $C = 1$. We use the ratio BC/WC of the best-case (BC) over worst-case (WC) execution time, to model workload variability. The actual execution time of each task is determined according to a uniform distribution between BC and WC. To compare the strategies under all parameter settings, we covered all of the variables from [1]. For the rest of the section, we keep the number of cores at 8, vary the value of U_{tot} from 1.5 to 3.5 to study the impact of system load, vary BC/WC from 0.2 to 1.0 to show the influence of workload variability, and vary the probability of failure scaling factor w from 10^{-5} to 10^{-1} to evaluate the impact of target reliability. Due to space limitations, we only report here a subset of our simulations results. For instance, we consider heuristic LPF for choosing the candidate task for relaxation, which is shown to be the best heuristic in [1]. However, a full set of results supplemented with two additional real frequencies sets and a larger range s_i, $s_i \in [0.1, 0.5]$ can be found in [15].

B. Results

We use the same *baseline* scheme as the reference paper[2], i.e., classic EDF scheduling with First-Fit Decreasing mapping and replica sets from [1] (where it is called the *static scheme*). Figures 4 and 5 present the energy consumption of our strategies and of the best performance strategy from the reference paper, divided by the energy consumption of the baseline. Therefore, the lower the better and data points below the $y = 1$ line denote cases in which these strategies

[2]The authors of [1] have not provided their source code to us; we did our best to ensure a fair assessment and comparison of results.

outperform the static scheme (i.e., achieve a lower energy consumption). These figures represent the trends with various parameters. See [15] for the full set of results.

Each subfigure shows results for a different combination of mapping and replica settings. Each line of subfigures is for a different replica sets (either from the reference paper or our new set) while each column is for a different mapping (FFD and WFD). For example, the bottom right plot presents energy savings of several scheduling approaches (different line colors) with WFD mapping and our replica setting. We report in these figures the number of seeds (out of 2000 in total) that could find a feasible solution for each setting. These numbers are reported in black above the horizontal axis in each figure. Note that WFD tends to find fewer or the same number of feasible solutions than FFD. This is because FFD tries to pack more tasks onto processors while WFD tends to spread tasks onto less loaded processors. Moreover, we plot the lower bound for online scheduling and offline scheduling (different line styles) in black lines, by which we could know the maximum energy saving that can be achieved without any overlapping and failure. We can notice that when we apply our replica setting, the lower bound of the offline scheduling is sometimes slightly lower than that of the online scheduling. The reason is that during the mapping phase, in the online scheduling, we assign all secondaries the same frequency as the primary, which makes it more difficult to find a feasible mapping than in the offline scheduling that assigns secondaries the maximum frequency. Hence, the offline scheduling may pick a lower frequency (see Section IV-B).

A clear observation is that our scheduling heuristics outperform EDF_PAPER under almost all combinations of settings. In particular, EDF_IDLE_CEQ_ONLINE always achieves the best performance. EDF_IDLE_CEQ_ONLINE could save up to 25% with respect to EDF_PAPER (see Fig. 4c). It should be noted that to clearly see the different capabilities of the scheduling heuristics in reducing the overlapping, we need more than one replica for each task. As we decrease the value of w, more replicas are needed to meet the reliability target. In Figure 4, we keep the value of BC/WC at 1 and increase the system load. We observe that the energy savings are closer to the lower bound at lower utilization, as we can find a feasible partitioning without overlapping for the minimum energy configurations. In Figure 5, for a fixed system load ($U_{tot} = 2.5$), the higher the BC/WC ratio, the lower the workload variability, and the further from the lower bound. This is because as BC/WC increases, jobs have larger execution times and there is a higher chance of overlap between the primary and the secondary replicas.

