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

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

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

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

RESEARCH REPORT
N° 9469
April 2022
Project-Team Roma

ISSN 0249-6399 ISRN INRIA/RR--9469--FR+ENG

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

Zhiwei Wu *†, Li Han *, Jing Liu *, Yves Robert †‡, Frédéric Vivien †

Project-Team Roma

Research Report n° 9469 — April 2022 — 23 pages

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 using 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.
This difficult tri-criteria mapping and scheduling problem (energy, deadline, reliability) has been studied only recently for workflows with arbitrary dependence constraints [20, 11]. We investigate new mapping and scheduling strategies based upon layers in the task graph, and which better balance replicas across processors, thereby decreasing the time overlap between the different replicas of the same task and saving energy. We compare these strategies with the two competitor approaches [20, 11] and a reference baseline [33] on a variety of benchmark workflows. Our best heuristics achieve an average energy gain of 40% over the competitors and of 80% over the baseline.

Key-words: real-time workflows, energy-aware, reliability, makespan, mapping, scheduling, tri-criteria optimization

* ECNU, Shanghaï, China
† Univ Lyon, EnsL, UCBL, CNRS, Inria, LIP, F-69342, LYON Cedex 07, France
‡ University of Tennessee Knoxville, USA

RESEARCH CENTRE GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Stratégies de placement et d'ordonnancement sensibles à la consommation énergétique pour graphes de tâches temps-réel avec contraintes de fiabilité

Résumé : Ce travail s'intéresse à la minimisation de la consommation énergétique lors du placement et de l'ordonnancement de graphes de tâches temps-réel soumis à des contraintes de fiabilité. Des instances d'un graphe de tâches sont soumises périodiquement à un système. Chaque instance est composée de plusieurs tâches et son exécution doit être terminée avant l'arrivée de l'instance suivante tout en respectant un niveau de fiabilité donné. Ce niveau de fiabilité est atteint en répliquant un certain nombre de fois chacune des tâches. La plateforme de calcul est constituée de processeurs identiques dont le voltage et la fréquence peuvent être modifiés (système *DVFS*). Chaque réplica de tâche peut se voir attribuer sa propre fréquence.

Ce problème tri-critère de placement et d'ordonnancement (énergie, dates butoir, fiabilité) n'a commencé à être étudié que très récemment avec des dépendances arbitraires [20, 11]. Nous étudions de nouvelles stratégies de placement et d'ordonnancement basées sur une notion de couche du graphe de tâches, et qui équilibrent mieux les réplicas entre les processeurs, ce qui permet de réduire le recouvrement temporel entre les différents réplicas d'une même tâche et, de ce fait, la consommation énergétique. Nous comparons ces stratégies à deux approches concurrentes [20, 11] et à une approche de référence [33], sur un tout un ensemble de graphes de tâches. Nos meilleures heuristiques obtiennent un gain d'énergie de 40% par rapport aux approches concurrentes et de 80% vis-à-vis de l'approche de référence.

Mots-clés : graphes de tâches temps-réel, consommation énergétique, faibilité, placement, ordonnancement, optimisation multi-critère

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 [4]. Informally, a workflow can be seen as the composition 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 [7, 10, 13] 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, so that scheduling and mapping decisions are taken based upon the *Worst case Execution Time (WCET)* of each task.

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 need to match the deadline, i.e., to guarantee a makespan that does not exceed the deadline bound. Because the problem is already NP-complete with two processors, independent tasks (no dependence) and no communication cost, 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 [32], 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 to take 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. This leads to achieving complicated trade-offs. Another important requirement is reliability: the execution of each task instance is prone to transient faults, so that several replicas of the same task instance must be executed in order to guarantee 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 [6]. 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 fault rate increases when scaling down frequency using DVFS [14], 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, it is possible to kill the other replicas of a task instance after one of them has succeeded, in order to limit this additional consumption, but several replicas may well have been executing concurrently until one of them succeeded, and the energy spent before their interruption 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 [20, 11], albeit with additional restrictions (see Section 4.1 for details). We present a novel approach that outperforms all previously published solutions on an extended set of simulations. 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 goal 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, and the design of several mapping and scheduling heuristics to solve it 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, here that can deal with arbitrary reliability thresholds;

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

3. An experimental evaluation based on a comprehensive set of simulations scenarios, showing that our best heuristics achieve a significant gain over the whole spectrum of application and platform parameters.

The rest of the paper is organized as follows. Section 2 provides a detailed description of the optimization problem under study. 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.

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.
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*) actually 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 [8]. 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 predecessors and successors of task T_i respectively. A task T_i can only start its execution when all of its predecessors have finished their execution and all intermediate data has been transferred.

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 $f_{max} = 1$. The interplay between task execution time and frequency is complicated [30]. To be closer to real cases, we let WCETs obey a randomized speedup function similar to Amdhal's law: specifically for task T_i at frequency f_j, the execution time 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 $[0.1, 0.3]$.

