Approximation Algorithms for Energy, Reliability, and Makespan Optimization Problems

Guillaume Aupy, Anne Benoit

Parallel Processing Letters

APPROXIMATION ALGORITHMS FOR ENERGY, RELIABILITY, AND MAKESPAN OPTIMIZATION PROBLEMS

GUILLAUME AUPY and ANNE BENOIT
LIP - ENS Lyon
46 Allée d'Italie
69364 Lyon Cedex 07, France

Received July 2014
Revised March 2015
Communicated by R. Klasing

ABSTRACT

We consider the problem of scheduling an application on a parallel computational platform. The application is a particular task graph, either a linear chain of tasks, or a set of independent tasks. The platform is made of identical processors, whose speed can be dynamically modified. It is also subject to failures: if a processor is slowed down to decrease the energy consumption, it has a higher chance to fail. Therefore, the scheduling problem requires us to re-execute or replicate tasks (i.e., execute twice the same task, either on the same processor, or on two distinct processors), in order to increase the reliability. It is a tri-criteria problem: the goal is to minimize the energy consumption, while enforcing a bound on the total execution time (the makespan), and a constraint on the reliability of each task. Our main contribution is to propose approximation algorithms for linear chains of tasks and independent tasks. For linear chains, we design a fully polynomial-time approximation scheme. However, we show that there exists no constant factor approximation algorithm for independent tasks, unless P=NP, and we propose in this case an approximation algorithm with a relaxation on the makespan constraint.

Keywords: Scheduling; energy; reliability; makespan; models; approximation algorithms.

1. Introduction

Energy-awareness is now recognized as a first-class constraint in the design of new scheduling algorithms. To help reduce energy dissipation, current processors from AMD, Intel and Transmeta allow the speed to be set dynamically, using a dynamic voltage and frequency scaling technique (DVFS). Indeed, a processor running at speed f dissipates f^3 watts per unit of time [10]. However, it has been recognized that reducing the speed of a processor has a negative effect on the reliability of a schedule: if a processor is slowed down, it has a higher chance to be subject to transient failures, caused for instance by software errors [32, 17]. In order to make up for the loss in reliability due to the lower speeds used for energy efficiency, we consider two standard techniques: *re-execution* consists in re-executing a task twice on the same processor [32, 31], while *replication* consists in executing the same task on two distinct processors simultaneously [5].

Motivated by the application of speed scaling on large scale machines [24], we consider a tri-criteria problem energy/reliability/makespan: the goal is to minimize the energy consumption, while enforcing a bound on the makespan, i.e., the total execution time, and a constraint on the reliability of each task. The application is a particular task graph, either a linear chain of tasks, or a set of independent tasks. The platform is made of identical processors, whose speed can be dynamically modified within a range $[f_{\min}, f_{\max}]$.

The schedule therefore requires us to (i) decide which tasks are re-executed or replicated; (ii) decide on which processor(s) each task is executed; (iii) decide at which speed each processor is processing each task. For a given schedule, we can compute the total execution time, also called *makespan*, and it should not exceed a prescribed deadline. Each task has a reliability that can be computed given its execution speed and its eventual replication or re-execution, and we must enforce that the execution of each task is reliable enough. Finally, we aim at minimizing the energy consumption. Note that we consider a set of homogeneous processors, but each processor may run at a different speed; this corresponds to typical current platforms with DVFS.

In this paper, we investigate the tri-criteria problem of minimizing the energy consumption with a bound on the makespan and a constraint on the reliability. Related work is discussed in Section 2. In Section 3, we formally introduce this tri-criteria scheduling problem, based on the previous models proposed in [31] and [8]. To the best of our knowledge, this is the first model including both re-execution and replication in order to deal with failures. The main contribution of this paper is then to provide approximation algorithms for some particular instances of this tri-criteria problem. For linear chains of tasks, we propose a fully polynomial-time approximation scheme (Section 4). Then in Section 5, we show that there exists no constant factor approximation algorithm for the tri-criteria problem with independent tasks, unless P=NP. We prove that by relaxing the constraint on the makespan, we can give a polynomial-time constant factor approximation algorithm. To the best of our knowledge, these are the first approximation algorithms for the tri-criteria problem.

2. Related work

In this section, we first discuss related work on how to handle failures, and then on problems aiming to minimize the energy consumption. Finally, we discuss papers targeting the tri-criteria problem.

Reliability: checkpointing vs replication. Failures are usually handled by adding redundancy, either continuously (replication) [31, 21, 23], or at periodic intervals (migration from faulty node to spare node, rollback and recovery) [22, 18]. In the latter case, the state of an application must be preserved (checkpointing), and the system must roll back to the last saved checkpoint. However, the amount of replication and/or the frequency of checkpointing must be optimized carefully. For example, systematic replication degrades performance, but the application is at larger risk if no replication is used.

We focus in this paper on the replication technique, which is natural for applications that consist in a graph of tasks, where the unit of replication is well defined (a whole task

is replicated) [21, 5]. Checkpointing for such applications is also under investigation [14], but it is out of scope of this paper.

Energy consumption minimization. The problem of minimizing the energy consumption without exceeding a given deadline, using DVFS, has been widely studied, without accounting for reliability issues. We refer to the survey by Albers [1] on the different energy-efficient algorithms. The general problem was introduced by Yao et al. [29], where the authors consider a power function of the speed f, $P(f) = f^\alpha$ with $\alpha > 1$ on a single processor. The energy consumption between t_1 and t_2 is then $E = \int_{t_1}^{t_2} P(f)df$. Recently, Rauber et al. [27] investigated and evaluated measurement methods, and concluded that classical energy models are not too far from the observed energy consumption; they make the classical assumption $\alpha = 3$.

Under this model, minimizing the energy consumption for a *linear chain of tasks* on any number of processors, without exceeding a given deadline, is known to be solvable in polynomial time [7].

Benoit et al. [15] studied the performance of greedy algorithms for the scheduling of *independent tasks* on p processors, and proposed some approximation results. Furthermore, they showed how the PTAS proposed by Alon et al. [3] can be adapted to solve the energy minimization problem with a constraint on the makespan. Several other variants of the problem have been studied, for instance, a constant factor approximation algorithm is proposed to solve the problem on a single processor where jobs have individual release times, deadlines and processing times, and where no preemption is allowed [4]. Closer to our work, Albers et al. [2] studied this problem with different release times and deadlines on multiple processors, considering an ideal model with no constraint on the set of possible speeds, and also proposed some approximation algorithms.

The bi-criteria (makespan, energy) problem for a *graph with precedence constraints* has also been studied extensively in the past years. Pruhs et al. [26] considered the problem of minimizing the makespan of the schedule without exceeding a given energy budget. Recently, Bampis et al. [12] improved their results. Aupy et al. [7] considered the complementary problem of minimizing the energy consumption under a makespan constraint for different speed models.

Tri-criteria problem. The papers cited above for the energy minimization problem do not account for reliability issues. However, Zhu et al. [32] showed that reducing the speed of a processor increases the number of transient failure rates of the system; the probability of failures increases exponentially, and this probability cannot be neglected in large-scale computing [24]. Few authors have tackled the tri-criteria problem including reliability, and to the best of our knowledge, there are no approximation algorithms for this problem. Zhu and Aydin [31] initiated the study of this problem, using re-execution. However, they restricted their study to the scheduling problem on a single processor, and did not provide any approximation ratio on their algorithm. More recently, Haque et al. [21] proposed an energy-efficient replication algorithm (EER) for periodic real-time applications, and evaluated its performance through simulation.

Assayad et al. [5] have proposed an off-line tri-criteria scheduling heuristic called TSH, which uses 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 replicate onto the next free processors. However, the complexity of this heuristic is unfortunately exponential in the number of processors, and the authors did not give an approximation ratio on their heuristic.

Finally, Aupy et al. [8] also studied the tri-criteria problem, but from a heuristic point of view, without ensuring any approximation ratio on their heuristics. Moreover, they did not consider replication of tasks, but only re-execution as in [31]. However, they presented a formal model of the tri-criteria problem, re-used in this paper.

3. Framework

Consider an application task graph $\mathcal{G} = (V, \mathcal{E})$, where $V = \{T_1, T_2, \ldots, T_n\}$ is the set of tasks, $n = |V|$, and where $\mathcal{E}$ is the set of precedence edges between tasks. If $(T_i, T_j) \in \mathcal{E}$, then there is a precedence constraint between tasks T_i and T_j, also denoted $T_i \rightarrow T_j$. It means that task T_j cannot start its execution before task T_i has been successfully executed. For $1 \leq i \leq n$, task T_i has a weight w_i, that corresponds to the computation requirement of the task. $S = \sum_{i=1}^{n} w_i$ is the sum of the computation requirements of all tasks.