Another common trend is that, in the majority of cases, WFD and our new replica setting help save energy. By considering our replica setting, we could find a lower $f_{opt(i)}$. This is because our replica setting plans for one replica at f_i and x replicas at f_{max} while the reference setting plans to run all replicas at f_i. By considering WFD, we usually: 1) achieve the highest energy savings, 2) improve the average performance of scheduling heuristics. For example, in subfigure 4a, from the two plots in the second row (with the same replica set but with different mappings), we can see that more scheduling heuristics achieve the lower bound with WFD than with FFD (with whom only EDF_IDLE_CEQ_ONLINE did). In subfigure 4d, we could see that applying the replica sets and mapping of the reference paper (top left plot), the best energy savings are achieved by EDF_IDLE_CEQ_ONLINE (around 37.5%), while the other heuristics save less that 25%. With our replica sets and mapping (bottom right), our scheduling heuristics achieve more than 40% energy savings. Even the EDF_PAPER saves up to 25%. Thus, our solutions outperform those of the reference paper. These improvements are representative of the trends that can be observed for all considered graphs, but suffer from some exceptions. For example, in Fig. 4a, when $w = 10^{-5}$, applying our replica set gives a higher lower bound which means we will lose even without any overlapping in the schedule. Overall, our task mapping and replica setting never achieves significantly bad performance, and most of the time achieves the best performance. The average improvement of our method over [1], computed across all our experiments, can be estimated to be of the order of 20%.

VI. RELATED WORK

Liu and Layland first introduced the Earliest Deadline First (EDF) and the Rate Monotonic (RM) scheduling policies for real-time systems and provided the utilization bounds for both policies in 1973 [5]. Since then, the real-time scheduling problem has been extensively studied. A significant amount of work has aimed at developing new scheduling algorithms under different assumptions, including the well-known energy management technology Dynamic Voltage and Frequency Scaling (DVFS). We refer to the original article [1] for the related work. Here, we only cover relevant work citing paper [1].

In [16], Taherin et al. propose an approach for energy management that is only applied on "low-criticality tasks in low-criticality mode to preserve the original reliability of the system". This approach cannot guarantee that a reliability threshold is met. Keeping the maximum power consumption below the chip thermal design power, Ansari et al. [17] have proposed a peak power management approach to meet thermal design power in fault-tolerant system. However, in the scheduling task graph, all tasks have the same period/deadline. Cao et al. in [18] proposed an affinity-driven modeling and scheduling approach to makespan optimization. They optimize average peak temperature and makespan, but not energy minimization. To improve quality in real-time system, Cao et al. in [19] proposed QoS-adaptive approximate real-time computation optimization. Approximate results of tasks are allowed: each task is composed of a mandatory part and an optional one that refines the result of the mandatory task. However, failures are not considered. Zhou et al. consider in [20] both transient and permanent faults. They try to improve soft-error reliability while satisfying a lifetime reliability constraint, but do not attempt to minimize energy consumption.