Fault model – We consider transient faults, modeled by an Exponential distribution with average arrival rate λ. The fault rate λ increases when the frequency is scaled down to save energy using DVFS [14]. 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 the execution of a task instance, 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 [19]. The *reliability* of a task instance is the probability of executing it successfully, in the presence of faults. Multiple replicas of tasks will be executed to mitigate the impact of transient faults. The reliability of a replica r_i^α of a task T_i running at frequency f_j is $R(r_i^\alpha, f_j) = \exp^{-\lambda(f_j) w_{i,f_j}}$. Note that two replicas of a given job will have the same execution time if run at the same frequency, because they operate on the same data.

Reliability threshold and deadline – There are two constraints for 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 set the reliability threshold of each task T_i as follows: $\mathcal{R}(T_i) = \sqrt[n]{\mathcal{R}(G)}$, where n is the number of tasks. Note that one could also give the set of $\mathcal{R}(T_i)$ for $i \in \{1 \dots n\}$ as inputs. Now, given the reliability threshold $\mathcal{R}(T_i)$ for each task T_i, the question is to determine the number of replicas k_{i,f_j} to use, and at which frequency f_j to execute them, so that $\mathcal{R}(T_i)$ is enforced, i.e., $\mathcal{R}(T_i) \geq R_i(f_j)$. A job fails only if all its replicas fail. Thus, we have $R_i(f_j) = 1 - (1 - R(r_i^\alpha, f_j))^{k_{i,f_j}}$.

Energy model – The energy cost of a replica 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 a single copy of job 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} copies is estimated as:

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

In Equation (1), we assume that there is no overlap between the primary replica (the first replica starting its execution) and the other replicas of a job, while secondary replicas fully overlap. Thus, the estimated energy consumption for a job 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.

Optimization Objective – The objective is to determine a set of replicas for each job and their execution frequencies, and to build a static schedule, where the replicas of each job 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 1; 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 job completes its execution successfully, all the other replicas of that job become redundant and are terminated instantaneously.

3 Scheduling heuristics

In this section, we describe the new heuristics that we introduce to solve the tri-criteria optimization problem. We start by outlining some design guidelines.

3.1 Task deadlines, reliability thresholds, 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 [32]). Consequently, if each task satisfies its deadline, the schedule is valid. We then have:

$$\mathcal{D}(T_i) = \begin{cases} \mathcal{D}(G) \text{ if } succ(T_i) = \phi \\ \min\limits_{T_j \in succ(T_i)} \mathcal{D}(T_j) - w(T_j) - c_{i,j} \text{ otherwise} \end{cases} \quad (2)$$

From the global reliability threshold $\mathcal{R}(G)$ we derive a reliability threshold $\mathcal{R}(T_i) = \sqrt[n]{\mathcal{R}(G)}$ for each task T_I, where n is the number of tasks in G. If each task satisfies its reliability threshold the global threshold is satisfied.

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^α	α-th replica of T_i
f_j	the j-th available frequency
$f(r_i^\alpha)$	frequency of replica r_i^α
$\mathcal{R}(G)$	reliability threshold for application G
$\mathcal{R}(T_i)$	reliability threshold for T_i
$R(r_i^\alpha, f_j)$	reliability of r_i^α 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^α
$FT(r_i^\alpha)$	completion time of r_i^α
$FT(T_i)$	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
$\mathcal{P}$	set of processors

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 a lower-bound on the remaining processing time for the workflow computed from the start of task T_i:

$$bl(T_i) = w_i + \max_{T_j \in succ(T_i)} c_{i,j} + bl(T_j) \tag{3}$$

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 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 from f_{max} to f_{min} for the first replica, 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 $\mathcal{R}(T_i) \leq 1 - (1 - R(r_i^1, f_{max}))^r < 1 - ((1 - R(r_i^1, f_{min})) \times (1 - R(r_i^1, f_{max}))^r)$.

3.3 Static Scheduling

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 job 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

$$P(r) = \arg\min_{p \in \mathcal{P}} EST(r, p) \tag{4}$$

and

$$ST(r) = EST(r, P(r)), \tag{5}$$

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

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

Here $ready(p)$ denotes the time at which processor p is available, according to the mapping and scheduling decisions previously taken. $repnum(T)$ 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 4: 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

$$FT(r_i^\alpha) = ST(r_i^\alpha) + w_{i, f(r_i^\alpha)} \tag{6}$$

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:

$$L(T_i) = \begin{cases} 1 + \max\limits_{T_j \in succ(T_i)} L(T_j) \text{ if } succ(T_i) \neq \phi, \\ 1 \text{ if } succ(T_i) = \phi \end{cases} \tag{7}$$

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 a same task.