The goal is to map the task graph onto p identical processors that can have arbitrary speeds, determined by their frequency, which can take any value in the interval $[f_{\min}, f_{\max}]$ (dynamic voltage and frequency scaling with continuous speeds). Higher frequencies, and hence faster speeds, allow for a faster execution, but they also lead to a much higher (supra-linear) energy consumption. Note that Aupy et al. [8] showed that it is always better to execute a task at a single speed, and therefore we assume in the following that each execution of a task is done at a single speed.

We now detail the three objective criteria (makespan, reliability, energy), and then formally define the optimization problem in Section 3.4.

3.1. *Makespan*

The makespan of a schedule is its total execution time. The first task is scheduled at time 0, so that the makespan of a schedule is simply the maximum time at which one of the processors finishes its computations. Given a schedule, the makespan should not exceed the prescribed deadline D.

Let $\mathcal{E}xe(w_i, f)$ be the execution time of a task T_i of weight w_i at speed f. We enforce the classical linear cost model for execution times [22]: $\mathcal{E}xe(w_i, f) = \frac{w_i}{f}$. Note that we consider a worst-case scenario, and the deadline D must be matched even in the case where all tasks that are scheduled to be executed several times fail during their first executions, hence all execution and re-execution times should be accounted for.

3.2. *Reliability*

To define the reliability, we use the failure model of [32], [31] and [28]. We do not consider fail-stop failures that correspond to hardware failures and interrupt definitively the failed processor (until repair), but rather *transient* failures, which are caused by software errors for example. Such failures invalidate only the execution of the current task; the processor subject to that failure will be able to recover and execute the subsequent tasks assigned to it (if any), for instance a re-execution of the failed task.

We use the reliability model that states that the radiation-induced transient failures follow a Poisson distribution [32]. The parameter λ of the Poisson distribution is then $\lambda(f) = \tilde{\lambda_0}\ e^{\tilde{d}\frac{f_{\max}-f}{f_{\max}-f_{\min}}}$, where $f_{\min} \leq f \leq f_{\max}$ is the processing speed, the exponent $\tilde{d} \geq 0$ is a constant, indicating the sensitivity of failure rates to dynamic voltage and frequency scaling, and $\tilde{\lambda_0}$ is the average failure rate at speed $f_{\max}$. We see that reducing the speed for energy saving increases the failure rate exponentially. The reliability of a task T_i executed once at speed f is the probability of a successful execution, and it is expressed as

$$R_i(f) = e^{-\lambda(f)\times\mathcal{E}xe(w_i,f)}.$$

Because the failure rate $\tilde{\lambda_0}$ is usually very small, of the order of 10^{-5} per time unit [5], or even 10^{-6} [11, 25], we can use the first order approximation of $R_i(f)$ as

$$R_i(f) = 1 - \lambda(f) \times \mathcal{E}xe(w_i, f) = 1 - \tilde{\lambda_0}\ e^{\tilde{d}\frac{f_{\max}-f}{f_{\max}-f_{\min}}} \times \frac{w_i}{f} = 1 - \lambda_0\ e^{-df} \times \frac{w_i}{f},$$

where $d = \frac{\tilde{d}}{f_{\max}-f_{\min}}$ and $\lambda_0 = \tilde{\lambda_0}e^{df_{\max}}$.

Note that this equation holds if $\lambda(f) \times \frac{w_i}{f} \ll 1$. With, say, $\lambda(f) = 10^{-5}$, we need $\frac{w_i}{f} \leq 10^3$ to get an accurate approximation with $\lambda(f)\times\frac{w_i}{f} \leq 0.01$: the task should execute within 16 minutes (960 seconds). In other words, large (computationally demanding) tasks require reasonably high processing speeds with this model (which makes full sense in practice).

We consider that a task is reliable enough when it is executed once at a speed greater than or equal to a threshold speed $f_{\texttt{rel}} = \gamma f_{\max}$, where $\frac{f_{\min}}{f_{\max}} \leq \gamma \leq 1$ is fixed by the user and corresponds to the reliability of the system. For highly critical systems, $\gamma = 1$ and therefore $f_{\texttt{rel}} = f_{\max}$ [30]. The value of $f_{\texttt{rel}}$ can also be fixed to ensure a bound R_0 on the global reliability of the system, which is expressed as $\Pi_{i=1}^{n} R_i(f_{\texttt{rel}})$ (it is the probability that all tasks are executed successfully):

$$\Pi_{i=1}^{n} R_i(f_{\texttt{rel}}) = e^{-\frac{\lambda(f_{\texttt{rel}})}{f_{\texttt{rel}}}\sum_{i=1}^{n} w_i} \geq R_0,$$

and therefore we obtain

$$f_{\texttt{rel}} \geq \frac{W\Big(\frac{\lambda_0 d \sum_{i=1}^{n} w_i}{-\ln R_0}\Big)}{d},$$

where W is the product logarithmic (Lambert) function.

In order to limit energy consumption, the execution speed of a task can be further decreased, but then the probability of having at least one transient failure during the execution of this task increases drastically, both because of the extended execution time and the increased failure rate $\lambda(f)$. In this case, we therefore enforce the execution of a *backup task*

[31, 30]. We do not execute automatically this task at the maximum speed (or speed $f_{\texttt{rel}}$) as was done in previous work, but rather we choose a re-execution speed such that the reliability of both executions is at least equal to the reliability of a single execution at speed $f_{\texttt{rel}}$. Therefore, either task T_i is executed only once at speed $f \geq f_{\texttt{rel}}$, or it is executed twice (speeds $f^{(1)}$ and $f^{(2)}$), and the reliability, i.e., the probability that at least one of the attempts do not fail: $R_i = 1 - (1 - R_i(f^{(1)}))(1 - R_i(f^{(2)}))$ should be at least equal to $R_i(f_{\texttt{rel}})$. Note that if $T_i \to T_j$, then the backup task of T_i must be scheduled to finish its execution before any execution of T_j, so that the precedence constraints are respected even in the event of a failure.

We restrict to one single backup task, which can be scheduled either on the same processor as the original task (what we call *re-execution*), or on another processor (what we call *replication*). As motivated earlier, a single backup task executed at speed $f_{\texttt{rel}}$ would ensure that the execution is reliable enough, hence there would be no need of further backup tasks [31, 30]. Intuitively, having two or more backup tasks may lead to further energy savings in a few particular cases, but at a price of a highly increased execution time (and a much more complex study).

Note that if both execution speeds are equal, i.e., $f^{(1)} = f^{(2)} = f$, then the reliability constraint becomes $1 - (\lambda_0 w_i \frac{e^{-df}}{f})^2 \geq R_i(f_{\texttt{rel}})$, and therefore

$$\lambda_0 w_i \frac{e^{-2df}}{f^2} \leq \frac{e^{-df_{\texttt{rel}}}}{f_{\texttt{rel}}} \ .$$

In the following, $f_{\text{inf},i}$ is the maximum between $f_{\min}$ and the solution to the equation $\lambda_0 w_i \frac{e^{-2df_{\text{inf},i}}}{(f_{\text{inf},i})^2} = \frac{e^{-df_{\texttt{rel}}}}{f_{\texttt{rel}}}$, and hence if task T_i is executed twice at a speed greater than or equal to $f_{\text{inf},i}$, then the reliability constraint is met.

3.3. *Energy*

The total energy consumption corresponds to the sum of the energy consumption of each task. Let E_i be the energy consumed by task T_i. For one execution of T_i at speed f, the corresponding energy consumption is $E_i(f) = \mathcal{E}xe(w_i, f) \times f^3 = w_i \times f^2$, which corresponds to the dynamic part of the classical energy models of the literature [10, 13]. Note that we do not take static energy into account, because all processors are up and alive during the whole execution.

If task T_i is executed only once at speed f, then $E_i = E_i(f)$. Otherwise, if task T_i is executed twice at speeds $f^{(1)}$ and $f^{(2)}$, it is natural to add up the energy consumed during both executions, just as we consider both execution times when enforcing the deadline on the makespan. Again, this corresponds to the worst-case execution scenario. We obtain $E_i = E_i(f^{(1)}) + E_i(f^{(2)})$. Note that some authors [31] consider only the energy spent for the first execution in the case of re-execution, which seems unfair: re-execution comes at a price both in the makespan and in the energy consumption. Finally, the total energy consumed by the schedule, which we aim at minimizing, is $E = \sum_{i=1}^{n} E_i$.

