Optimal Real-Time Application Execution Strategy for Meeting Reliability and Deadline Constraints with Minimal Energy Consumption on Uniprocessor

Zheng Li, Shangping Ren, Li Wang
{zli80}@hawk.iit.edu

Dept of CS, Illinois Institute of Technology
Chicago, IL, 60616
Optimal Real-Time Application Execution Strategy for Meeting Reliability and Deadline Constraints with Minimal Energy Consumption on Uniprocessor

Zheng Li, Shangping Ren, Li Wang
{zli80}@hawk.iit.edu

Dept of CS, Illinois Institute of Technology
Chicago, IL, 60616

Illinois Institute of Technology, Dept. of CS, IITCSTR-2012-0408
April 2012

Abstract

We study the problem of how to employ dynamic voltage frequency scaling technique to minimize the overall energy consumption for hard real-time applications under given reliability and deadline constraints. We first present a closed formula to analytically quantify the reliability real-time applications. We then introduce and formally prove the optimal execution strategies when processor’s frequency can be scaled continuously and discretely. Based on these strategies, task re-execution and checkpointing based algorithms are proposed for both continuous and discrete frequency scenarios. Experimental results show that our proposed approach can save up to 20% more energy than other heuristics existed in the literature.

1 Introduction

For the past decade, extensive power aware research has been conducted across different design abstraction levels, from the transistor level all the way to the system level [1, 2]. As more and more transistors are integrated into a single chip, the chip power consumption has been increasing exponentially, which has caused more stringent power/energy constraints to system designers. As a result, more and more aggressive power aware reduction techniques have been proposed (e.g using extremely low supply voltages and threshold voltages [3, 4]). While these techniques can greatly reduce energy consumption, the reliability of the entire application is degraded. Furthermore, with the continuing scaling of CMOS technologies and reducing the design margins for higher performance, soft errors of digital systems caused by transient faults occur more frequently than ever. Therefore, how to save the energy consumption most effectively and, at the same time, to ensure the system reliability has increasingly become a prominent issue for real-time system designers [5].

Improving reliability of a system and its energy saving performance are at odds for real-time systems. First, in order to recover from the transient errors, extra computing resources must be reserved so that programs can be reexecuted partially or entirely when program errors strike. This implies that part of the slack time must be reserved for the purpose of recovering program errors instead of being utilized for energy saving. The more transient faults that need to be recovered, the more resources need to be reserved, and thus the less energy efficiency the system can be. Moreover, even though the slack times are available, to exploit slack time in energy saving can also compromise the reliability. Earlier research in the area of power management under timing constraints has shown that Dynamic Voltage Frequency Scaling (DVFS) technique is one of the most effective technique to save energy consumption while still guarantees the satisfaction of timing constraints [6, 7, 8]. However, when scaling down the frequency and lowering down the device supplying voltage, the device fault rate increases and therefore the reliability of the system decreases.

A few papers have been published on the study of the tradeoffs between the energy reduction and reliability. For instance, Zhu et al. [9] provided the reliability-ware power management (RA-PM) scheme to address the trade-offs.
The RA-PM approach allocates a recovery task for every real-time task whose execution frequency is scaled down so that when an error does occur, the recover task can be executed to ensure the required reliability. In order to decide which tasks should be selected for execution under scaled down frequency for energy saving purpose, some heuristic approaches are proposed, such as longest task first (LTF) [10] and slack usage efficiency factor (SUEF) based heuristic algorithm [10].

It is not difficult to see that the RA-PM approach is conservative as it statically divides the available slack time to multiple recovery blocks for a pre-determined set of tasks. Zhao [11] proposed the shared recovery (SHR) technique. The SHR approach reserves a recovery block. The recovery block is not for a specific task, rather it is for any task that needs recover at run-time. However, this technique is only provisioned for single fault recovery. In other words, when a fault occur during an execution of a task, the recovery block is used to recover the faulty task, then the remaining tasks are executed under the maximum processing frequency. Zhao et al extended the work to allow multiple fault recoveries [12]. The extended global shared recovery (GSHR) technique also allows that the reliability constraint be set to an arbitrary target value. A heuristic approach, the Incremental Reliability Configuration (IRCS), is provided to find the number of recovery tasks and task execution frequency assignments.

Backward fault recovery strategy are commonly used for fault recovery, which restore the system state to a previous safe state and repeat the computation [13, 14]. Although task re-execution and checkpointing are common techniques for backward fault recovery, most of recent work in real-time community has nevertheless focused on using task re-execution for fault recovery [10, 11, 12]. Zhu et al. [15] has also noticed that the checkpoint cost has significant impact on total energy savings. In fact, their data shows when the checkpointing cost exceeds 10% of the task’s computation workload, no energy saving can be achieved.

Rather than exploring different heuristic approaches to balance the trade offs between reliability and energy, we theoretical analysis the problem and propose an optimal execution strategy that meets both reliability and deadline constraints but with minimal energy consumption for hard real-time applications. We also re-evaluate the two commonly used fault recovery techniques, i.e., re-execution and checkpointing. In fact, if the number of checkpointing is appropriately selected, the checkpointing based recovery approach can be more effective from energy saving perspective.

The main contributions of the paper can be summarized below:

• develop an optimal application execution strategy that meets both reliability and deadline constraints with minimal energy consumption and theoretically prove the optimality of the execution strategy.

• re-evaluate the two fault recovery strategies, i.e., task re-execution based recovery and checkpointing based recovery, and show that checkpointing based recovery can be more effective if used appropriately.

• experimentally confirm and verify the theoretical conclusions and compare the results with two existing work in the literature.

The rest of the paper is organized as following. In Section 2, we give system models. Based on the models, we formally define the problem the paper is to address, i.e., find an optimal application execution strategy that meets both reliability and deadline constraints but with minimal energy consumption on a uniprocessor.

## 2 System Models and Problem Formulation

In this section, we first give the models our work is based upon and then define the problem the paper is to address, i.e., find an optimal application execution strategy that meets both reliability and deadline constraints but with minimal energy consumption on a uniprocessor.

### 2.1 Models and Assumptions

#### Application and Processor Model

We consider a real-time embedded application $A$ with $m$ independent tasks $\{\tau_1, ..., \tau_m\}$ sharing a common end-to-end deadline $D$. All tasks are executed on a DVFS enabled processor, with working frequency between the range $[f_{\min}; f_{\max}]$, where $f_{\min}$ and $f_{\max}$ are the minimum and maximum frequency, respectively. We assume that the frequency values are normalized with respected to $f_{\max}$, that is, $f_{\max} = 1$. The worst case execution time (WCET)
of task $\tau_i$ under the maximum processor frequency $f_{max}$ is given as $c_i$. When the processor runs at frequency of $f$, with $f_{min} \leq f \leq f_{max}$, the WCET of task $\tau_i$ becomes $\frac{c_i}{f}$.

**Power Consumption Model**

We adopt the similar system-level power model as given in [16, 17]. Specifically,

$$P = P_s + h(P_{\text{ind}} + P_d),$$

and

$$P_d = C_{ef} f^{C_m}$$

Where $P_s$ is the static power, such as maintaining the basic circuits and keeping the clock running, which can only be avoided by powering off the system. We assume the system is always on and hence $P_s$ is always consumed. $P_{\text{ind}}$ is independent of processing frequency, which is constant and can be avoided by putting the system to sleep, i.e., set $h = 0$. While $h = 1$, the system is in active state. $P_d$ is the frequency-dependent power consumption, $C_{ef}$ and $C_m(\geq 2)$ are effective switching capacitance and dynamic power exponent, respectively. They are both system-dependent constants. $f$ is the processing frequency.