Algorithm 1: The BasicLayer algorithm

```
   Input: A graph G = (V, E); the set F of possible frequencies; the reliability threshold
          R(G), the deadline D(G)
 1 foreach T_i ∈ V do
 2     Compute bl(T_i) using Equation 3
 3     Compute D(T_i) using Equation 2
 4     Compute L(T_i) using Equation 7
 5     R(T_i) ← ⁿ√R(G)
 6 Sort the tasks in a list TaskList by non increasing bottom-levels
 7 while TaskList ≠ ∅ do
 8     LayerIdentifier ← L(Head(TaskList))
 9     TasksInLayer ← {RemoveHead(TaskList)}
10     while L(Head(TaskList)) = LayerIdentifier do
11         TasksInLayer ← TasksInLayer ∪ {RemoveHead(TaskList)}
12     map the tasks in TasksInLayer using Algorithm 2
13     if not feasible then
14         map the tasks in TasksInLayer using Algorithm 3
15         if not feasible then
16             clear the whole schedule
17             reschedule application G, while only using Algorithm 3 to map each task
18             if not feasible then
19                 return failure to build a schedule
20             else
21                 return schedule
22 return schedule
```

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. BasicLayer first computes the bottom-level, deadline, reliability threshold and layer index of each task in lines 2 to 5. Then BasicLayer sorts tasks by their bottom-levels (line 6). In lines 8 to 11, BasicLayer builds a set of tasks that are in the same layer. In line 12, 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 14). 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 16), and the whole graph is rescheduled using only the task-by-task approach (Line 17). 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-

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

foreach $T_i \in TLayer$ **do**
  $rep\_num(T_i) \leftarrow k_{i,f_{max}}$
  Compute $ST(r_i^1)$ using Equation 5
$max\_rep\_num \leftarrow \max_{T_i \in TasksInLayer} rep\_num(T_i)$
foreach $k = 2$ *to* max_rep_num **do**
  foreach $T_i \in TasksInLayer$ **do**
    if $rep\_num(T_i) \leq k$ **then**
      Compute $ST(r_i^k)$ using Equation 5
foreach $T_i \in TasksInLayer$ **do**
  $FT(T_i) \leftarrow \max_{1 \leq k \leq rep\_num(T_i)} FT(r_i^k)$
  if $FT(T_i) > \mathcal{D}(T_i)$ **then**
    return *not feasible*
  else
    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

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

layer approach. We first compute the number of replicas for each task in *TasksInLayer* 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^α to represent any secondary replica (when $\alpha > 1$). After mapping all primary replicas in the layer, we map a first secondary replica for each task, then a second secondary replica, and so on (Lines 5 to 8). Finally, we check whether any task exceeds its deadline. 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. First, we compute the number of replicas with the same assumption as for Algorithm 2. Then we map each task primary replica and secondary replicas (lines 3 and 4).

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. 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 (1). 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

Algorithm 4: Schedule optimization

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

Sort the tasks in a list *TaskList* by non decreasing bottom-levels
has_changed $\leftarrow$ *true*
while *has_changed* **do**
  has_changed $\leftarrow$ *false*
  TasksInLayer $\leftarrow$ *ExtractFirstLayer*(*TaskList*)
  foreach $T_i \in$ *TasksInLayer* **do**
    foreach $2 \leq k \leq rep\_num(T_i)$ **do**
      $newFT(r_i^k) \leftarrow \min\left\{\mathcal{D}(T_i), \min_{T_j \in suc(T_i)} ST(T_j) - c_{i,j}\right\}$
    Minimize the overlap between replicas of T_i
    foreach $2 \leq k \leq rep\_num(T_i)$ **do**
      if $newFT(r_i^k) > FT(r_i^k)$ **then**
        $FT(r_i^k) \leftarrow newFT(r_i^k)$
        has_changed $\leftarrow$ *true*
        $ST(r_i^k) \leftarrow FT(r_i^k) - w_{i,f_{max}}$
  foreach $T_i \in$ *Executable* **do**
    $newFT(r_i^1) \leftarrow \min\left\{\mathcal{D}(T_i), \min_{2 \leq k \leq rep\_num(T_i)} ST(r_i^k)\right\}$
    if $FT(r_i^1) < newFT(r_i^1)$ **then**
      decrease r_i^1 frequency as much as possible while satisfying $FT(r_i^1) \leq newFT(r_i^1)$ and $k_{i,f(r_i^1)} \leq rep\_num(T_i)$
      postpone the start of r_i^1 such that $FT(r_i^1) = newFT(r_i^1)$
      has_changed $\leftarrow$ *true*
    if $rep\_num(T_i) > k_{i,f(r_i^1)}$ **then**
      delete one secondary replica of T_i

secondary replicas of the same job. In the BasicLayer algorithm, we map each replica as early as possible. Hence, the schedule may contain some slack with respect to deadlines. 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). Then, we build at Line 5 the layer containing the first task, exactly as we did in Lines 8–11 of Algorithm 1. We delay the start of each 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 job (Line 9), while forbidding any of them to start earlier than in the original schedule. Then we delay the secondary replica start time according to the new computed completion times (Lines 11–14). 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). 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 21). We repeat the whole process as long as we can improve the schedule.