3.4. *Optimization problem*

Given an application graph $\mathcal{G} = (V, \mathcal{E})$ and p identical processors, TRI-CRIT is the problem of finding a schedule that specifies which tasks should be executed twice, on which processor and at which speed each execution of a task should be processed, such that the total energy consumption E is minimized, subject to the deadline D on the makespan and to the local reliability constraints $R_i \geq R_i(f_{\mathtt{rel}})$ for each task $T_i \in V$.

Note that TRI-CRIT may have no solution: it may well be the case that the deadline cannot be enforced even if all tasks are executed only once at speed $f_{\max}$.

We focus in this paper on the two following sub-problems that are restrictions of TRI-CRIT to special application graphs:

- TRI-CRIT-CHAIN: the graph is such that $\mathcal{E} = \cup_{i=1}^{n-1}\{(T_i, T_{i+1})\}$, i.e., $T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_n$ (linear chain of tasks).
- TRI-CRIT-INDEP: the graph is such that $\mathcal{E} = \emptyset$ (independent tasks).

4. Linear chains

In this section, we focus on the TRI-CRIT-CHAIN problem, that was shown to be NP-hard even on a single processor [8]. We derive an FPTAS (Fully Polynomial-Time Approximation Scheme) to solve the general problem with replication and re-execution on p processors. We start with some preliminaries in Section 4.1 that allow us to characterize the shape of an optimal solution, and then we detail the FPTAS algorithm and its proof in Section 4.2.

Note that TRI-CRIT-CHAIN has a solution if and only if $\frac{S}{f_{\max}} \leq D$: all tasks must fit within the deadline when executed at the maximum speed. In this section, we therefore assume that $\frac{S}{f_{\max}} \leq D$, otherwise there is no solution.

4.1. *Characterization*

While TRI-CRIT-CHAIN is NP-hard even on a single processor, the problem has polynomial complexity if neither replication nor re-execution can be used. Indeed, each task is executed only once, and the energy is minimized when all tasks are running at the same speed (see [7]).

Lemma 4.1. *Without replication or re-execution, solving* TRI-CRIT-CHAIN *can be done in polynomial time, and each task is executed at speed* $\max\left(f_{\mathtt{rel}}, \frac{S}{D}\right)$ *on the same processor.*

Proof. For a linear chain of tasks, all tasks can be mapped on the same processor, and scheduled following the dependencies. No task may start earlier by using another processor because of precedence constraints, and all tasks run at the same speed [7]. Since there is no replication nor re-execution, each task must be executed at least at speed $f_{\mathtt{rel}}$ for the reliability constraint. If $S/f_{\mathtt{rel}} > D$, then the tasks should be executed at speed S/D so that the deadline constraint is matched (recall that $S = \sum_{i=1}^{n} w_i$), hence the result. This is feasible because $S/D \leq f_{\max}$. □

Next, accounting for replication and re-execution, we characterize the shape of an optimal solution. For linear chains, it turns out that with a single processor, only re-execution will be used, while with more than two processors, there is an optimal solution that does not use re-execution, but only replication. Furthermore, only two processors are needed to achieve the optimal solution.

Lemma 4.2 (Replication or re-execution) *When there is only one processor, it is optimal to only use re-execution to solve* TRI-CRIT-CHAIN. *When there are at least two processors, it is optimal to only use replication to solve* TRI-CRIT-CHAIN*, and only two processors are needed.*

Proof. With one processor, the result is obvious, since replication cannot be used. With more than one processor, if re-execution was used on task T_i, for $1 \leq i \leq n$, we can derive a solution with the same energy consumption and a smaller execution time by using replication instead of re-execution. Indeed, all instances of tasks T_j, for $j < i$, must finish before T_i starts its execution, and similarly, all instances of tasks T_j, for $j > i$, cannot start before both copies of T_i has finished its execution. Therefore, there are always at least two processors available when executing T_i for the first time, and the execution time is reduced when executing both copies of T_i in parallel (replication) rather than sequentially (re-execution). Finally, only two processors are needed because all tasks may be executed on a same processor, say P_1, and all backup tasks may also be executed on a same processor, say P_2. □

We further characterize the shape of an optimal solution by showing that two copies of the same task should always be executed at the same speed, and at the same time when the task is replicated.

Lemma 4.3 (Speed of the replicas) *For a linear chain, when a task is executed two times, it is optimal to have both replicas executed at the same speed, and at the same time if the task is replicated (two processors).*

Proof sketch. With one processor, we have seen in the previous lemma that it was optimal to only use re-execution. The proof for re-execution can be found in [8]: by convexity of the energy and reliability functions, it is always advantageous to execute two times the task at the same speed, even if the application is not a linear chain.

With two or more processors, we have seen in the previous lemma that it was optimal to only use replication, and that only two processors were needed. Let us consider a solution for which there exists i such that task T_i is executed twice at speeds $f^{(1)} < f^{(2)}$. Then the solution where task T_i is executed twice at speed $\frac{f^{(1)}+f^{(2)}}{2}$ at the same time (in parallel on both processors) meets the reliability and makespan constraints, and has a lower energy consumption, because of the convexity of the energy and reliability functions. Technical details can be found in the companion research report [6]. □

We can further characterize an optimal solution by providing detailed information about the execution speeds of the tasks, depending on whether they are executed only once, re-executed, or replicated.

Lemma 4.4. *If $D > \frac{S}{f_{rel}}$, then in any optimal solution of* TRI-CRIT-CHAIN*, all tasks that are neither re-executed nor replicated are executed at speed f_{rel}.*

Proof sketch. The proof for $p = 1$ (re-execution) can be found in [8]. For $p \geq 2$, then from Lemma 4.2, there are replications but no re-executions. We consider two tasks, one that is replicated and one that is executed only once. Then, if the speed of this last task is strictly greater than $f_{\texttt{rel}}$, we show that we can decrease its speed down to $f_{\texttt{rel}}$ and increase the speed of the replicated task, while keeping the deadline and decreasing the energy consumption. Reliability constraints still hold. Technical details can be found in the companion research report [6]. □

Let V_r be the subset $V_r \subseteq V$ of tasks that are either re-executed or replicated. We denote by X the total weight of these tasks, i.e., $X = \sum_{T_i \in V_r} w_i$. According to Lemma 4.4, the other tasks take a time $\frac{S-X}{f_{\texttt{rel}}}$, and the remaining time available for tasks of V_r is $D - \frac{S-X}{f_{\texttt{rel}}}$. Intuitively, a good solution would be such that all tasks are executed at the same speed $f_{\texttt{re-ex}}$, as small as possible, so that the deadline constraint is met, as illustrated in Figure 1. We must also ensure that $f_{\texttt{re-ex}}$ is not smaller than $f_{\min}$, and if this speed allows each task of V_r to meet the reliability constraint, then we can derive the energy of a schedule.

Following Lemma 4.4, we are able to precisely define $f_{\texttt{re-ex}}$, and give a closed form expression of the energy of a schedule when $f_{\texttt{re-ex}}$ is large enough. The proof uses a convexity argument, and it can be found in [6].

Corollary 4.1. *Given a subset V_r of tasks re-executed or replicated, let $X = \sum_{T_i \in V_r} w_i$, and*

$$f_{re\text{-}ex} = \begin{cases} \max\left(f_{\min}, \frac{2X}{Df_{rel}-S+X} f_{rel}\right) & \text{if } p = 1; \\ \max\left(f_{\min}, \frac{X}{Df_{rel}-S+X} f_{rel}\right) & \text{if } p \geq 2. \end{cases}$$

Then, if $f_{re\text{-}ex} \geq \max_{T_i \in V_r} f_{\inf,i}$, all tasks of V_r are executed twice at speed $f_{re\text{-}ex}$, and the optimal energy consumption is

$$(S - X) f_{rel}^2 + 2X f_{re\text{-}ex}^2. \tag{2}$$

Note that the energy consumption only depends on X, and therefore TRI-CRIT-CHAIN *is equivalent in this case to the problem of finding the optimal set of tasks that have to be re-executed or replicated.*

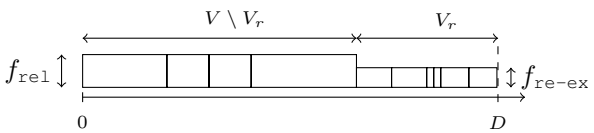


Figure 1. Illustration of the set V_r and $f_{\texttt{re-ex}}$

Re-execution speeds. We are now ready to compute the optimal solution, given a subset $V_r \subseteq V$. We have not accounted yet for tasks $T_i \in V_r$ such that $f_{\text{inf},i} > f_{\texttt{re-ex}}$. (Recall that $f_{\text{inf},i}$ is the minimum speed at which T_i may be executed to satisfy the reliability constraint). In this case, T_i is executed at speed $f_{\text{inf},i}$, and all the other tasks are (tentatively) executed at a new speed $f^{\texttt{new}}_{\texttt{re-ex}} \leq f_{\texttt{re-ex}}$ such that D is exactly met. We do this iteratively until there are no more tasks T_i such that $f_{\text{inf},i} > f^{\texttt{new}}_{\texttt{re-ex}}$. Using the procedure COMPUTE_$V_l(V_r)$ (see Algorithm 1), we can therefore compute the optimal energy consumption in a time polynomial in $|V_r|$. We denote by V_l the set of tasks that are re-executed or replicated at speed $f_{\text{inf},i}$ (it is a subset of V_r, the set of tasks that are re-executed or replicated). Note that all tasks of $V_r \setminus V_l$ are executed at the speed $f_{\texttt{re-ex}}$ returned by COMPUTE_$V_l(V_r)$.