Under the power consumption model, the energy consumption of task $T_i$ under frequency $f_i$ can be expressed as:

$$E_i(f_i) = (P_{\text{ind}} + C_{ef}(f_i)^{C_m}) \cdot t_i$$

Where $t_i = \frac{c_i}{f_i}$, which is the execution time of task $i$ under $f_i$.

From (3), it is not difficult to see that scaling down the processing frequency can save frequency-dependent energy, but, on the other hand, it can also increase the frequency-independent energy because of longer execution time due to lower frequency. Hence, the Energy-Efficient frequency ($f_{ee}$) exists, under which, further scaling down the processing frequency will increase the total energy consumption. Previous research studies [17] have given the definition:

$$f_{ee} = C_m \sqrt{\frac{P_{\text{ind}}}{C_{ef}(C_m - 1)}}$$

**Transient Fault and Reliability Model**

In this paper, we only consider soft errors caused by transient faults. We take the same assumption as in [17] that the arrival of transient faults follows Poisson distribution. The average fault arrival rate at frequency $f$ ($< f_{max}$) can be expressed as [5]:

$$\lambda(f) = \lambda_0 10^{\frac{d(1 - f)}{f_{\text{min}}}}$$

where $\lambda_0$ is the average error rate at $f_{max}$ and $d(\geq 3)$ is a system-dependent constant, representing the sensitivity of transient fault due to DVFS.

To simplify the representation, formula (5) can be written as:

$$\lambda(f) = \hat{\lambda}_0 10^{-\hat{d}f}$$

Where $\hat{\lambda}_0 = \lambda_0 10^{-\frac{d}{f_{\text{min}}}}$, $\hat{d} = \frac{d}{f_{\text{min}}}$.

If we define the reliability of the task as the probability of completing the task successfully, and the processing frequency of task $i$ is denoted as $f_i$, the reliability of task $i$ can be modeled as:

$$R_i(f_i) = e^{-\lambda(f_i) \cdot s_i}$$

Where $s_i = \frac{c_i}{f_i}$, which is the execution time of task $i$ under $f_i$.

From formula (3) and (6), scaling down the frequency $f$ reduces energy consumption, but also decrease the reliability. However, if we can tolerate faults, i.e., be able to recover from faults when they occur, we can improve the reliability. Commonly used fault recovery techniques are task re-execution and checkpointing. For task re-execution based recovery technique, fault detection takes place at the end of each task and the whole task needs to be re-executed if a fault occurs; whereas for checkpointing based recovery strategy, a task is divided into several short intervals by inserted checkpoints, fault detection is done at the end of each interval. Hence, fault recovery only requires re-executing the interval where fault occurs.
According to the property of poisson distribution, when the fault arrival rate is $\lambda$, the probability of at most $k$ fault arriving in the time interval of $[0, t]$ can be expressed as[18]:

$$R(\lambda, t, k) = \sum_{i=0}^{k} \frac{(\lambda t)^i e^{-\lambda t}}{i!}$$  

(8)

Assuming recoveries are executed under $f_{\text{max}}$. In order to guarantee that $k$ faults are tolerated, the longest $k$ fault recovery durations have to be reserved. We use processing state and recovery state to distinguish the normal application execution and fault recovery, respectively. Let $\text{rec}(h)$ to denote $h$th longest fault recovery, then for tolerating $k$ faults, the length of recovery stage is $\sum_{h=1}^{k} \text{rec}(j)$.

2.2 Problem Definition

Before we formally define the problem of finding an optimal application execution strategy, we first introduce a lemma.

**Lemma 1.** Assume an application’s total execution time is divided into $t_1, t_2, \cdots, t_n$ segments, and segment $t_i$ is executed under frequency $f_i$, $k$ faults need to be tolerated, all fault recoveries are executed under $f_{\text{max}}$, and $\text{rec}(h)$ is the $h$th longest fault recovery duration, then the reliability of the application, which is defined as the probability of completing the application successfully, can be represented as:

$$R_{A}(\langle f_i, t_i \rangle_n, k) = \sum_{i=0}^{k} \frac{(\sum_{j=1}^{n} \lambda(f_j) t_j)^i}{i!} e^{-\sum_{j=1}^{n} \lambda(f_j) t_j} e^{-\lambda(f_{\text{max}})(\sum_{h=1}^{k} \text{rec}(h))}$$

(9)

where $\langle f_i, t_i \rangle_n = [(f_1, t_1), (f_2, t_2), \cdots, (f_n, t_n)]$.

To simplify the representation, we define:

$$r(x, k) = \sum_{i=0}^{k} \frac{x^i e^{-x}}{i!} e^{-\lambda(f_{\text{max}})(\sum_{h=1}^{k} \text{rec}(h))}$$

(10)

then formula (9) can be re-written as:

$$R_{A}(\langle f_i, t_i \rangle_n, k) = r(\sum_{i=1}^{n} \lambda(f_i) t_i, k)$$

(11)

To avoid diverging from the main flow of the paper, we leave the proof in the appendix section.

As the probability of fault occurrence is relatively small, the expected energy consumption for recovery stage is ignorable comparing to the energy consumption for processing stage. Hence, the expected energy consumption of the application can be approximately defined as:

$$E_{A}(\langle f_i, t_i \rangle_n, k) = \sum_{i=1}^{n} (P_{\text{ind}} + C_{\text{ef}}(f_i)^C m) \cdot t_i$$

(12)

For simplicity, we introduce a new function $\delta(x)$:

$$\delta(x) = P_{\text{ind}} + C_{\text{ef}} x^C m$$

Then formula (12) can be re-written as:

$$E_{A}(\langle f_i, t_i \rangle_n, k) = \sum_{i=1}^{n} \delta(f_i) t_i$$

(13)

We formally define the problem of finding an optimal application execution strategy for meeting reliability and deadline constraints with minimal energy cost on uniprocessor system, the RDE problem for short, as below.
Problem 1. Given the available processing frequencies \( f \), \( f_{\min} \leq f \leq f_{\max} \) with \( f_{\min} \geq f_{ee} \), application \( A = \{\tau_1, \ldots, \tau_m\} \) with the WCET time as \( c_i \) under \( f_{\max} \) for task \( \tau_i \), the application reliability requirement is \( R_g \) and the end-to-end deadline is \( D \), find the processing frequencies and corresponding execution duration \( (f_i, t_i)_n = [(f_1, t_1), \ldots, (f_n, t_n)] \), and the number of faults \( k \) to be tolerated during the processing stage with

**Objective:**

\[
\min E_A((f_i, t_i)_n, k)
\]

**Subject to:**

\[
R_A((f_i, t_i)_n, k) \geq R_g \\
\sum_{i=1}^{n} t_i + \sum_{h=1}^{k} rec(h) \leq D
\]

Where \( rec(h) \) indicates \( h \)th longest fault recovery, which depends on the fault recovery strategy (checkpointing or task re-execution).

We address the problem in two steps: we first consider the case when the processor frequency can be continuously scaled with the range of \( [f_{\min}, f_{\max}] \), and then consider the case when there is only a set of discrete frequencies available for scaling.

\section{Optimal Application Execution Strategy under Continuous Frequency Scaling}

In this section, we assume the processing frequency can be continuously scaled on a processor and call the problem defined in Section 2.2 under the assumption of the C-RDE problem.