3.7 Dynamic optimization

In all executions of our heuristics, the successful execution of a replica leads to the immediate cancellation of all other replicas of the same job. This is a crucial source of energy saving.

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 job 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 job.

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 [33] and two state-of-the-art approaches, OEA [20] and MILP [11].

Baseline QFEC Some related heuristics have been designed in the book [33]. Chapter 2 of [33] introduces heuristics for the bi-criteria problem (energy, makespan) while chapter 3 deals with the bi-criteria problem (reliability, makespan). Both chapters target heterogeneous platforms. In

this work, we have extended the heuristic QFEC of chapter 3 in [33] 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 [32]. This QFEC extension is used as a reference to report the energy savings of the other heuristics.

OEA The OEA heuristic [20] minimizes energy consumption whilst 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 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 [20], 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 [11] was published in last year's conference and presents a novel Mixed Integer Linear Programming (MILP) solver to deal with the same tri-criteria problem as in this paper, so it is the closest related work to our approach. However, MILP is limited by design to at most two replicas per task; hence, it cannot match high reliability thresholds that require three or more replicas. This intrinsic limitation comes from the fact that the system of equations is no longer linear when some tasks are allowed to have an arbitrary number of replicas: to the best of our knowledge, Equation (2) of [11] cannot be extended. Moreover, MILP does not take communication costs into account.

Comparison fairness We point out that all energy-aware heuristics discussed in this section (HEFT-OEA, ODS-OEA, MILP) were designed to solve problem instances with low reliability thresholds: there is a single replica per task in HEFT-OEA and ODS-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 HEFT-OEA and ODS-OEA to deal with replicas, and we have optimized their implementation in the same way as for our own heuristics. We have not extended MILP because we do not know how to add extra replicas and still derive a linear system of equations; therefore we only report results with MILP for low reliability scenarios in Section 4.4.

4.2 Experimental methodology

In the experiments, we have $M = 8$ cores and, following [19], we assume that the transient fault arrival rate at f_{max} is $\lambda_0 = 10^{-6}$ and the system sensitivity factor is $d = 4$. The static power and

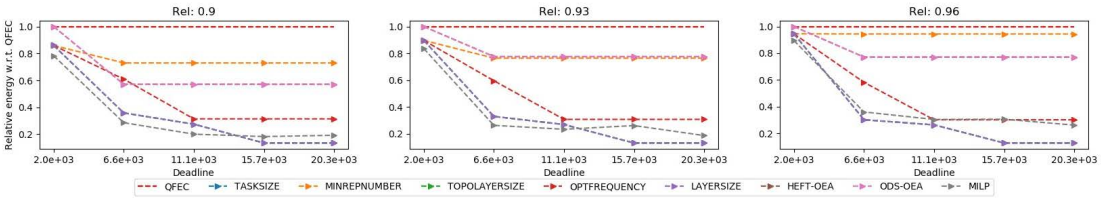

Figure 1: Assessing the performance of MILP on the BLAST workflow (with 10 tasks).

the frequency-independent part of the dynamic power are set to 5% and 15% respectively of the maximum frequency-dependent power consumption with $C = 1$. As detailed below, we validate our methods on a comprehensive set of parameters and assess their impact on performance.

Frequency Set. We have 4 sets of frequencies. The plots in Section 4.4 are for a 5-level frequency set which is taken from a real microprocessor [26] and whose (normalized) values are 1.0, 0.8, 0.6, 0.4, and 0.15. We also validate our methods on two other real frequency sets [15, 30] and one synthetic set ranging from 0.1 to 1 with steps of 0.1.

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) [12, 5, 22], as well as the 3 most classical decomposition algorithms of a $k \times k$ tiled matrix (LU, QR and Cholesky) [9]. 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 [16]. The actual execution time (at f_{max}) of each task instance T_i is randomly generated from a uniform distribution between $0.8 \times w_i$ and w_i.

Reliability Threshold. We first compute the reliability of the whole workflow

$$R(G) = \prod_{i=1}^{n} \mathcal{R}_i(f_{max})$$

when there is a single replica per task running at f_{max}. This will give us a corresponding probability of failure $F = 1 - R(G)$. Then we consider three reliability thresholds $1 - F$, $1 - \frac{F}{10}$ and $1 - \frac{F}{100}$. Thus, the reliability thresholds vary from one workflow to another.