Algorithm 1: Computing re-execution speeds; tasks in V_r are re-executed or replicated.

procedure COMPUTE_$V_l(V_r)$
begin
$V_l^{(0)} = \emptyset$;

$$f^{(0)}_{\texttt{re-ex}} = \begin{cases} \max\left(f_{\min}, \frac{2X}{Df_{\texttt{rel}} - S + X} f_{\texttt{rel}}\right) & \text{if } p = 1; \\ \max\left(f_{\min}, \frac{X}{Df_{\texttt{rel}} - S + X} f_{\texttt{rel}}\right) & \text{if } p \geq 2. \end{cases}$$

$j = 0$;
while $j = 0$ *or* $V_l^{(j)} \neq V_l^{(j-1)}$ **do**
$j := j + 1$;
$V_l^{(j)} = \left\{T_i \in V_r \mid f_{\text{inf},i} > f^{(j-1)}_{\texttt{re-ex}}\right\}$;

$$f^{(j)}_{\texttt{re-ex}} = \begin{cases} \max\left(f_{\min}, \frac{\sum_{T_i \in V_r \setminus V_l^{(j)}} 2w_i}{D - \frac{S-X}{f_{\texttt{rel}}} - \sum_{T_i \in V_l^{(j)}} \frac{2w_i}{f_{\text{inf},i}}}\right) & \text{if } p = 1; \\ \max\left(f_{\min}, \frac{\sum_{T_i \in V_r \setminus V_l^{(j)}} w_i}{D - \frac{S-X}{f_{\texttt{rel}}} - \sum_{T_i \in V_l^{(j)}} \frac{w_i}{f_{\text{inf},i}}}\right) & \text{if } p \geq 2. \end{cases}$$

return $(V_l^{(j)}, f^{(j)}_{re\text{-}ex})$;

Let $(V_l, f_{\texttt{re-ex}})$ be the result of COMPUTE_$V_l(V_r)$. Then the optimal energy consumption is $(S - X)f^2_{\texttt{rel}} + \sum_{T_i \in V_l} 2w_i f^2_{\text{inf},i} + \sum_{T_i \in V_r \setminus V_l} 2w_i f^2_{\texttt{re-ex}}$.

Lemma 4.5. *If* $D > \frac{S}{f_{rel}}$, TRI-CRIT-CHAIN *can be solved using an exponential time exact algorithm.*

Proof. The algorithm computes for every subset V_r of tasks the energy consumption if all tasks in this subset are re-executed, and it chooses a subset with the minimal energy consumption, that corresponds to an optimal solution. It takes an exponential time to compute every subset $V_r \subseteq V$, with $|V| = n$. □

Thanks to Corollary 4.1, we are also able to identify problem instances that can be solved in polynomial time.

Theorem 1. TRI-CRIT-CHAIN can be solved in polynomial time in the following cases:

(1) $D \leq \frac{S}{f_{\texttt{rel}}}$ (no re-execution nor replication);

(2) $p = 1$, $D \geq \frac{1+c}{c}\frac{S}{f_{\texttt{rel}}}$, where c is the only positive solution to the polynomial $7X^3+21X^2-3X-1=0$, and hence $c = 4\sqrt{\frac{2}{7}}\cos\frac{1}{3}(\pi - \tan^{-1}\frac{1}{\sqrt{7}}) - 1 \quad (c \approx 0.2838)$, and for $1 \leq i \leq n$, $f_{\text{inf},i} \leq \frac{2c}{1+c} f_{\texttt{rel}}$ (all tasks can be re-executed);

(3) $p \geq 2$, $D \geq 2\frac{S}{f_{\texttt{rel}}}$, and for $1 \leq i \leq n$, $f_{\text{inf},i} \leq \frac{1}{2} f_{\texttt{rel}}$ (all tasks can be replicated).

Proof sketch. First, when $D \leq \frac{S}{f_{\texttt{rel}}}$, the optimal solution is to execute each task only once, at speed $\frac{S}{D}$. When $D > \frac{S}{f_{\texttt{rel}}}$, we can show that the minimum of the energy function is reached when the total weight of the re-executed or replicated tasks is

$$X = \begin{cases} c(Df_{\texttt{rel}} - S) & \text{if } p = 1; \\ (Df_{\texttt{rel}} - S) & \text{if } p \geq 2. \end{cases} \tag{3}$$

Necessarily, when this total weight is greater than S, the optimal solution is to re-execute or replicate all the tasks, hence the theorem. In the detailed proof, in order to establish the bounds stated in the theorem, we consider the two cases $p = 1$ and $p \geq 2$ (see [6]).

For $p = 1$, the goal is to minimize $E(X) = (S-X)f_{\texttt{rel}}^2 + 2X\left(\frac{2X}{Df_{\texttt{rel}}-S+X} f_{\texttt{rel}}\right)^2$. We differentiate E and obtain that the minimum is reached when $7X^3 + 21(Df_{\texttt{rel}} - S)X^2 - 3(Df_{\texttt{rel}} - S)^2X - (Df_{\texttt{rel}} - S)^3 = 0$, hence the solution $X = c(Df_{\texttt{rel}} - S)$. When $X \geq S$, re-executing each task is the best strategy to minimize the energy consumption, and that corresponds to the case $D \geq \frac{1+c}{c}\frac{S}{f_{\texttt{rel}}}$. We show in the detailed proof that if $f_{\text{inf},i} > f_{\texttt{re-ex}}$ for some task T_i, it is still optimal to re-execute task T_i at speed $f_{\text{inf},i}$, and then a new re-execution speed will be used. Algorithm 1 returns tasks that are executed at speed $f_{\text{inf},i}$, together with the re-execution speed for all the other tasks.

For $p = 2$, the reasoning is similar with $E(X) = (S - X)f_{\texttt{rel}}^2 + 2X\left(\frac{X}{Df_{\texttt{rel}}-S+X} f_{\texttt{rel}}\right)^2$. □

4.2. *FPTAS for* TRI-CRIT-CHAIN

We derive in this section a fully polynomial-time approximation scheme (FPTAS) for TRI-CRIT-CHAIN, based on the FPTAS for SUBSET-SUM [16], and the results of Section 4.1. Without loss of generality, we use the term *replication* for either re-execution or replication, since both scenarios have already been clearly identified. The problem consists in identifying the set of replicated tasks V_r, and then the optimal solution can be derived from Corollary 4.1; it depends only on the total weight of these tasks, $X = \sum_{T_i \in V_r} w_i$.

Note that we do not account in this section for $f_{\text{inf},i}$ or $f_{\min}$ for readability reasons: $f_{\text{inf},i}$ can usually be neglected because $\lambda_0 w_i/f$ is supposed to be very small whatever f, and $f_{\min}$ simply adds subcases to the proofs (rather than an execution at speed f, the speed should be $\max(f, f_{\min})$).

First, we introduce a few preliminary functions in Algorithm 2, and we exhibit their properties. These are the basis of the approximation algorithm.

When $D > \frac{S}{f_{\text{rel}}}$, X-OPT$(V, D, p)$ returns the optimal value for the weight X of the subset of replicated tasks V_r, i.e., the value that minimizes the energy consumption for TRI-CRIT-CHAIN, according to Equation (3). The optimality comes directly from the proof of Theorem 1.

Given a value X, ENERGY(V, D, p, X) returns the optimal energy consumption when a subset of tasks of total weight X is replicated.