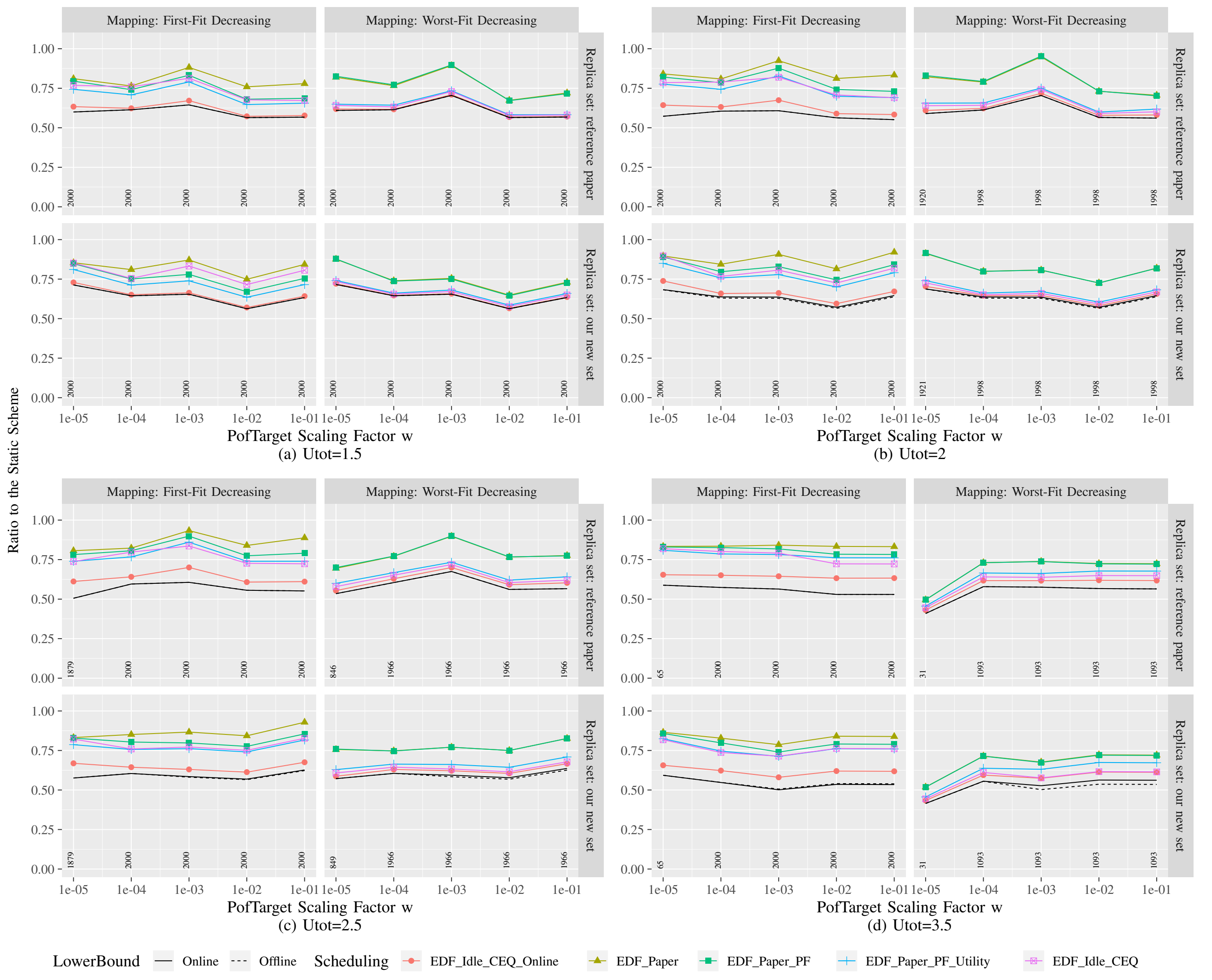

Figure 4: Impact of U_{tot} and w with $BC/WC = 1$.

VII. Conclusion

In this work, we have revisited the challenging problem presented by Haque, Aydin and Zhu in [1], namely minimizing the expected energy consumption of a set of preemptive periodic real-time tasks, executing on a parallel platform where processors are subject to transient failures. Replication is used to enforce all deadlines, as well as the reliability threshold. We have improved the approach of [1] as follows. First, we use a different formula to estimate the energy consumption, which is supposed to be closer to actual execution scenarios. Secondly, in the mapping phase, we apply a layered WFD strategy, which is expected to be helpful for load-balancing, and for decreasing the overlap among copies of the same task. Finally, we implement several novel scheduling strategies, which introduce the idea of reordering chunks in between deadlines and of taking advantage of the utilization of the processor. Moreover, we have established that the sole problem of scheduling tasks with different WCETs, knowing the number of replicas, frequency and assigned processor for each task, is NP-complete in the strong sense.

We have evaluated the improvement of our strategies with a discrete event simulator (made publicly available). Extensive experiments conducted for various range of parameters have shown that: 1) Our new replica sets and WFD help finding a lower frequency, which gives good pre-condition for further energy savings; 2) our scheduling heuristics significantly outperform EDF_Paper. More specifically, EDF_Idle_CEQ and EDF_Idle_CEQ_Online have the best performance under a wide range of scenarios, with an average gain in energy of 20%.