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 F 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 F 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

$$c = \frac{S}{b}\frac{1}{T} \tag{8}$$

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 (8).

When not specified otherwise, we report performance for the case $c = 1$.

Deadline. We set five deadlines for each workflow: tightest, tight, medium, relatively loose, 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

As already stated, because it relies on a linear program, the MILP method can only find results for workflows with at most two replicas per task. Furthermore, the time complexity of the MILP method is quite large: the authors of MILP report in [11] 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 [11]). Therefore, in the simulations involving MILP, there are three differences from the general setting: (i) we only consider the 8 workflows from the Pegasus suite that could generate graphs with 10-30 tasks (for instance, the Montage workflow has a minimal task number of 133 and is not considered); (ii) we do not consider the communication costs because MILP do not take communication times into account; hence, we set the CCR value $c = 0$; and (iii) we consider three relatively low reliability thresholds 0.9, 0.93 and 0.96 for MILP to find a solution. In other words, we only consider problem instances where each task needs no more than two replicas in order to reach the global reliability threshold.

4.4 Results

On Figures 1 through 5, 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.

MILP experiments Figure 1 displays the relative performance of MILP and of our heuristics. Due to the limitations of MILP (cf. Section 4.3), we set the task size of the considered workflows to 10, 20, and 30 in our experiments. There are 10 tasks in the workflow used for Figure 1. This figure shows that when the deadline becomes larger (and thus looser), our methods (e.g., TaskSize, LayerSize) perform better than MILP. Indeed, with larger deadlines, there is more scheduling freedom and it becomes possible to decrease the overlap between the replicas of a same job. Such a strategy leads to significant reduction of the energy consumption and explains why our heuristics outperform MILP. One can also observe, that when the reliability increases our heuristics outperform MILP for tighter deadlines.

When the workflows include 20 tasks and when the reliability is set at 0.96, MILP fails to find a solution for the SRASearch workflow. When the task size grows to 30 (with a reliability of 0.96), MILP also fails to find solutions for the Epigenomics and BWA workflows. This is because when the size of workflows increases, for the reliability threshold to be satisfied some tasks need to have more than two replicas, which MILP cannot provide. On the contrary, our methods do not suffer from this limitation. Overall, MILP achieves the best performance only when the deadline is very tight or the reliability threshold is very low.

General experiments Figure 2 reports performance when the reliability is low enough that each task only requires a single replica. Heft-OEA and ODS-OEA, which were designed for

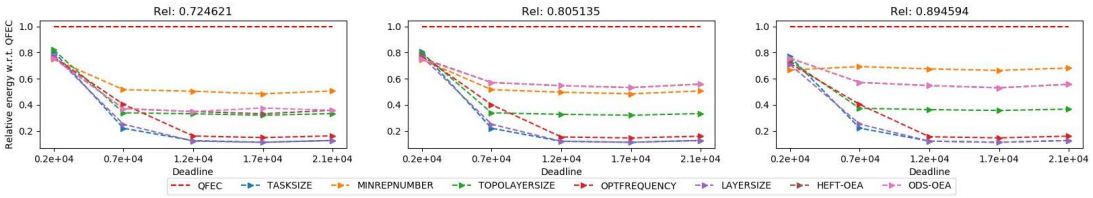

Figure 2: Assessing performance with the Montage workflow when the reliability threshold is very low.

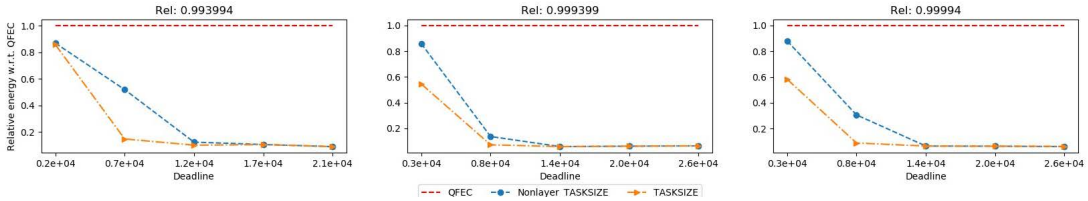

Figure 3: Assessing the impact of the layer-by-layer approach on the Montage workflow.

this peculiar case, are nonetheless outperformed by our heuristics, and especially by LayerSize and TaskSize.

We assess with Figure 3 the impact of the layer-by-layer approach. 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.

We assess with Figure 4 the impact of dynamic optimizations. Once again, when the deadline is tight this type of optimization can lead to additional gains. Similarly to the layer-by-layer approach, the dynamic optimizations never lead to worse performance for all types of workflows under all settings and should therefore always be used.