Then, the function TRIM(L, ε, X) trims a sorted list of numbers $L = [L_0, \cdots, L_{m-1}]$ in time $O(m)$, given L and ε. L is sorted into non decreasing order. The function returns a trimmed list, where two consecutive elements differ by at least a factor $(1 + \varepsilon)$, except the last element, that is the smallest element of L strictly greater than X. This trimming procedure is quite similar to that used for SUBSET-SUM [16], except that the latter keeps only elements lower than X. Indeed, SUBSET-SUM can be expressed as follows: given n strictly positive integers $a_1, \ldots, a_n$, and a positive integer X, we wish to find a subset I of $\{1, \ldots, n\}$ such that $\sum_{i \in I} w_i$ is as large as possible, but not larger than X. In our case, the optimal solution may be obtained either by approaching X by below or by above.

Given a list $L = [L_0, \ldots, L_{m-1}]$, ADD-LIST$(L, x)$ adds element x at the end of list L (i.e., it returns the list $[L_0, \ldots, L_{m-1}, x]$); $L + w$ is the list $[L_0 + w, \ldots, L_{m-1} + w]$; and MERGE-LISTS$(L, L')$ is merging two sorted lists (and returns a sorted list).

Finally, the approximation algorithm is APPROX-CHAIN(V, D, p, ε) (see Algorithm 2), where $0 < \varepsilon < 1$, and it returns an energy consumption E that is not greater than $(1 + \varepsilon)$ times the optimal energy consumption.

We now prove that this approximation scheme is an FPTAS:

Theorem 2. APPROX-CHAIN is a fully polynomial-time approximation scheme for TRI-CRIT-CHAIN.

Proof sketch. We assume that

- if $p = 1$, then $\frac{S}{f_{\text{rel}}} < D < \frac{1+c}{c}\frac{S}{f_{\text{rel}}} < 5\frac{S}{f_{\text{rel}}}$;
- if $p \geq 2$, then $\frac{S}{f_{\text{rel}}} < D < 2\frac{S}{f_{\text{rel}}}$;

otherwise the optimal solution is obtained in polynomial time (see Theorem 1, where the definition of c is also given).

Let $I_{\text{inf}} = \{V' \subseteq V \mid w(V') \leq \text{X-OPT}(V, D, p)\}$, and $I_{\text{sup}} = \{V'' \subseteq V \mid w(V'') > \text{X-OPT}(V, D, p)\}$. Note that I_{inf} is not empty, since $\emptyset \in I_{\text{inf}}$.

First we characterize the solution with the following lemma:

Lemma 4.6. *Suppose $D > \frac{S}{f_{rel}}$. Then in the solution of* TRI-CRIT-CHAIN*, the subset of replicated tasks V_r is either an element $V' \in I_{\text{inf}}$ such that $w(V')$ is maximum, or an element $V'' \in I_{\text{sup}}$ such that $w(V'')$ is minimum.*

Proof. Recall first that according to Lemma 4.4, the energy consumption of a linear chain is not dependent on the number of tasks replicated, but only on the sum of their weights.

Algorithm 2: Approximation algorithm for TRI-CRIT-CHAIN.

function X-OPT(V, D, p)
begin
  $S = \sum_{T_i \in V} w_i$;
  if $p = 1$ **then return** $c(Df_{rel} - S)$;
  else return $Df_{rel} - S$;

function ENERGY(V, D, p, X)
begin
  $S = \sum_{T_i \in V} w_i$;
  if $p{=}1$ **then return** $(S{-}X)f_{rel}^2{+}2X\left(\max\left(f_{\min}, \frac{2X}{Df_{rel}-S+X} f_{rel}\right)\right)^2$;
  else return $(S-X)f_{rel}^2 + 2X\left(\max\left(f_{\min}, \frac{X}{Df_{rel}-S+X} f_{rel}\right)\right)^2$;

function TRIM(L, ε, X)
begin
  $m = |L|$; $L = [L_0, \ldots, L_{m-1}]$; $L' = [L_0]$; $last = L_0$;
  for $i = 1$ **to** $m - 1$ **do**
    if *(*$last \leq X$ *and* $L_i > X$*) or* $L_i > last \times (1 + \varepsilon)$ **then**
      $L' =$ ADD-LIST(L', L_i); $last = L_i$;
  return L'*;*

function APPROX-CHAIN(V, D, p, ε)
begin
  $X = \lfloor$X-OPT$(V, D, p)\rfloor$; $n = |V|$; $L^{(0)} = [0]$;
  for $i = 1$ **to** n **do**
    $L^{(i)} =$ MERGE-LISTS$(L^{(i-1)}, L^{(i-1)} + w_i)$;
    $L^{(i)} =$ TRIM$(L^{(i)}, \varepsilon/(28 \times 2n), X)$;
  Let $Y_1 \leq Y_2$ be the two largest elements of $L^{(n)}$;
  return $\min($ENERGY$(V, D, p, Y_1),$ ENERGY$(V, D, p, Y_2))$*;*

Then the lemma is obvious by convexity of the functions, and because X-OPT returns the optimal value of X, the weight of the replicated tasks. Therefore, the closest the weight of the set of replicated tasks is to the optimal weight, the better the solution is. □

We are now ready to give a sketch of the proof for Theorem 2. Let $X_1 = \max_{V_1 \in I_{\inf}} w(V_1)$, and $X_2 = \min_{V_2 \in I_{\sup}} w(V_2)$. Thanks to Lemma 4.6, the optimal set of replicated tasks V_o is such that $X_o = w(V_o) = X_1$ or $X_o = X_2$. The corresponding

energy consumption is (Corollary 4.1):

$$E_{opt} = \begin{cases} (S - X_o) f_{\text{rel}}^2 + \frac{(2X_o)^3}{(Df_{\text{rel}} - S + X_o)^2} f_{\text{rel}}^2 & \text{if } p = 1 \\ (S - X_o) f_{\text{rel}}^2 + \frac{2X_o^3}{(Df_{\text{rel}} - S + X_o)^2} f_{\text{rel}}^2 & \text{if } p \geq 2 \end{cases}$$

The solution returned by APPROX-CHAIN corresponds either to Y_1 or to Y_2, where Y_1 and Y_2 are the two largest elements of the trimmed list. We can then prove that at least one of these two elements, denoted X_a, is such that $X_a \leq X_o \leq (1 + \varepsilon')X_a$, where $\varepsilon' = \frac{\varepsilon}{28}$. Next, we show that the energy E_a obtained with this value X_a is such that $E_{opt} \leq E_a \leq (1 + \varepsilon)E_{opt}$ (see [6] for details).

The energy consumption returned by APPROX-CHAIN, denoted as E_{algo}, is such that $E_{algo} \leq E_a$, since we take the minimum out of the consumption obtained for Y_1 or Y_2, and X_a is either Y_1 or Y_2. Therefore, $E_{algo} \leq (1 + \varepsilon)E_{opt}$.

It is clear that the algorithm is polynomial both in the size of the instance and in $\frac{1}{\varepsilon}$, given that the trimming function and APPROX-CHAIN have the same complexity as in the original approximation scheme for SUBSET-SUM (see [16]), and all other operations are polynomial in the problem size (X-OPT, ENERGY).

5. Independent tasks

In this section, we focus on the problem of scheduling independent tasks, TRI-CRIT-INDEP. Similarly to TRI-CRIT-CHAIN, we know that TRI-CRIT-INDEP is NP-hard, even on a single processor, because the problem is then identical to the chain problem (except that the order of tasks does not matter) [8]. We first prove in Section 5.1 that there exists no constant factor approximation algorithm for this problem, unless P=NP. We discuss and characterize solutions to TRI-CRIT-INDEP in Section 5.2, while highlighting the intrinsic difficulty of the problem. The core result is a constant factor approximation algorithm with a relaxation on the constraint on the makespan (Section 5.3).

5.1. *Inapproximability of* TRI-CRIT-INDEP

First, note that it is more difficult to characterize the feasibility of the problem with independent tasks when $p \geq 2$ than for TRI-CRIT-CHAIN. Indeed, deciding whether there is a solution or not is NP-hard, while for TRI-CRIT-CHAIN, there is a solution if and only if executing each task at speed $f_{\max}$ matches the deadline.

The feasibility problem is defined as follows: given a problem instance of TRI-CRIT-INDEP, does there exist a solution that matches the deadline and reliability constraints?

Lemma 5.1. *The feasibility problem of* TRI-CRIT-INDEP *is NP-complete.*

Proof. The feasibility problem of TRI-CRIT-INDEP is obviously in NP: given a solution to TRI-CRIT-INDEP, including the set of replicated or re-executed tasks, and the speed at which each task instance is executed, it is easy to check in polynomial time that the deadline is not exceeded and that the reliability constraints are satisfied.