\subsection{Optimal Application Execution Strategy and its Properties}

In this section, we will give the theoretical analysis of the optimal execution strategy for C-RDE problem.

**Lemma 2** (Uniform Frequency). Given an application \( A \) which has a set of independent tasks \( \{\tau_1, \ldots, \tau_i, \ldots, \tau_m\} \) with \( \tau_i \)'s WCET as \( c_i \) under \( f_{\max} \). If all execution strategies take the same execution time \( D \) to complete the application and \( k(\geq 0) \) faults have to be tolerated, executing the application under uniform frequency \( f_0 = \frac{\sum_{i=1}^{m} c_i}{D} \) achieves the highest reliability and consumes least energy, where \( (f_i, t_i)_n \) denotes an execution strategy with \( \sum_{i=1}^{n} f_i t_i = \sum_{i=1}^{n} c_i f_{\max} \), \( \sum_{i=1}^{n} t_i = D \), and \( f_{\min} \leq f_i \leq f_{\max} \).

**Proof.** \( R_A((f_i, t_i)_n, k) = r(\sum_{i=1}^{n} \lambda(f_i) t_i, k) \) and \( R_A((f_0, D)_1, k) = r(\lambda(f_0) D, k) \). As \( \lambda(x, k) \) are convex for \( x > 0 \), we have \( \sum_{i=1}^{n} \lambda(f_i) t_i \geq \lambda(f_0) D \) according to formula (14)\(^1\). Furthermore \( r(x, k) \) is a monotonically decreasing function \( x > 0 \)\(^2\), hence we have \( r(\sum_{i=1}^{n} \lambda(f_i) t_i, k) \leq r(\lambda(f_0) D, k) \), i.e., \( R_A((f_i, t_i)_n, k) \leq R_A((f_0, D)_1, k) \). Similarly, according to formula (13), \( E_A((f_i, t_i)_n, k) = \sum_{i=1}^{n} \delta(f_i) t_i \) and \( E_A((f_0, D)_1, k) = \delta(f_0) D \). As \( \delta(x) \) is a convex function for \( x > 0 \), then \( \sum_{i=1}^{n} \delta(f_i) t_i \geq \delta(f_0) D \), i.e., \( E_A((f_i, t_i)_n, k) \geq E_A((f_0, D)_1, k) \). \( \square \)

\(^1\)Given a convex function \( g(x) \), for \( n \in \mathbb{N}^+ \) and \( x_i, t_i \in \mathbb{R}^+ \), we have

\[
\sum_{i=1}^{n} g(x_i) t_i \geq g\left( \frac{\sum_{i=1}^{n} x_i t_i}{\sum_{i=1}^{n} t_i} \right) \left( \sum_{i=1}^{n} t_i \right)
\]

which can be directly derived from convex function definition.

\(^2\)The proof will be given in appendix.
The application’s optimal executing strategy is to use frequency $f_d$ where $f$ we have $k$ execution. Therefore, to guarantee the task re-execution based recovery requires that the whole task is re-executed if a fault is detected at the end of its execution interval; likewise, if $f_r > f_d$, the reliability constraint is the dominant one. In this case, if there exist unused slack time before the deadline, we can utilize it to further reduce the energy consumption without violating the reliability constraint. In the following sections, we will discuss how to use fault recovery to further reduce energy consumption while meeting application’s reliability and deadline constraints.

**3.2 Task Re-execution Based Recovery for the C-RDE Problem**

The task re-execution based recovery requires that the whole task is re-executed if a fault is detected at the end of its execution. Therefore, to guarantee $k$ faults are tolerated, the longest $k$ tasks’ WCETs have to be reserved for recovery.
where $C$ under the frequency $f$ with small error margin.

The coarse-grain search first finds the interval where the optimal processing frequency based on (19) that meets both constraints but with minimal energy consumption.

Algorithm 1 searches for the optimal solution for the C-RDE problem. It has two steps: coarse-grain search (line 1 - 8) and fine-grain search (line 11 - 22). The coarse-grain search stage. Based on the uniform frequency properties given in Section 3.1, if the application $A = \{\tau_1, \ldots, \tau_m\}$ is to be executed with a uniform frequency, the minimum processing frequency for meeting the deadline $D$ is:

$$f(k) = \max\{\frac{C}{D - \sum_{i=1}^{m} l_i}, f_{\min}\}$$

(19)

where $C = \sum_{i=1}^{m} c_i$, $l_i$ is the WCET of $i$th longest task in the application. The reliability and energy consumption under the frequency $f(k)$ are

$$R_A^{re}((f(k), \frac{C}{f(k)}), k) = \frac{k}{|\lambda(f(k))|} \cdot e^{-\lambda(f(k))} \cdot \frac{C^{|\lambda(f(k)}|}{\lambda(f(k))} \cdot \frac{e^{-\lambda(f(k))|\sum_{i=1}^{m} rec(h)}}{\lambda(f(k))}$$

(20)

$$E_A^{re}((f(k), \frac{C}{f(k)}), k) = \delta(f(k)) \cdot \frac{C}{f(k)}$$

(21)

From(20) and (21), it is clear that increasing the number of faults to be tolerated reduces processing stage time, hence to meet the deadline constraint, processing frequency needs to be increased. Increasing processing frequency increases the application’s reliability, but at the same time, it also increases energy consumption. For a given reliability constraint $R_g$ and a deadline constraint $D$, we need to find how many faults needed to be tolerated so that we can know the optimal execution frequency based on (19) that meets both constraints but with minimal energy consumption. Algorithm 1 searches for the optimal solution for the C-RDE problem. It has two steps: coarse-grain search (line 1 - 8) and fine-grain search (line 11 - 22). The coarse-grain search first finds the interval where the optimal processing frequency exists, then fine-grain search is used to find the optimal solution in the interval returned by rough-search with small error margin.

In the coarse-grain search stage, we set the granularity as task’s WCET. If the coarse-grain search returns with $k = 0$ at line 8, it means all available time should be used for processing state as an optimal execution strategy.

If the coarse-grain search returns with $k > 0$, it indicates $R_A^{re}((f(k - 1), \frac{C}{f(k - 1)}), k - 1) < R_g$ and $f(k - 1) < f_{\max}$. As the reliability constraint is not satisfied, hence the number of tolerated faults needs to be increased from $k - 1$ to $k$ which will increase the recovery stage time and decrease the processing stage time by the WCET of $k$th longest task. At the end of coarse-grain search, we get $f_{\text{coarse}}$ (assume running the application under $f_{\max}$ can be guaranteed as a valid solution), which is a valid solution, i.e., meets both reliability and deadline constraints, and the optimal processing frequency is within the interval $[f(k - 1), f_{\text{coarse}}].$

In fine-search stage (line 10 - 21), we gradually increase the frequency from $f(k - 1)$ to $f_{\text{coarse}}$ by the granularity of $\varepsilon$ to approach the optimal solution.

The time cost for loop from line 1-7 is $O(n)$, and while loop from line 12-20 is bounded by $\frac{f_{\max}}{\varepsilon}$ which is constant if we set $\varepsilon$ to $x\%f_{\max}$. Hence, the time complexity for the algorithm is $O(n)$.

We use an example to illustrate the procedure of the algorithm.