Figure 5 compares the performance of the different heuristics in the general case. When the deadline increases, MinRepNumber and TopoLayerSize do not succeed to reach the same "asymptotic" performance as our best strategies. This is because with MinRepNumber and TopoLayerSize, some tasks are not allowed to have an additional replica, which forbids the primary replicas for those tasks to run at frequency f_{min} although the deadline is loose enough to allow it. Same to the low reliability case, the performances of Heft-OEA and of ODS-OEA are not competitive. Among the three remaining heuristics, LayerSize and TaskSize achieve

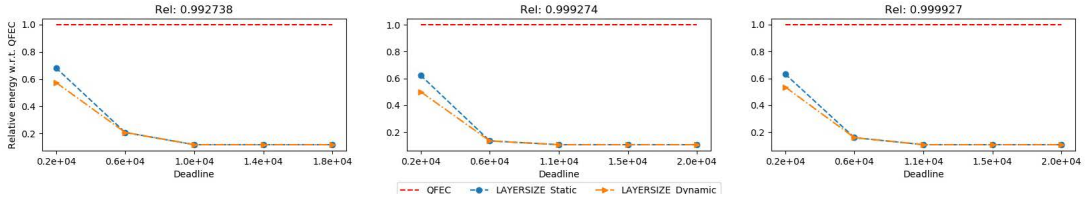

Figure 4: Assessing the impact of the dynamic optimization on the Cycles workflow.

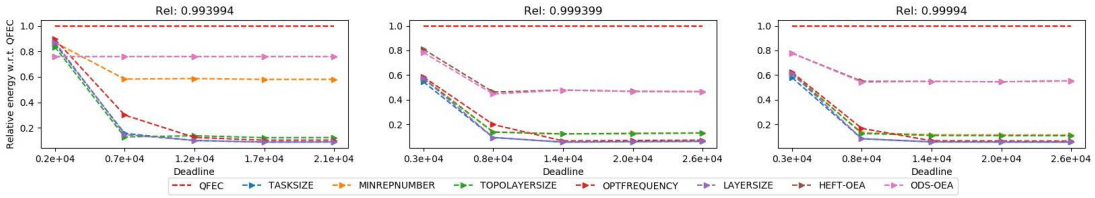

Figure 5: Assessing performance with the Montage workflow.

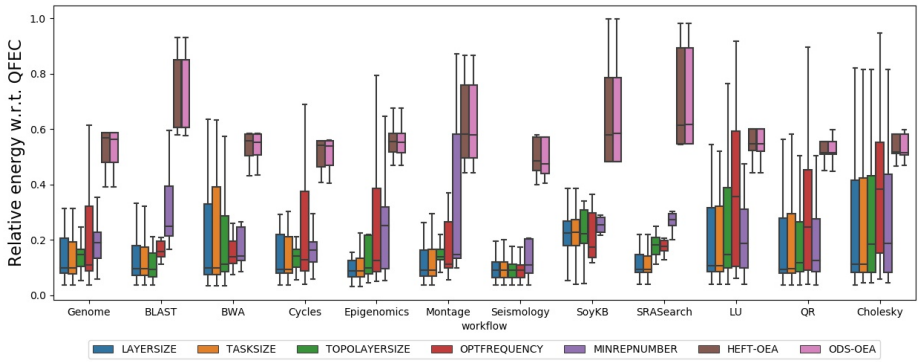

Figure 6: Average performance of the different heuristics for each workflow over all the parameter settings.

similar performance overall, while that of OptFrequency is slightly worse. These conclusions, drawn from a single workflow and a single frequency set, are representative of our whole set of simulations. Of course, due to the diversity of the workflows and of the parameter settings, there are a few instances where for instance, OptFrequency achieves good performance (with workflow SoyKB), or where LayerSize and TaskSize have slightly sub-optimal performance (usually when the deadline is the tightest).

Synthesis Figure 6 presents the average performance achieved by each heuristic, over the whole set of experiments (for each workflow, for the different frequency sets, reliability thresholds, deadlines, and communication-to-computation ratio). It is obvious that Heft-OEA and ODS-OEA achieve rather poor performance, even though they already lead to significant gains with respect to QFEC. OptFrequency, MinRepNumber, and TopoLayerSize, while achieving better performance than ODS-OEA and Heft-OEA (and thus than QFEC), are nevertheless outperformed by the two best heuristics (except OptFrequency for SoyKB). In most cases LayerSize and TaskSize achieve the best average performance; even if for some workflows their average is good, their worst case may be worse than that of some other heuristics. On average, LayerSize and TaskSize have an energy gain of 40% over the competitors and save 80% of the energy consumed by QFEC. When the deadline is loose, they can save up to 95% of QFEC cost.