To establish the completeness, we use a reduction from 2-PARTITION [19]. We consider an instance $\mathcal{I}_1$ of 2-PARTITION: given n strictly positive integers $a_1, \ldots, a_n$, does a

subset I of $\{1, \ldots, n\}$ exist such that $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$? Let $S = \sum_{i=1}^{n} a_i$. We build the following instance $\mathcal{I}_2$ of our problem with n independent tasks T_i, to be mapped on $p = 2$ processors, where

- task T_i has a weight $w_i = a_i$;
- $f_{\min} = f_{\mathtt{rel}} = f_{\max} = S/2$;
- $D = 1$.

If $\mathcal{I}_1$ has a solution I, then $\mathcal{I}_2$ also has a solution, where all tasks corresponding to integers in I are mapped on the first processor, and the other tasks are mapped on the second processor. Each task is executed only once at speed $f_{\max}$. Similarly, if $\mathcal{I}_2$ has a solution, then the indices of the set of tasks mapped on the first processor return a set I that is a solution to $\mathcal{I}_1$. Therefore, the feasibility problem of TRI-CRIT-INDEP is NP-complete.

□

Note that this holds because there is a maximum speed $f_{\max}$ in the model; otherwise, there is always a solution to the problem.

For TRI-CRIT-INDEP, a λ-approximation algorithm is a polynomial-time algorithm that returns a solution of energy consumption $E_{algo} \leq \lambda \times E_{opt}$, where E_{opt} is the energy consumption of the optimal solution, if there is a solution to the problem. Because the feasibility problem is NP-hard, we prove that there is no λ-approximation algorithm, unless P=NP, because such an algorithm would allow us to decide on the feasibility of the problem, and hence to solve in polynomial time an NP-complete problem.

Corollary 5.1. *For all $\lambda > 1$, there does not exist any λ-approximation algorithm for* TRI-CRIT-INDEP*, unless P=NP.*

Proof. This result is a corollary of Lemma 5.1, because if there is a λ-approximation algorithm for TRI-CRIT-INDEP, then its solution gives us the answer to the feasibility problem in polynomial time: if it returns a solution of energy consumption E_{algo}, then there is a solution to the problem, and if it does not return a solution, it means that there is no solution by definition of the approximation algorithm. Therefore, the inapproximability result is true unless P=NP. □

5.2. *Characterization*

As discussed in Section 1, the problem of scheduling independent tasks is usually close to a problem of load balancing, and can be efficiently approximated for various mono-criterion versions of the problem (minimizing the makespan or the energy, for instance). However, TRI-CRIT-INDEP turns out to be much harder, and cannot be approximated, as seen in Section 5.1, even when reliability is not a constraint. Note that if there is no maximum speed $f_{\max}$, then there exists a polynomial-time approximation algorithm for this problem with no reliability, see [2].

Adding reliability further complicates the problem, since we no longer have the property that on each processor, there is a constant execution speed for the tasks executed on

this processor. Indeed, some processors may process both tasks that are not replicated (or re-executed), hence at speed $f_{\texttt{rel}}$, and replicated tasks at a slower speed. Similarly to Section 4.2, we use the term *replication* for either re-execution or replication; if a task is replicated, it means it is executed two times, and it appears two times in the load of processors, be it the same processor or two distinct processors.

Furthermore, contrary to the TRI-CRIT-CHAIN problem, we do not always have the same execution speed for both executions of a task, as in Lemma 4.3:

Lemma 5.2. *In an optimal solution of* TRI-CRIT-INDEP*, if a task T_i is executed twice:*

- *if both executions are on the same processor, then both are executed at the same speed that is strictly smaller than $\frac{1}{\sqrt{2}} f_{rel}$;*
- *however, when the two executions of this task are on distinct processors, then they are not necessarily executed at the same speed. Furthermore, one of the two speeds can be greater than $\frac{1}{\sqrt{2}} f_{rel}$.*

Moreover, we have $w_i < \frac{1}{\sqrt{2}} D f_{rel}$.

Proof. When both executions occur on the same processor, it was shown in [8] that both are executed at the same speed that is strictly smaller than $\frac{1}{\sqrt{2}} f_{\texttt{rel}}$. Indeed, if the speed was greater than $\frac{1}{\sqrt{2}} f_{\texttt{rel}}$, replacing these two executions by a single execution at speed $f_{\texttt{rel}}$ would lead to a better energy consumption (and a lower execution time).

In the case of distinct processors, we give below an example in which the optimal solution uses different speeds for a replicated task, with one speed greater than $\frac{1}{\sqrt{2}} f_{\texttt{rel}}$. Note that one of the speeds is necessarily at most $\frac{1}{\sqrt{2}} f_{\texttt{rel}}$, otherwise a solution with only one execution of this task at speed $f_{\texttt{rel}}$ would be better, similarly to the case with re-execution.

Consider a problem instance with two processors, $f_{\texttt{rel}} = f_{\max}$, $D = \frac{6.4}{f_{\max}}$, and three tasks such that $w_1 = 5$, $w_2 = 3$, and $w_3 = 1$. Because of the time constraints, T_1 and T_2 are necessarily executed on two distinct processors, and neither of them can be re-executed on its processor. The problem consists in scheduling task T_3 to minimize the energy consumption. There are three possibilities:

- T_3 is executed only once on any of the processors, at speed $f_{\texttt{rel}} = f_{\max}$;
- T_3 is executed twice on the same processor; it is executed on the same processor as T_2, hence having an execution time of $D - \frac{w_2}{f_{\max}} = \frac{3.4}{f_{\max}}$, and therefore both executions are done at a speed $\frac{2}{3.4} f_{\max}$;
- T_3 is executed once on the same processor as T_1 at a speed $\frac{1}{1.4} f_{\max}$, and once on the other processor at a speed $\frac{1}{3.4} f_{\max}$.

It is easy to see that the minimum energy consumption is obtained with the last solution, and that $\frac{1}{1.4} f_{\max} > \frac{1}{\sqrt{2}} f_{\texttt{rel}}$, therefore we have exhibited a solution with two tasks not executed at the same speed, and with one of the speeds greater than $\frac{1}{\sqrt{2}} f_{\texttt{rel}}$.

Finally, note that since at least one of the executions of the task should be at a speed lower than $\frac{1}{\sqrt{2}} f_{\texttt{rel}}$, and since the deadline is D, in order to match the deadline, the weight of the replicated task has to be strictly lower than $\frac{1}{\sqrt{2}} D f_{\texttt{rel}}$.

□

Because of this lemma, usual load balancing algorithms are likely to fail, since processors handling only non-replicated tasks should have a much higher load, and speeds of replicated tasks may be very different from one processor to another in the optimal solution.

We now derive lower bounds on the energy consumption, that will be useful to design an approximation algorithm in the next section.

Lemma 5.3 (Lower bound without reliability) *The optimal solution of* TRI-CRIT-INDEP *cannot have an energy lower than* $\frac{S^3}{(pD)^2}$.

Proof. Let us consider the problem of minimizing the energy consumption, with a deadline constraint D, but without accounting for the constraint on reliability. A lower bound is obtained if the load on each processor is exactly equal to $\frac{S}{p}$, and the speed of each processor is constant and equal to $\frac{S}{pD}$. The corresponding energy consumption is $S \times \left(\frac{S}{pD}\right)^2$, hence the bound. □

However, if the speed $\frac{S}{pD}$ is small compared to $f_{\texttt{rel}}$, the bound is very optimistic since reliability constraints are not matched at all. Indeed, replication must be used in such a case. We investigate bounds that account for replication in the following, using the optimal solution of the TRI-CRIT-CHAIN problem.

Lemma 5.4 (Lower bound using linear chains) *For the* TRI-CRIT-INDEP *problem, the optimal solution cannot have an energy lower than the optimal solution to the* TRI-CRIT-CHAIN *problem on a single processor with a deadline* pD*, where the weight of each re-executed task is lower than* $\frac{1}{\sqrt{2}} D f_{\texttt{rel}}$.

Proof. We can transform any solution to the TRI-CRIT-INDEP problem into a solution to the TRI-CRIT-CHAIN problem with deadline pD and a single processor. Tasks are arbitrarily ordered as a linear chain, and the solution uses the same number of executions and the same speed(s) for each task. It is easy to see that the TRI-CRIT-INDEP problem is more constrained, since the deadline on each processor must be enforced. The constraint on the weights of the re-executed tasks comes from Lemma 5.2. Therefore, the solution to the TRI-CRIT-CHAIN problem is a lower bound for TRI-CRIT-INDEP. □

The optimal solution may however be far from this bound, since we do not know if the tasks that are re-executed on a chain with a long deadline pD can be executed at the same speed when the deadline is D. The constraint on the weight of the re-executed tasks allows us to improve slightly the bound, and this lower bound is the basis of the approximation algorithm that we design for TRI-CRIT-INDEP.