Example 1. Assume an application consisting of five tasks with WCET as 10, 20, 60, 80, 120 under $f_{\max}$, respectively, the reliability and deadline constraints are $R_g = 99.9999\%$ and $480$, respectively. TABLE 1 gives the steps produced by Algorithm 1. When the number of faults, $k$ is 0 or 1, the reliability constraint is unsatisfied. Then we increase $k$ to

<table>
<thead>
<tr>
<th>$k$</th>
<th>$f(k)$</th>
<th>$R_A^{re}((f(k), \frac{C}{f(k)}), k)$</th>
<th>observation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.6042</td>
<td>92.69%</td>
<td>$R_A^{re} &lt; R_g$ (invalid)</td>
</tr>
<tr>
<td>1</td>
<td>0.8059</td>
<td>99.999%</td>
<td>$R_A^{re} &lt; R_g$ (invalid)</td>
</tr>
<tr>
<td>2</td>
<td>1.0357</td>
<td>$f(k) &gt; f_{\max}$ (invalid)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.8956</td>
<td>99.999909%</td>
<td>solution</td>
</tr>
</tbody>
</table>

Table 1: Example for Algorithm 1

IITCSTR-2012-0408 7
Algorithm 1 TRE-C-RDE \((A = \{\tau_1, \ldots, \tau_m\}, R_g, D)\)

1: for \(k = 0\) to \(m\) do
2:  if \(R_A^k((f(k), \frac{C}{f(k)})_1, k) < R_g\) and \(f(k) < f_{\text{max}}\) then
3:    \(k = k + 1;\)
4:  else
5:    break;
6:  end if
7: end for
8: \(f_{\text{coarse}} = \min\{f(k), f_{\text{max}}\};\)
9: \(f_{\text{opt}} = f_{\text{coarse}};\)
10: if \(k > 0\) then
11:    \(f' = f(k - 1);\)
12:    while \(f' < f_{\text{coarse}}\) do
13:      if \(R_A^k((f', \frac{C}{f'})_1, k - 1) \geq R_g\) then
14:        \(f_{\text{opt}} = f';\)
15:        \(k = k - 1;\)
16:        break;
17:      else
18:        \(f' = f' + \varepsilon;\)
19:      end if
20:  end while
21: end if
22: \(E_{\text{opt}} = E_A^k((f_{\text{opt}}, \frac{C}{f_{\text{opt}}})_1, k);\)
23: return \(E_{\text{opt}}, f_{\text{opt}}, k;\)

2, however, \(f(2) > f_{\text{max}}\), which breaks the loop. The algorithm then enters the fine-grain search stage to find the \(f_{\text{opt}}\), by gradually adjusting the frequency (\(\varepsilon\) is set to 1\%\(f_{\text{max}}\)). The optimal execution strategy is to use \(f_{\text{opt}} = 0.8956\) and reserve the largest \(c_i\) to recovery state tolerate one possible failure.

### 3.3 Checkpointing Based Recovery for the C-RDE Problem

Task re-execution strategy requires reserving the task’s WCET for possible fault recoveries, which may not be effective due to the long fault recovery stage. Intuitively, if we can divide a long task into several smaller ones, then the recovery time can be reduced, hence more time may be used to scale down the frequency for energy saving. This is the motivation for checkpointing based recovery strategy.

Assume the checkpointing overhead \(q\) (including the fault detection time cost) is a constant under \(f_{\text{max}}\). If having \(h\) checkpoints in the processing stage and up to \(k\) faults to be tolerated in the recovery stage, by uniform frequency scaling property, the minimum possible processing frequency \(f(h, k)\) can be expressed as:

\[
  f(h, k) = \max\left\{\frac{C + \frac{qh}{f(h, k)}}{D - \sum_{i=1}^{k} l'_i}, \frac{1}{f_{\text{min}}}\right\}
\]  

(22)

where \(C = \sum_{i=1}^{m} c_i\) and \(l'_i\) is WCET of the \(i\)th longest task in \(A'\) under \(f_{\text{max}}\), which depends on the placement of checkpoints. \(A'\) is the new task set when some of the tasks in the original set \(A\) are divided into smaller ones by inserted checkpoints.

Then the reliability and total energy consumption of the application under \(f(h, k)\) can be expressed as:

\[
  R_A^h((f(h, k), \frac{C + \frac{qh}{f(h, k)} }{f(h, k)})_1, k) = \sum_{i=0}^{k} \left(\lambda(f(h, k)) \frac{C + \frac{qh}{f(h, k)}}{f(h, k)}\right) e^{-\lambda(f(h, k))} \frac{C + \frac{qh}{f(h, k)}}{f(h, k)} \frac{C + \frac{qh}{f(h, k)}}{f(h, k)} \cdot e^{-\lambda(f_{\text{max}}) \sum_{h=1}^{k} \text{rec}(h)}
\]  

(23)
And the total energy consumption can be written as:

$$E^k_{A}(f(h, k), \frac{C + qh}{f(h, k)})_1(k) = \delta(f(h, k)) \frac{C + qh}{f(h, k)}$$  \hspace{1cm} (24)

where \( rec(h) = l'_h \).

For task re-execution based recovery strategy, if the number of faults to be tolerated is fixed, the duration of reserved recovery stage is fixed. However, for checkpointing based approach, different checkpointing strategy may have different checkpointing intervals, hence, the recovery time may be different. Therefore, how the checkpoints are inserted can significantly impacts the solution.

Adding checkpoints may reduce the recovery time and hence more processing time can be used to scale down the frequency to save energy; but reduced frequency also reduces reliability. However, increasing the number of tolerated faults may improve the reliability at the cost of increased energy consumption. In this sense, we can perform actions of taking checkpointing and increasing number of faults to be tolerated iteratively until no more energy can be further saved because of checkpointing overhead.

The checkpointing based approach for solving the C-RDE problem is given in Algorithm 2. The basic idea of the Algorithm 2 is we use task re-execution algorithm 1 to find the optimal execution solution when no checkpointing is taken (line 1). We then incrementally add a checkpoint to the task with longest interval and obtain a new task set \( A' \) (line 10). Based on \( A' \), run algorithm 1 again to check whether it can get a better solution (line 12 -16). Repeat 9 - 16 until reaching the maximum number of checkpoints, which can be calculated as:

$$max_{\text{chk}} = \frac{D - \sum_{i=1}^{m} c_i}{\text{chk\_overhead}}$$  \hspace{1cm} (25)

A data structure \( TC \) is used to record the checkpointing placement on the task set \( A \), \textit{index, wcet, chk, inv} indicate the task index, WCET, number of inserted checkpoints and the length of interval (including checkpointing overhead), respectively. \( TC_{\text{opt}} \) and \( f_{\text{opt}} \) are to record the currently optimal checkpoints placement and the optimal processing frequency.