5 Related work

A lot of work focus on this tri-criteria optimization problem (reliability, deadlines, energy), but for independent task sets. [19] proposes both static and dynamic methods to reduce overlapping between replicas, thus minimizing energy consumption. [18] improves the approach of [19] through three steps, and proposes several new scheduling heuristics, which have the best performance under a wide range of scenarios. [27] considers an energy-budget-aware reliability management (enBudRM) method for multi-core embedded systems featuring hybrid energy source, while [24] proposes heuristic-based scheduling techniques to minimize the energy consumption of task executions when faults are absent, and preserves feasibility under the worst case of fault occurrences.

There are other work that consider managing energy using DVFS for real-time workflows. [38] considers energy-aware duplication scheduling algorithms for a parallel application on homogeneous systems. [23] presents energy-conscious scheduling to implement joint reduction between the schedule length and energy consumption of a parallel application on heterogeneous systems. [21] 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. [31] also examines the same problem as that reported in [21] by switching off inefficient processors to reduce static energy consumption based on slack time reclamation. The MaxRe [36] and RR [35] 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 work 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. [17] proposes reliability-aware power management schemes to save energy while guaranteeing a certain level of system reliability on homogeneous multiprocessors. [29] introduces a power-efficient reliability management method through dynamic redundancy and voltage scaling under variations on homogeneous manycore processors. [28] 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 [2] but again with major differences: mapping of the tasks onto the processors is assumed to be given, as in [20], and reliability is achieved through re-execution, not replication. Similarly, [25] 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. [37] 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. [17] studied energy minimization problem for real-time workflows on homogeneous platforms, while [34] worked on the same problem on heterogeneous platforms. Both work save energy by scaling down frequencies of selected tasks, and preserve reliability by shared recovery. But they only can guarantee the original reliability level which means all tasks running at f_{max} with no replica. Finally, [1] 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 apply to workflows of arbitrary shape are OEA [20] and MILP [11], 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 work 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 [3] 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 40% over the Heft-OEA and ODS-OEA competitors and save 80% of the energy consumed by QFEC. We also report partial comparisons with MILP, which is limited to 30 tasks per workflow with at most another replica each, and 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.

References

[1] Ismail Assayad, Alain Girault, and Hamoudi Kalla. Tradeoff exploration between reliability, power consumption, and execution time for embedded systems. *International Journal on Software Tools for Technology Transfer*, 15(3):229–245, 2013.

[2] Guillaume Aupy, Anne Benoit, and Yves Robert. Energy-aware scheduling under reliability and makespan constraints. In *International Conference on High Performance Computing (HiPC'2012)*. IEEE Computer Society Press, 2012.

[3] Olivier Beaumont, Vincent Boudet, and Yves 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.

[4] Shishir Bharathi, Ann Chervenak, Ewa Deelman, Gaurang Mehta, Mei-Hui Su, and Karan Vahi. Characterization of scientific workflows. In *SC'08 Workshop: The 3rd Workshop on Workflows in Support of Large-scale Science (WORKS08) web site*, Austin, TX, November 2008. ACM/IEEE.

[5] Shishir Bharathi, Ann Chervenak, Ewa Deelman, Gaurang Mehta, Mei-Hui Su, and Karan Vahi. Characterization of scientific workflows. In *Workflows in Support of Large-Scale Science (WORKS)*, pages 1–10. IEEE, 2008.

[6] Alan Burns and Andy Wellings. *Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX*. Addison-Wesley, 4th edition, 2009.

[7] Eddy Caron, Frédéric Desprez, Tristan Glatard, Maheshwari Ketan, Johan Montagnat, and Damien Reimert. Workflow-based comparison of two distributed computing infrastructures. In *Workflows in Support of Large-Scale Science (WORKS10)*, New Orleans, November 14 2010. In Conjunction with Supercomputing 10 (SC'10), IEEE. hal-00677820.

[8] Huangke Chen, Xiaomin Zhu, Dishan Qiu, and Ling Liu. Uncertainty-aware real-time workflow scheduling in the cloud. In *IEEE 9th Int. Conf. Cloud Computing (CLOUD)*, pages 577–584, 2016.

[9] Jaeyoung Choi, Jack J Dongarra, L Susan Ostrouchov, Antoine P Petitet, David W Walker, and R Clint Whaley. Design and implementation of the scalapack lu, qr, and cholesky factorization routines. *Scientific Programming*, 5(3):173–184, 1996.

[10] Peter Couvares, Tevik Kosar, Alain Roy, Jeff Weber, and Kent Wenger. Workflow Management in Condor. In Ian Taylor, Ewa Deelman, Dennis Gannon, and Matthew Shields, editors, *Workflows for e-Science*, pages 357–375. Springer, 2007.

[11] Minyu Cui, Angeliki Kritikakou, Lei Mo, and Emmanuel 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)*, pages 387–399, 2021.

[12] Rafael Ferreira da Silva, Weiwei Chen, Gideon Juve, Karan Vahi, and Ewa Deelman. Community resources for enabling research in distributed scientific workflows. In *e-Science (e-Science), 2014 IEEE 10th International Conference on*, volume 1, pages 177–184. IEEE, 2014.