5.3. *Approximation algorithm for* TRI-CRIT-INDEP

We have seen in Section 5.1 that there exists no constant factor approximation algorithm for TRI-CRIT-INDEP, unless P=NP, even without accounting for the reliability constraint.

This is due to the constraint on the makespan and the maximum speed $f_{\max}$. Therefore, in order to provide a constant factor approximation algorithm, we relax the constraint on the makespan and propose a β-time λ-energy approximation algorithm, following the notations from [26]. The solution E_{algo} is such that $E_{algo} \leq \lambda \times E_{opt}$, where E_{opt} is the optimal solution with the deadline constraint D, and the makespan of the solution returned by the algorithm, M_{algo}, is such that $M_{algo} \leq \beta \times D$.

If the original problem with deadline D has no solution, because of the deadline relaxation, the β-time λ-energy approximation algorithm may or may not return a solution (contrarily to a λ-approximation algorithm that would not return any solution either), but then there is no guarantee to ensure because there is no optimal solution. Therefore, we do not consider such cases for proving the correctness and guarantee of the algorithm. In particular, we assume that for all i, $w_i/f_{\max} \leq D$, and that $S/pf_{\max} \leq D$, otherwise we know that there is no solution.

The result of Section 5.1 means that for all $\lambda > 1$, there is no 1-time λ-energy approximation algorithm for TRI-CRIT-INDEP, unless P=NP. Therefore, we present an algorithm that realizes a β-time $\left(1 + \frac{1}{\beta^2}\right)$-energy approximation, where β can be slightly smaller than 2 and can take any arbitrarily large value: $\beta \geq \max\left(2 - \frac{3}{2p+1}, 2 - \frac{p+2}{4p+2}\right)$.

Algorithm. In the first step of the algorithm, we schedule each task with a big weight alone on one processor, with no replication (at speed $\frac{w_i}{D}$). A task T_i is considered as *big* if $w_i \geq \max(\frac{S}{p}, Df_{\texttt{rel}})$. This step is done in polynomial time: we sort the tasks by non increasing weights, and then we check whether the current task is such that $w_i \geq \max(\frac{S}{p}, Df_{\texttt{rel}})$. If it is the case, we schedule the task alone on an unused processor and we let $S = S - w_i$ and $p = p - 1$. The procedure ends when the current task is small enough, i.e., all remaining tasks are such that $w_i < \max(\frac{S}{p}, Df_{\texttt{rel}})$, with the updated values of S and p. Note that there are always enough unused processors because selected big tasks are such that $w_i \geq \frac{S}{p}$, and therefore there cannot be more than p such tasks (and this is true at each step). When $p = 1$, either there is only one remaining task of size S, or there are only small tasks left.

These big tasks can be safely ignored in the remainder of the algorithm, hence the abuse of notations S and p for the remaining load and the remaining processors. Indeed, we will prove that this first step of the algorithm takes decisions that are identical to the optimal solution, and therefore these tasks that are executed once, alone on their processor, have the same energy consumption and the same deadline as in the optimal solution. The next step depends on the remaining load S:

- If $S > pDf_{\texttt{rel}}$, i.e., the remaining load is *large enough*, we do not use replication, but we schedule the tasks at speed $\frac{S}{pD}$, using a simple scheduling heuristic, LONGEST-PROCESSING-TIME [20]. Tasks are numbered by non increasing weights, and at each time step, we schedule the current task on the least loaded processor. Thanks to the lower bound of Lemma 5.3, the energy consumption is not greater than the optimal energy consumption, and we determine β such that the deadline is enforced.
- If $S \leq pDf_{\texttt{rel}}$, the previous bound is not good enough, and therefore we use

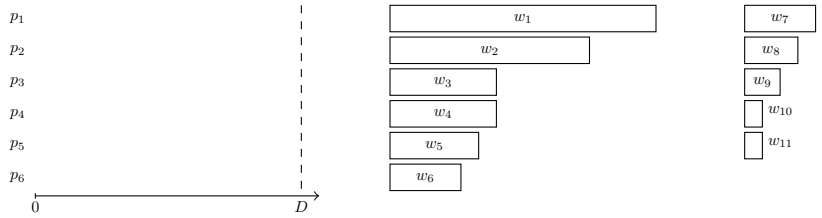
the FPTAS on a linear chain of tasks with deadline pD for TRI-CRIT-CHAIN (see Theorem 2). The FPTAS is called with

$$\varepsilon = \min\left(\frac{2w_{min}}{3S}\left(\frac{f_{\min}}{f_{\mathtt{rel}}}\right)^2, \frac{1}{3\beta^2}\right), \tag{4}$$

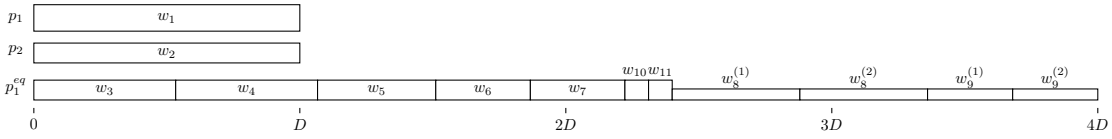
where $w_{min} = \min_{1\leq i\leq n} w_i$. Note that it is slightly modified so that only tasks of weight $w < \frac{1}{\sqrt{2}} D f_{\mathtt{rel}}$ can be replicated, and that we enforce a minimum speed $f_{\min}$. The FPTAS therefore determines which tasks should be executed twice, and it fixes all execution speeds.

We then use LONGEST-PROCESSING-TIME in order to map the tasks onto the p processors, at the speeds determined earlier. The new set of tasks includes both executions in case of replication, and tasks are sorted by non increasing execution times (since all speeds are fixed). At each time step, we schedule the current task on the least loaded processor. If some tasks cannot fit in one processor within the

(a) Input: six processors and eleven tasks



(b) Schedule the *big* tasks on p_1 and p_2, and call APPROX-CHAIN with deadline $(6-2)D$ on the remaining tasks



(c) Greedy algorithm to schedule the new tasks

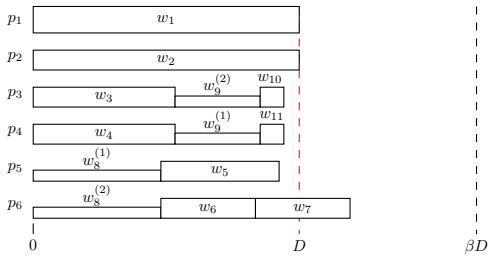


Figure 2. β-time $\left(1+\frac{1}{\beta^2}\right)$-energy approximation algorithm for independent tasks

deadline βD, we re-execute them at speed $\frac{w_i}{\beta D}$ on two processors. Thanks to the lower bound of Lemma 5.4, we can bound the energy consumption in this case.

We illustrate the algorithm on an example in Figure 2, where eleven tasks must be mapped on six processors. For each task, we represent its execution speed as its height, and its execution time as its width. There are two *big* tasks, of weights w_1 and w_2, that are each mapped on a distinct processor. Then, we have $p = 4$ and we call APPROX-CHAIN with deadline $4D$; tasks T_8 and T_9 are replicated. Finally, LONGEST-PROCESSING-TIME greedily maps all instances of the tasks, slightly exceeding the original bound D, but all tasks fit within the extended deadline.

This algorithm leads to the following theorem:

Theorem 3. For the problem TRI-CRIT-INDEP, there are β-time $\left(1 + \frac{1}{\beta^2}\right)$-energy approximation algorithms, for all $\beta \geq \max\left(2 - \frac{3}{2p+1}, 2 - \frac{p+2}{4p+2}\right)$, that run in polynomial time.

Proof sketch. The proof is quite involved and can be found in the companion research report [6]. We first prove the optimality of the first step of the algorithm, i.e., the optimal solution would schedule tasks of weight greater than $\max(\frac{S}{p}, Df_{\mathtt{rel}})$ alone on a processor.

Next, we tackle the case where the load is *large enough* ($S > pDf_{\mathtt{rel}}$), and we obtain a minimum on the approximation ratio of the deadline β.