\begin{algorithm}
  Algorithm 2 CHK-C-RDE (\( A\{\tau_1, \ldots, \tau_m\}, \ R_g, \ D, \ q \))
  \begin{algorithm}
  1: \( (E^0_{\text{opt}}, f^0_{\text{opt}}, k) = \text{TRE-C-RDE (}\ A, \ R_g, \ D) \)
  2: \text{for } i = 1 \text{ to } m \text{ do}
  3: \( TC(i).index = i; TC(i).wcet = c_i; \)
  4: \( TC(i).chk = 0; TC(i).inv = c_i; \)
  5: \text{end for}
  6: \( TC_{\text{opt}} = TC; \)
  7: \( \text{chk\_num} = 0; \)
  8: \text{while } \text{chk\_num} \leq max_{\text{chk}} \text{ do}
  9: \( \text{chk\_num} + +; \)
  10: \( (A', TC) = \text{INSERT-CHK (}\ TC, \ \text{chk\_cost}); \)
  11: \( (E, f, k) = \text{TRE-C-RDE (}\ A', \ R_g, \ D); \)
  12: \text{if } E^0_{\text{opt}} > E \text{ then}
  13: \( E^0_{\text{opt}} = E; \)
  14: \( f_{\text{opt}} = f; \)
  15: \( TC_{\text{opt}} = TC; \)
  16: \text{end if}
  17: \text{end while}
  18: \text{return } TC_{\text{opt}}, f_{\text{opt}}, k;
  \end{algorithm}
\end{algorithm}

The time complexity of the algorithm is dominated by while loop (line 8-17). Inside the loop, amortized time cost of INSERT-CHK (line 10) is \( O(m + max_{\text{chk}}) \) and that of TRE-C-RDE (line 11) is \( O(m) \). The time complexity of the algorithm is \( O(m) \).
4 Optimal Application Execution Strategy under Discrete Frequency Scaling

In this section, we discuss the case when there are only a set of discrete processing frequencies, i.e., \( \{f_1, f_2, \ldots, f_n\} \) with \( f_i < f_j \) if \( i < j \), available for scaling. We call the problem defined in Section 2.2 the D-RDE for short.

4.1 Task Re-execution Based Recovery for the D-RDE Problem

In this section, we discuss the scenario using task re-execution technique for fault recoveries.

**Lemma 5.** Assume \( F = \{f_1, \ldots, f_n\} \) are available frequencies with \( f_i < f_j \) if \( i < j \), if \( f_{\text{opt}}^c \) and \( k^c \) are the application’s optimal execution strategy for the C-RDE problem using task re-execution for fault recovery, then for D-RDE problem using task re-execution for fault recovery, the optimal strategy is

1. if \( \exists f_v \in F \) such that \( f_v = f_{\text{opt}}^c \), then \( f_{\text{opt}}^c \) and \( k^c \) is also the optimal strategy for solving the D-RDE problem.
2. if \( \exists f_v \in F \) such that \( f_v < f_{\text{opt}}^c < f_{v+1} \), then partition the total execution time into two segments \( t_v \) and \( t_{v+1} \), and using \( f_v \) for \( t_v \) and \( f_{v+1} \) for \( t_{v+1} \), is the optimal strategy for solving the D-RDE problem with ability to tolerate \( k^c \) faults.

**Lemma 6.** If \( (\langle f_{\text{opt}}^c, t_1 \rangle, k^c) \) and \( (\langle f_1, t_2 \rangle, k') \) with \( \langle f_1, t_2 \rangle = [(f_v, t_v), (f_{v+1}, t_{v+1})] \) are the optimal strategies for solving the C-RDE and D-RDE problems by using task re-execution for fault recovery, respectively, where \( f_v < f_{\text{opt}}^c < f_{v+1} \), then \( k' \geq k^c \).

The proof of lemma 5 and 6 is given in appendix.

With Lemma 5 and 6, finding optimal strategy for the D-RED problem becomes finding the optimal value of \( t_v, t_{v+1} \) and \( k^c \). As both \( f_v \) and \( f_{v+1} \) are greater than \( f_{\text{ee}} \), from energy saving perspective, the frequency of \( f_v \) should be used to execute the application as long as possible. This is the basic idea behind Algorithm 6. \( A_v \) and \( A_{v+1} \) are the stacks used to record the tasks to be executed under \( f_v \) and \( f_{v+1} \), respectively. By default, all tasks’ execution frequencies are set as \( f_{v+1} \). Then for each possible number of tolerated fault \( k' \), we move the longest task from \( A_{v+1} \) to \( A_v \) as long as reliability and deadline constraints are satisfied.

**Algorithm 4** TRE-D-RDE \((A\{\tau_1, \ldots, \tau_m\}, R_y, D, F)\)

1. \((f_{\text{opt}}^c, k^c) = \) TRE-C-RDE\((A, R_y, D)\);
2. get \( F' = [f_v, f_{v+1}] \) satisfying \( f_v < f_{\text{opt}}^c < f_{v+1} \)
3. \((A_v, A_{v+1}) = \) ASSIGN-FRE\((A, R_y, D, k, F')\)
4. return \( A_v, A_{v+1} \)

Using TRE-C-RDE to get \( f_{\text{opt}}^c \) and \( k^c \), and then apply ASSIGN-FRE to do the frequency assignment, we get the algorithm for TRE-D-RDE problem, which is the algorithm 4.

4.2 Checkpointing Based Recovery for the D-RDE Problem

The checkpointing based algorithm (CHK-D-RDE) is similar to that of task re-execution algorithm, the only difference is the task set is modified when checkpoints are added. Algorithm 5 gives the solution.
5 Evaluation and Discussion

We first introduce two baseline algorithms and one definition.

- LTF (longest-task-first) [10]: Always select the task with longest WCET and allocate as much slack to it as possible.
- SUEF (slack usage efficiency factor) [10]: Always select the task with the largest ratio of the amount of energy saved to the required slack time.

Definition 2. Tasks execution time heterogeneity (TETH)

\[
TETH = \sqrt{\frac{\max\{c_i|1 \leq i \leq n\}}{\min\{c_i|1 \leq i \leq n\}}}
\]

Where \(c_i\) is the WCET of task \(i\).

In the following experiments, we set the average soft error rate \(\lambda_0 = 10^{-6}\), \(m = 3\) and \(P_{ind} = 0.05P_d\), and \(C_{ef} = 1\). The WCET of the tasks are randomly generated between the \(\min\{c_i\}\) and \(\max\{c_i\}\), where \(\min\{c_i\}\) is set to 20. As LTF and SUEF only work when the reliability constraint is set to reliability of executing the application under \(f_{max}\), to be fairness, which is set as the reliability constraint \(R_g\) in our experiments. And we use \(chk\_cost\) to indicate checkpointing cost and \(utilization(\Gamma)\) to represent the utilization of the processor, which can be written as:

\[
utilization(\Gamma) = \frac{\sum_{i=1}^{m} c_i}{D}
\]

The normalized energy cost is the energy cost normalized to the one of executing the application under \(f_{max}\).

5.1 Simulation Results for the C-RDE problem

In this section, we will evaluate the performance of our proposed algorithms for C-RDE problem under different scenarios.

Fig. 1 evaluates the task execution time variation impact. From which, we can see \(TRE-C-RDE\) and \(CHK-C-RDE\) are more sensitive to \(TETH\). The reason is, when \(TETH\) increases, the tasks’ execution time variation increases, so length of recovery stage may grow, which will reduce the available slack time for \(TRE-C-RDE\). While, \(CHK-C-RDE\) can overcome this shortcoming by adding checkpoints to split the long tasks into smaller ones, which behaves even better in larger \(TETH\). Comparing with \(LTF\) and \(SUEF\), our proposed algorithms always performance better.

Fig. 2 gives the utilization impact. \(LTF\) and \(SUEF\) always consume more energy than \(TRE-C-RDE\) and \(CHK-C-RDE\), while the gap becomes smaller when utilization becomes larger. \(CHK-C-RDE\) always performs the best, even under \(utilization(\Gamma) = 0.9\), which can save about 10% normalized energy cost more than the other three.