[13] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John Good, Anastasia Laity, Joseph Jacob, and Daniel Katz. Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems. *Scientific Programming Journal*, 13(3):219–237, 2005.

[14] Anand Dixit and Alan Wood. The impact of new technology on soft error rates. In *2011 International Reliability Physics Symposium*, pages 5B–4. IEEE, 2011.

[15] Frequency behavior-intel. `https://en.wikichip.org/wiki/intel/frequency_behavior`.

[16] Gene H. Golub and Charles F. Van Loan. *Matrix Computations, Fourth Edition*. Johns Hopkins University Press, 4th edition, 2013.

[17] Yifeng Guo, Dakai Zhu, and 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, page 1–8, USA, 2011. IEEE Computer Society.

[18] Li Han, Louis-Claude Canon, Jing Liu, Yves Robert, and Frédéric 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.

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

[20] Jing Huang, Renfa Li, Xun Jiao, Yu Jiang, and Wanli Chang. Dynamic DAG Scheduling on Multiprocessor Systems: Reliability, Energy, and Makespan. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):3336–3347, 2020.

[21] Qingjia Huang, Sen Su, Jian Li, Peng Xu, Kai Shuang, and Xiao 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*, pages 781–786, 2012.

[22] Gideon Juve, Ann Chervenak, Ewa Deelman, Shishir Bharathi, Gaurang Mehta, and Karan Vahi. Characterizing and profiling scientific workflows. *Future Generation Computer Systems*, 29(3):682–692, 2013.

[23] Young Choon Lee and Albert Y. Zomaya. Energy conscious scheduling for distributed computing systems under different operating conditions. *IEEE Trans. Parallel Distributed Syst.*, 22(8):1374–1381, 2011.

[24] Yu Liu, Han Liang, and Kaijie 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*, pages 1444–1449, 2010.

[25] Paul Pop, Kåre Harbo Poulsen, Viacheslav Izosimov, and Petru 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)*, pages 233–238, 2007.

[26] Nikzad Babaii Rizvandi, Albert Y Zomaya, Young Choon Lee, Ali Javadzadeh Boloori, and Javid Taheri. Multiple frequency selection in dvfs-enabled processors to minimize energy consumption. *Energy-Efficient Distributed Computing Systems*, pages 443–463, 2012.

[27] Sepideh Safari, Mohsen Ansari, Mohammad Salehi, and Alireza Ejlali. Energy-budget-aware reliability management in multi-core embedded systems with hybrid energy source. *CSI Journal on Computer Science and Engineering*, 15:31–43, 07 2018.

[28] Mohammad Salehi, Alireza Ejlali, and Bashir M. Al-Hashimi. Two-phase low-energy n-modular redundancy for hard real-time multi-core systems. *IEEE Trans. Parallel Distributed Syst.*, 27(5):1497–1510, 2016.

[29] Mohammad Salehi, Mohammad Khavari Tavana, Semeen Rehman, Florian Kriebel, Muhammad Shafique, Alireza Ejlali, and Jörg 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*, pages 225–230, 2015.

[30] David C Snowdon, Godfrey Van Der Linden, Stefan M Petters, and Gernot Heiser. Accurate run-time prediction of performance degradation under frequency scaling. In *Workshop on Operating Systems Platforms for Embedded Real-Time applications*, page 58, 2007.

[31] Zhuo Tang, Ling Qi, Zhenzhen Cheng, Kenli Li, Samee Ullah Khan, and Keqin Li. An energy-efficient task scheduling algorithm in dvfs-enabled cloud environment. *J. Grid Comput.*, 14(1):55–74, 2016.

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

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

[34] Longxin Zhang, Kenli Li, Keqin Li, and Yuming Xu. Joint optimization of energy efficiency and system reliability for precedence constrained tasks in heterogeneous systems. *International Journal of Electrical Power and Energy Systems*, 78:499–512, 2016.

[35] Laiping Zhao, Yizhi Ren, and Kouichi Sakurai. Reliable workflow scheduling with less resource redundancy. *Parallel Comput.*, 39(10):567–585, 2013.

[36] Laiping Zhao, Yizhi Ren, Yang Xiang, and Kouichi 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*, pages 434–441, 2010.

[37] Dakai Zhu and Hakan Aydin. Energy management for real-time embedded systems with reliability requirements. In *Proceedings of the IEEE/ACM International Conference on Computer-aided Design (ICCAD)*, pages 528–534, 2006.

[38] Ziliang Zong, Adam Manzanares, Xiaojun Ruan, and Xiao Qin. EAD and PEBD: two energy-aware duplication scheduling algorithms for parallel tasks on homogeneous clusters. *IEEE Trans. Computers*, 60(3):360–374, 2011.

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399