Finally, we reuse this approximation ratio on the deadline to handle the remaining case where the FPTAS for TRI-CRIT-CHAIN is used, and we explain how to map the tasks at the speeds assigned by the FPTAS. We then check the energy consumption of the schedule to prove the approximation ratio on the energy consumption. □

6. Conclusion

In this paper, we have designed efficient approximation algorithms for the tri-criteria energy/reliability/makespan problem, using replication and re-execution to increase the reliability, and dynamic voltage and frequency scaling to decrease the energy consumption. Because of the antagonistic relationship between energy and reliability, this tri-criteria problem is much more challenging than the standard bi-criteria problem, which aims at minimizing the energy consumption with a bound on the makespan, without accounting for a constraint on the reliability of tasks.

We have tackled two classes of applications. For linear chains of tasks, we propose a fully polynomial-time approximation scheme. However, we show that there exists no constant factor approximation algorithm for independent tasks, unless P=NP, and we are able in this case to propose an approximation algorithm with a relaxation on the makespan constraint.

Discussion on the energy function. In this paper, we considered the energy function to be $E_i(f) = \mathcal{E}xe(w_i, f) \times f^\alpha$ with $\alpha = 3$. Most of the results would still hold with $\alpha \geq 2$, because they are based on the convexity of the energy function. We would however need to

define c_α as the only positive root of the equation $2^\alpha X^{\alpha-1}(X+\alpha)-(1+X)^\alpha = 0$ instead of c in Theorem 1. The fact that there is only one positive root is harder to prove for α as the equation is not a polynomial in the general case, but it may be proven using the theory of implicit functions. Furthermore, we do not have a nice closed-form formula for c_α, but one can show that c_α is an increasing function of α. Finally, in the proof of Theorem 2, we use the fact that we can bound $\frac{1+c_\alpha}{c_\alpha}$ by a constant. This constant becomes 7.5 if $\alpha \geq 2$, and therefore the function APPROX-CHAIN needs to be modified: 28 should be replaced by 60.

Future work. As future work, it may be possible to improve the deadline relaxation by using an FPTAS to schedule independent tasks [9] rather than LONGEST-PROCESSING-TIME [20]. Also, an open problem is to find approximation algorithms for the tri-criteria problem with an arbitrary graph of tasks. Even though efficient heuristics have been designed with re-execution of tasks (but no replication) [8], it is not clear how to derive approximation ratios from these heuristics. It would be interesting to design efficient algorithms using replication and re-execution for the general case, and to prove approximation ratios on these algorithms. A first step would be to tackle fork and fork-join graphs, inspired by the study on independent tasks. Finally, more sophisticated models for reliability could also be considered, for instance to guarantee a global reliability constraint or to authorize more than one backup task.

Acknowledgements. The authors are with Université de Lyon, France. A. Benoit is with the Institut Universitaire de France. We would like to thank the reviewers for their comments and suggestions, which greatly improved the final version of this paper. This work was supported in part by the ANR *RESCUE* project.

References

[1] S. Albers. Energy-efficient algorithms. *Communications of the ACM*, 53(5):86–96, 2010.

[2] S. Albers, F. Müller, and S. Schmelzer. Speed scaling on parallel processors. In *Proceedings of the Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '07, pages 289–298, New York, NY, USA, 2007. ACM.

[3] N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid. Approximation schemes for scheduling. In *Proceedings of SODA'97, the 8th annual ACM-SIAM Symposium On Discrete Algorithms*, pages 493–500, Philadelphia, PA, USA, 1997. Society for Industrial and Applied Mathematics.

[4] A. Antoniadis and C.-C. Huang. Non-preemptive speed scaling. *Journal of Scheduling*, 16(4):385–394, 2013.

[5] I. Assayad, A. Girault, and H. Kalla. Tradeoff exploration between reliability power consumption and execution time. In *Proceedings of SAFECOMP, the Conf. on Computer Safety, Reliability and Security*, Washington, DC, USA, 2011. IEEE CS Press.

[6] G. Aupy and A. Benoit. Approximation algorithms for energy, reliability and makespan optimization problems. Research Report 8107, INRIA, France, July 2014.

Available at graal.ens-lyon.fr/~abenoit.

[7] G. Aupy, A. Benoit, F. Dufossé, and Y. Robert. Reclaiming the energy of a schedule: models and algorithms. *Concurrency and Computation: Practice and Experience*, 2012.

[8] G. Aupy, A. Benoit, and Y. Robert. Energy-aware scheduling under reliability and makespan constraints. In *Proceedings of HiPC'2012, the IEEE Int. Conf. on High Performance Computing*, 2012. Also available at gaupy.org/?paper as INRIA Research report 7757.

[9] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*. Springer Verlag, 1999.

[10] H. Aydin and Q. Yang. Energy-aware partitioning for multiprocessor real-time systems. In *Proceedings of IPDPS, the Int. Parallel and Distributed Processing Symposium*, pages 113–121. IEEE CS Press, 2003.

[11] M. Baleani, A. Ferrari, L. Mangeruca, A. Sangiovanni-Vincentelli, M. Peri, and S. Pezzini. Fault-tolerant platforms for automotive safety-critical applications. In *Proceedings of Int. Conf. on Compilers, Architectures and Synthesis for Embedded Systems*, pages 170–177. ACM Press, 2003.

[12] E. Bampis, D. Letsios, and G. Lucarelli. A note on multiprocessor speed scaling with precedence constraints. In *Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures*, pages 138–142. ACM, 2014.

[13] N. Bansal, T. Kimbrel, and K. Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM*, 54(1):1 – 39, 2007.

[14] A. Benoit, A. Cavelan, Y. Robert, and H. Sun. Assessing general-purpose algorithms to cope with fail-stop and silent errors. In *Proceedings of the PMBS Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems, held with SC14*, Nov. 2014.

[15] A. Benoit, P. Renaud-Goud, and Y. Robert. On the performance of greedy algorithms for power consumption minimization. In *Proceedings of ICPP 2011, the Int. Conf. on Parallel Processing*, pages 454 –463, Sept. 2011.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to algorithms, third edition, 2009.

[17] V. Degalahal, L. Li, V. Narayanan, M. Kandemir, and M. J. Irwin. Soft errors issues in low-power caches. *IEEE Transactions on Very Large Scale Integration Systems*, 13:1157–1166, October 2005.

[18] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Survey*, 34:375–408, 2002.

[19] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[20] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17:416–429, 1969.

[21] M. A. Haque, H. Aydin, and D. Zhu. Energy-aware task replication to manage reliability for periodic real-time applications on multicore platforms. In *Proceedings of*

the Fourth IEEE International Green Computing Conference (IGCC'13), June 2013.

[22] R. Melhem, D. Mossé, and E. Elnozahy. The interplay of power management and fault recovery in real-time systems. *IEEE Transactions on Computers*, 53:217–231, 2004.

[23] X. Ni, E. Meneses, N. Jain, and L. V. Kalé. ACR: Automatic Checkpoint/Restart for Soft and Hard Error Protection. In *Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis*, SC '13. ACM, 2013.

[24] A. J. Oliner, R. K. Sahoo, J. E. Moreira, M. Gupta, and A. Sivasubramaniam. Fault-aware job scheduling for bluegene/l systems. In *Proceedings of IPDPS, the Int. Parallel and Distributed Processing Symposium*, pages 64–73, 2004.

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

[26] K. Pruhs, R. van Stee, and P. Uthaisombut. Speed scaling of tasks with precedence constraints. *Theor. Comp. Sys.*, 43:67–80, March 2008.

[27] T. Rauber, G. RÃ$\frac{1}{4}$nger, M. Schwind, H. Xu, and S. Melzner. Energy measurement, modeling, and prediction for processors with frequency scaling. *The Journal of Supercomputing*, 70(3):1451–1476, 2014.

[28] S. M. Shatz and J.-P. Wang. Models and algorithms for reliability-oriented task-allocation in redundant distributed-computer systems. *IEEE Transactions on Reliability*, 38:16–27, 1989.

[29] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 374–382. IEEE, 1995.

[30] D. Zhu. Reliability-aware dynamic energy management in dependable embedded real-time systems. In *Real-Time and Embedded Technology and Applications Symposium*, pages 397–407. IEEE CS Press, 2006.

[31] D. Zhu and H. Aydin. Energy management for real-time embedded systems with reliability requirements. In *Proceedings of ICCAD, the IEEE/ACM Int. Conf. on Computer-Aided Design*, pages 528–534, 2006.

[32] D. Zhu, R. Melhem, and D. Mossé. The effects of energy management on reliability in real-time embedded systems. In *Proceedings of ICCAD, the IEEE/ACM Int. Conf. on Computer-Aided Design*, pages 35–40, 2004.