Future work will aim at extending the algorithms to periodic workflows instead of independent task sets. The dependencies between nodes will dramatically complicate the problem. Another interesting direction is to deal with the same problem

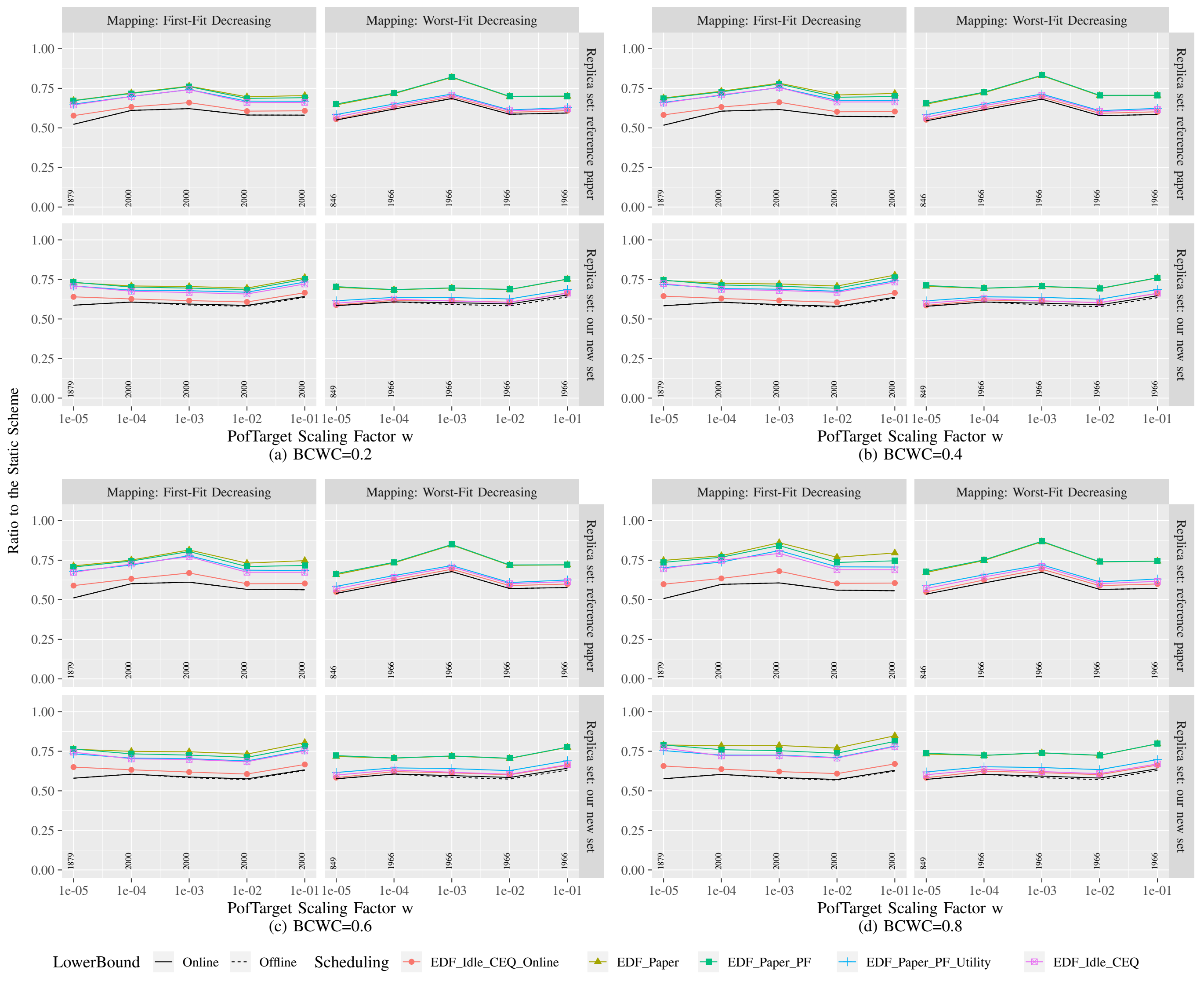

Figure 5: Impact of BC/WC and w with $U_{tot} = 2.5$.

with independent tasks, but targeting heterogeneous multicore systems.