Fig. 3 evaluates the impact of checkpointing overhead on \(CHK-C-RDE\) algorithm. In this experiment, the WCET of shortest task and longest one are 20 and 500, respectively, and the mean of them is 260. When \(chk\_cost = 2\), \(CHK-C-RDE\) algorithm saves 7% normalized energy more than \(TRE-C-RDE\), by increasing the \(chk\_cost\), the gap reduces and the two lines cross at \(chk\_cost = 20\) eventually.

Fig. 4 shows the impact of \(d\), which is a system constant indicating the sensitivity of soft-error. The normalized energy cost of \(LTF\) and \(SUEF\) remain the same for different \(d\), while that of \(TRE-C-RDE\) and \(CHK-C-RDE\) algorithm increase when \(d\) becomes larger. These are due to that, the reliability constraint is naturally satisfied for \(LTF\) and \(SUEF\), so \(d\) has no impact of their procedures. The advantage of our proposed algorithms are obvious, even under the scenario with \(d = 5\), \(CHK-C-RDE\) algorithm still save 25% normalized energy consumption more than \(LTF\) and \(SUEF\).
Algorithm 6 ASSIGN-FRE \( (A\{\tau_1, \ldots, \tau_m\}, R_g, D, k, F') \)

1: \( \bar{A}_{v+1} = A_{v+1} = \text{push}(A) \) by \( wcet \) in ascending order
2: \( \bar{A}_v = A_v = \Phi \);
3: \textbf{for} \( k' = k \) to \( |A| \) \textbf{do}
4: \hskip 1em \textbf{while} \( |A_{v+1}| > 0 \) \textbf{do}
5: \hskip 2em \textbf{if} Reliability \( \geq R_g \) and Deadline \( \leq D \) \textbf{then}
6: \hskip 3em \( \bar{A}_{v+1} = A_{v+1} \);
7: \hskip 3em \( A_v = A_v \);
8: \hskip 3em \text{index} = \text{pop}(A_{v+1})
9: \hskip 3em \text{push}(A_v, \text{index})
10: \hskip 2em \textbf{else}
11: \hskip 3em \textbf{break};
12: \hskip 2em \textbf{end if}
13: \hskip 1em \textbf{end while}
14: \textbf{if} Reliability \( \geq R_g \) and Deadline \( \leq D \) \textbf{then}
15: \hskip 1em \( \bar{A}_{v+1} = A_{v+1} \);
16: \hskip 1em \( A_v = A_v \);
17: \textbf{end if}
18: \textbf{end for}
19: \textbf{return} \( \bar{A}_v, A_{v+1} \)

Figure 1: Impact of TETH (utilization(\( \Gamma \)) = 0.7, chk\_cost = 2, d = 3)
Figure 2: Impact of utilization ($TETH = 5$, $chk\_cost = 2$, $d = 3$)

Figure 3: Impact of checkpointing cost ($TETH = 5$, $utilization(\Gamma) = 0.7$, $d = 3$)
5.2 Simulation Results for the D-RDE problem

We show the effectiveness of our proposed algorithms for D-RDE problem in this section, the available discrete frequencies are set as \{0.4, 0.6, 0.8, 1.0\}.

Fig. 5 shows that, the performance of TRE-D-RDE (CHK-D-RDE) is closed to that of TRE-C-RDE (CHK-C-RDE) (about 3\% gap), and the utilization variation doesn’t impact too much.

Fig. 6 investigates the impact of tasks execution time variation. When the TETH increases from 1 to 10, the performance gap between CHK-C-RDE and CHK-D-RDE decreases from 6\% to 3\%, and which between TRE-C-RDE and TRE-D-RDE becomes even smaller.

6 Conclusion

In this paper, we have discussed the problem of how to employ DVFS technique to minimize the overall energy consumption for a hard real-time system under given reliability and deadline constraints. In the system model, we assume that fault arrival rate follows Poisson distribution and the two different fault recovery strategies are applied for fault tolerances. Based on the assumptions, we have introduce an optimal execution strategy for solving the problem. For the scenario with processor’s frequency can be continuously scaled down, we have proved that executing the application under uniform frequency is the optimal solution. While for the scenario with only a set of discrete frequencies are available, the optimal strategy is to execute the application under two neighboring frequencies. According to these analysis, task re-execution and checkpointing based algorithms have proposed for both scenarios mentioned above. We further compare the algorithms with other heuristics in literature and the experiment results show that our proposed algorithms outperform by as much as 20\% with respective to energy saving.

The current work is based on uniprocessor. Extending the work to multi-processor environment will be our immediate next step.
Figure 5: Impact of utilization for algorithms of C-RDE and D-RDE ($TETH = 5, chk_{\text{cost}} = 2, d = 3$)

Figure 6: Impact of TETH for algorithms of C-RDE and D-RDE ($utilization(\Gamma) = 0.7, chk_{\text{cost}} = 2, d = 3$)
References


7 Appendix

In this section, we first show that \( r(x, k) (k \geq 0) \) is a decreasing function when \( x > 0 \) and then provide the proof for Lemma 1.

According to the definition of \( r(x, k) \), we have

\[
r(x, k) = \sum_{i=0}^{k} \frac{x^i e^{-x}}{i!} \cdot e^{-\lambda(f_{\text{max}})(\sum_{h=1}^{i} \text{rec}(h))}
\]

If \( k = 0 \), then \( r(x, 0) = e^{-x} \), which is obvious decreasing for \( x > 0 \). For the scenario \( k > 0 \), as the \( h \)th longest fault recovery duration, i.e., \( \text{rec}(h) \), is fixed when the fault recovery strategy is determined, we can treat it as constant.

If we define \( b(i) = e^{-\lambda(f_{\text{max}})(\sum_{h=1}^{i} \text{rec}(h))} \), then we have \( b(i+1) < b(i) \) and \( 0 < b(i) \leq 1 \) when \( i \geq 0 \). Hence, the first derivation of \( r(x, k) \) is \( r'(x, k) = (\sum_{i=0}^{k-1} \frac{x^i (b(i+1) - b(i))}{i!} - \frac{b(k)x^k}{k!}) \cdot e^{-x} \). As \( b(i+1) - b(i) < 0 \) and \( b(k) > 0 \) when \( x > 0 \), we have \( r'(x, k) < 0 \), which implies \( r(x, k) \) is decreasing when \( x > 0 \). These conclude the proof.

Lemma 1: Assume an application’s total execution time is divided into \( t_1, t_2, \cdots, t_n \) segments, and segment \( t_i \) is executed under frequency \( f_i \), \( k \) faults need to be tolerated, all fault recoveries are executed under \( f_{\text{max}} \), and \( \text{rec}(h) \) is the \( h \)th longest fault recovery duration, then the reliability of the application, which is defined as the probability of completing the application successfully, can be represented as:

\[
R_A((f_1, t_1), n, k) = \sum_{i=0}^{k} \left( \sum_{j=1}^{n} \frac{\lambda(f_j) t_{j}^i e^{-\lambda(f_j) t_j}}{i!} \right) \cdot e^{-\lambda(f_{\text{max}})(\sum_{h=1}^{i} \text{rec}(h))}
\]

where \( (f_i, t_i)_n = [(f_1, t_1), (f_2, t_2), \cdots, (f_n, t_n)] \).

Proof: When \( k \) faults happen, in the worst case scenario, \( k \) longest fault recoveries will be taken. Now, we proof the lemma by induction.