VIII. Acknowledgments

This paper is partially supported by funding under National Key Research, Development Project 2017YFB1001800, NSFC 61972150, 61572195, and a JORISS grant from ENS de Lyon and ECNU.

References

[1] M. A. Haque, H. Aydin, and D. Zhu, "On reliability management of energy-aware real-time systems through task replication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 3, pp. 813–825, 2017.

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

[3] B. Zhao, H. Aydin, and D. Zhu, "Energy management under general task-level reliability constraints," in *18th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2012, pp. 285–294.

[4] D. Johnson, A. Demers, J. Ullman, M. Garey, and R. Graham, "Worst-case performance bounds for simple one-dimensional packing algorithms," *SIAM Journal on Computing*, vol. 3, no. 4, pp. 299–325, 1974.

[5] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[6] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, "Approximation algorithms for bin packing: A survey," in *Approximation Algorithms for NP-hard Problems*, D. S. Hochbaum, Ed. PWS Publishing Co., 1997, pp. 46–93.

[7] H. Aydin and Q. Yang, "Energy-aware partitioning for multiprocessor real-time systems," in *Proc. 17th Int. Symp. Parallel and Distributed Processing*, ser. IPDPS '03. IEEE Computer Society, 2003.

[8] D. C. Snowdon, G. Van Der Linden, S. M. Petters, and 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.

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

[10] A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*, ser. Algorithms and Combinatorics. Springer-Verlag, 2003, vol. 24.

[11] L. Han, L.-C. Canon, J. Liu, f. vivien, and Y. Robert, "Code to schedule for periodic real-time tasks under reliability constraints with minimal

energy consumption," Sep 2019. [Online]. Available: https://figshare.com/articles/Code_to_schedule_for_periodic_real-time_tasks_under_reliability_constraints_with_minimal_energy_consumption/9778319/1

[12] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.

[13] N. B. Rizvandi, A. Y. Zomaya, Y. C. Lee, A. J. Boloori, and J. Taheri, "Multiple frequency selection in dvfs-enabled processors to minimize energy consumption," *Energy-Efficient Distributed Computing Systems*, pp. 443–463, 2012.

[14] "Frequency behavior-intel," https://en.wikichip.org/wiki/intel/frequency_behavior.

[15] "Results with three real frequencies sets," https://figshare.com/s/7cd9483ae5e56421d0e7.

[16] A. Taherin, M. Salehi, and A. Ejlali, "Reliability-aware energy management in mixed-criticality systems," *IEEE Transactions on Sustainable Computing*, 2018.

[17] M. Ansari, S. Safari, A. Y. Khaksar, M. Salehi, and A. Ejlali, "Peak power management to meet thermal design power in fault-tolerant embedded systems," *IEEE Transactions on Parallel and Distributed Systems*, 2018.

[18] K. Cao, J. Zhou, P. Cong, L. Li, T. Wei, M. Chen, S. Hu, and X. S. Hu, "Affinity-driven modeling and scheduling for makespan optimization in heterogeneous multiprocessor systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[19] K. Cao, G. Xu, J. Zhou, T. Wei, M. Chen, and S. Hu, "Qos-adaptive approximate real-time computation for mobility-aware iot lifetime optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2018.

[20] J. Zhou, T. Wei, M. Chen, X. S. Hu, Y. Ma, G. Zhang, and J. Yan, "Variation-aware task allocation and scheduling for improving reliability of real-time mpsocs." in *DATE*, 2018, pp. 171–176.