Step 1): When \( n = 1 \), then \( (f_i, t_i)_n = [(f_1, t_1)] \), we have

\[
R_A((f_1, t_1), 1, k) = \sum_{i=0}^{k} \left( \sum_{j=1}^{n} \frac{\lambda(f_j) t_{j}^i e^{-\lambda(f_j) t_j}}{i!} \right) \cdot e^{-\lambda(f_{\text{max}})(\sum_{h=1}^{i} \text{rec}(h))}
\]

This is obvious true.

Step 2): Suppose \( n = n_1 (> 1) \), we have

\[
R_A((f_1, t_1), n, k) = \sum_{i=0}^{k} \left( \sum_{j=1}^{n_1} \frac{\lambda(f_j) t_{j}^i e^{-\lambda(f_j) t_j}}{i!} \right) \cdot e^{-\lambda(f_{\text{max}})(\sum_{h=1}^{i} \text{rec}(h))}
\]

Where \( (f_i, t_i)_n = [(f_1, t_1), (f_2, t_2), \cdots, (f_n, t_n)] \).

Step 3): When \( n = n_1 + 1 \), then \( (f_i, t_i)_n = [(f_1, t_1), (f_2, t_2), \cdots, (f_{n_1+1}, t_{n_1+1})] \). If exactly \( i \) faults happen in the processing stage, there must be \( l \) (\( 0 \leq l \leq i \)) of them occur when the processing frequency is \( f \in \{f_1, \cdots, f_{n_1}\} \) and the remaining \( i - l \) faults happen when \( f = f_{n_1+1} \), then the reliability of the application can be written as:

\[
R_A((f_1, t_1), n, k) = \sum_{i=0}^{k} \sum_{l=0}^{i} \left( \sum_{j=1}^{n_1} \frac{\lambda(f_j) t_{j}^i e^{-\lambda(f_j) t_j}}{i!} \right) \cdot e^{-\lambda(f_{\text{max}})(\sum_{h=1}^{i} \text{rec}(h))}
\]

IITCSTR-2012-0408
According to binomial formula, that is \( \sum_{i=0}^{n} \binom{n}{i} \left( \frac{x}{y} \right)^i = \frac{(x+y)^n}{y^n} \), we have:

\[
R_A(\{f_i, t_i\}_n, k) = \sum_{i=0}^{k} \left( \binom{n}{i} \frac{\lambda(f_j) t_j}{i!} e^{-\sum_{j=1}^{n} \frac{\lambda(f_j) t_j}{i!}} \right), \quad e^{-\lambda(f_{\text{max}})(\sum_{k=1}^{n} \text{rec}(h))}
\]

These conclude the proof.

**Lemma 5:** Assume \( F = \{ f_1, \cdots , f_n \} \) are available frequencies with \( f_i < f_j \) if \( i < j \), if \( f_{\text{opt}} \) and \( k^c \) are the application's optimal execution strategy for the C-RDE problem using task re-execution for fault recovery, then for D-RDE problem using task re-execution for fault recovery, the optimal strategy is

1. if \( \exists f_{\text{c}} \in F \) such that \( f_{\text{c}} = f_{\text{opt}} \), then \( f_{\text{opt}} \) and \( k^c \) is also the optimal strategy for solving the D-RDE problem.

2. if \( \exists f_{\text{c}} \in F \) such that \( f_{\text{c}} < f_{\text{opt}} < f_{v+1} \), then partition the total execution time into two segments \( t_v \) and \( t_{v+1} \), and using \( f_{\text{c}} \) for \( t_v \) and \( f_{\text{opt}} \) for \( t_{v+1} \), is the optimal strategy for solving the D-RDE problem with ability to tolerate \( k^c \) faults.

**Proof:** It is trivial for case 1, we only need to give the proof for case 2. Case 2 can be interpreted as if \( \{ (f_v, t_v), (f_{v+1}, t_{v+1}) \} \) is the optimal solution when the available processing frequencies are \( f_{\text{c}} \) and \( f_{\text{opt}} \), then it is also the optimal solution when all the frequencies in \( F \) are available. We need to prove option \( \{ (f_v, t_v), (f_{v+1}, t_{v+1}) \} \) beats any other valid options. As valid options (satisfying reliability and deadline constraint) could be under one uniform frequency, two processing frequencies or multiple processing frequencies, we need to consider all these three scenarios.

\( \{ (f_v, t_v), (f_{v+1}, t_{v+1}) \} \) consumes less energy cost than any other valid options with one uniform processing frequency.

For any valid option with one uniform processing frequency \( f_j \), as \( f_{\text{opt}} > f_j \), in order to meet the reliability and deadline constraints, we have \( f_j \leq f_{v+1} \). As all the \( f_v, f_{v+1} \) and \( f_j \) are higher than \( f_{\text{c}} \), executing the tasks under higher frequency consumes more energy, so the option with uniform frequency \( f_j \) has higher energy cost.

The option \( \{ (f_v, t_v), (f_{v+1}, t_{v+1}) \} \) consumes less energy cost than any other valid option with combinations of two processing frequencies except \( f_{\text{c}} \) and \( f_{\text{opt}} \).

Let option \( \{ (f_a, t_a), (f_b, t_b) \} \) denotes any valid option with combinations of two processing frequencies except \( f_{\text{c}} \) and \( f_{\text{opt}} \), then possible ones should fall into the following two groups: A) \( f_a \geq f_{v+1} \), \( f_b \geq f_{v+1} \), and \( f_a < f_v, f_b < f_{v+1} \). For case A), as the processing frequency is always no lower than \( f_{v+1} \), therefore, it consumes more energy than that of \( \{ (f_v, t_v), (f_{v+1}, t_{v+1}) \} \).

For case B), we have \( R_A(\{ (f_v, t_v), (f_{v+1}, t_{v+1}) \}, k^c) = \delta(f_v) t_v + (f_{v+1} t_{v+1}) + \delta(f_{v+1}) t_{v+1}, \) and \( R_A(\{ (f_a, t_a), (f_b, t_b) \}, k^c) = \delta(f_a) t_a + \delta(f_b) t_b, \)\( R_A(\{ (f_v, t_v), (f_{v+1}, t_{v+1}) \}, k^c) < \delta(f_a) t_a + \delta(f_b) t_b, \) which implies \( E_A(\{ (f_v, t_v), (f_{v+1}, t_{v+1}) \}) < E_A(\{ (f_a, t_a), (f_b, t_b) \}).\)

First we consider the case \( k'' = k \) which involves the following three sub-scenarios.

2.1) \( t_v + t_{v+1} = t_a + t_b \): As \( \delta(f) = P_{\text{ind}} + C_{ef} f^{C_{ef}} (C_{m} \geq 2) \) is increasing and convex for \( f > 0 \), as shown in Fig. 7(b), by setting \( \delta_1 = \delta(f_v) t_v + (f_{v+1} t_{v+1}) + \delta_2 = \delta(f_a) t_a + \delta(f_b) t_b, \) where \( t_a + t_b \), we have \( \delta_1 < \delta_2, \) then \( \delta(f_v) t_v + (f_{v+1} t_{v+1}) < \delta(f_a) t_a + \delta(f_b) t_b, \) which implies \( E_A(\{ (f_v, t_v), (f_{v+1}, t_{v+1}) \}) < E_A(\{ (f_a, t_a), (f_b, t_b) \}).\)

2.2) \( t_v + t_{v+1} > t_a + t_b \): Considering another option \( \{ (f_a, t_a), (f_b, t_b) \} \) with \( t_a' + t_b' = t_v + t_{v+1} \) and \( f_a t_a + f_b t_b = f_a t_a' + f_b t_b' > t_a + t_b, \) which implies option \( \{ (f_a, t_a), (f_b, t_b) \} \) consumes less energy than option \( \{ (f_a, t_a), (f_b, t_b) \}.\) According to 2.1), the energy consumption of \( \{ (f_v, t_v), (f_{v+1}, t_{v+1}) \} \) is less than \( \{ (f_a, t_a), (f_b, t_b) \}, \) hence, it is also less than \( \{ (f_a, t_a), (f_b, t_b) \}.\)

2.3) \( t_v + t_{v+1} < t_a + t_b \): This case is impossible. Suppose it is possible, considering another option \( \{ (f_v, t_v'), (f_{v+1}, t_{v+1}') \} \) with \( t_v' + t_{v+1}' = t_a + t_b \) and \( f_a t_a' + f_b t_b' = f_a t_a + f_b t_b, \) we have \( R_A(\{ (f_v, t_v'), (f_{v+1}, t_{v+1}') \}) = \delta(f_v) t_v' + \delta(f_{v+1}) t_{v+1}', \) which implies \( R_A(\{ (f_a, t_a), (f_b, t_b) \}) > R_A(\{ (f_v, t_v'), (f_{v+1}, t_{v+1}') \}).\) As the option \( \{ (f_v, t_v'), (f_{v+1}, t_{v+1}') \} \) is valid, so \( \{ (f_v, t_v), (f_{v+1}, t_{v+1}) \} \) must be valid.
Consider the energy consumption, as \( t_v' + t_{v+1}' > t_v + t_{v+1} \) and \( f_{v+1} > f_v \geq f_{ee} \), this implies \( E_A(\{(f_v, t_v'), (f_{v+1}, t_{v+1}')\}, k') < E_A(\{(f_v, t_v), (f_{v+1}, t_{v+1})\}, k') \). So the option \( (\{(f_v, t_v'), (f_{v+1}, t_{v+1}')\}, k') \) has less energy consumption than \( (\{(f_v, t_v), (f_{v+1}, t_{v+1}), k'\}, k') \), which contradicts with the assumption that \( (\{(f_v, t_v), (f_{v+1}, t_{v+1}), k'\}, k') \) is the optimal one when processing frequencies are \( f_v \) and \( f_{v+1} \).

When \( k'' \neq k' \), based on the proof in 2.3, we have \( (\{(f_v, t_v'), (f_{v+1}, t_{v+1}')\}, k'') \) with \( f_v t_v' + f_{v+1} t_{v+1}' = f_v t_v + f_{v+1} t_{v+1} \) and \( t_v' + t_{v+1}' = t_v' + t_{v+1}' \) has higher reliability and same deadline with option \( (\{(f_v, t_v), (f_{v+1}, t_{v+1}), k''\}, k'') \). As the option \( (\{(f_v, t_v'), (f_{v+1}, t_{v+1}')\}, k'') \) is valid, then \( (\{(f_v, t_v'), (f_{v+1}, t_{v+1}')\}, k'') \) must be valid. According 2.1), \( (\{(f_v, t_v'), (f_{v+1}, t_{v+1}')\}, k'') \) consumes less energy than \( (\{(f_v, t_v), (f_{v+1}, t_{v+1}), k'\}, k'') \). As \( (\{(f_v, t_v), (f_{v+1}, t_{v+1})\}, k') \) is the optimal one when processing frequencies are set to \( f_v \) and \( f_{v+1} \), hence, which should consume less energy than \( (\{(f_v, t_v'), (f_{v+1}, t_{v+1}')\}, k'') \) as well as \( (\{(f_v, t_v), (f_{v+1}, t_{v+1})\}, k'') \), these conclude the proof for the case \( k'' \neq k' \).

3) The option \( (\{(f_v, t_v), (f_{v+1}, t_{v+1})\}, k') \) consumes less energy cost than any valid options with multiple processing frequencies.

Suppose \( (\{(f_1, t_1), (f_2, t_2), \cdots, (f_n, t_n)\}, k'')(n \geq 3) \) is a valid option, then either A) all the frequencies are no lower than \( f_{v+1} \) or B) some of them are no higher than \( f_v \) and others are no lower than \( f_{v+1} \) holds. For case A), comparing with option \( (\{(f_v, t_v), (f_{v+1}, t_{v+1})\}, k') \), the application will be executed under higher frequencies, hence, consuming more energy. For case B), assume frequencies \( f_1, \cdots, f_j(j \geq 1) \) are no higher than \( f_v \) and \( f_{j+1}, \cdots, f_n \) are no lower than \( f_{v+1} \). Then, according to lemma 2, the option \( (\{(f_1', \sum_{i=1}^j t_i), (f_2', \sum_{i=j+1}^n t_i), k''\}) \) with \( f_1' (\sum_{i=1}^j t_i) = \sum_{i=1}^j f_i t_i, f_2' (\sum_{i=j+1}^n t_i) = \sum_{i=j+1}^n f_i t_i \) has higher reliability and less energy consumption. Based on 2), the option \( (\{(f_v, t_v), (f_{v+1}, t_{v+1})\}, k') \) still beats the option \( (\{(f_1', \sum_{i=1}^j t_i), (f_2', \sum_{i=j+1}^n t_i), k''\}) \), hence, we get the conclusion.

1), 2) and 3) conclude the proof for Lemma 5.

Lemma 6: If \( (\{(f_{opt}, t_1), k_c\}) \) and \( (\{(f_1, t_1), k_c\}) \) with \( \{f_1, t_1\} = \{(f_{opt}, t_1), (f_{v+1}, t_{v+1})\} \) are the optimal strategies for solving the C-RDE and D-RDE problems by using task re-execution for fault recovery, respectively, where \( f_{opt} < f_{opt} < f_{v+1} \), then \( k' \geq k'' \).

Proof: We prove this by contradiction. Suppose \( k' < k'' \), according to lemma 2, then \( (\{(f', t_v + t_{v+1})\}, k') \) with \( f'(t_v + t_{v+1}) = f_v t_v + f_{v+1} t_{v+1} \) is a valid option for C-RDE problem, as \( (\{(f_{opt}, t), k\}) \) is the optimal solution for C-RDE, then we have \( f' > f_{opt} \) and \( t_v + t_{v+1} < t \). As \( (\{(f_{opt}, t), k''\}) \) is valid and \( t_v + t_{v+1} < t \), which means some slack time is unused for option \( (\{(f_v, t_v), (f_{v+1}, t_{v+1})\}, k') \) even if we increase the tolerated faults in recovery stage from \( k' \) to \( k'' \). Increasing the tolerated faults will improve the reliability, so the unused slack time can be further explored to scale down the frequency, that is, there must be a valid option \( (\{(f_v, t_v'), (f_{v+1}, t_{v+1}')\}, k'_opt) \) with \( f'_v + t_{v+1}' > f_v + t_{v+1} \) for D-RDE problem existed, which has less energy consumption than \( (\{(f_v, t_v), (f_{v+1}, t_{v+1})\}, k') \). This contradicts with the assumption that \( (\{(f_v, t_v), (f_{v+1}, t_{v+1})\}, k') \) is the optimal solution for D-RDE when frequencies are set to \( f_v \) and \( f_{v+1} \). These conclude the proof for Lemma 6.