BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **19**, No 4
Sofia • 2019 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2019-0035

An Efficient Fault-Tolerant Scheduling Approach with Energy Minimization for Hard Real-Time Embedded Systems

Barkahoum Kada[1], *Hamoudi Kalla*[2]

[1]*Computer Science Department, University Batna 2, Batna 05000, Algeria*
[2]*Department of Computer Science, University Batna 2, Batna 05000, Algeria*
E-mails: b.kada@univ-batna2.dz Hamoudi.kalla@ univ-batna2.dz

***Abstract*: *In this paper, we focus on two major problems in hard real-time embedded systems fault tolerance and energy minimization. Fault tolerance is achieved via both checkpointing technique and active replication strategy to tolerate multiple transient faults, whereas energy minimization is achieved by adapting Dynamic Voltage Frequency Scaling (DVFS) technique. First, we introduce an original fault-tolerance approach for hard real-time systems on multiprocessor platforms. Based on this approach, we then propose DVFS_FTS algorithm for energy-efficient fault-tolerant scheduling of precedence-constrained applications. DVFS_FTS is based on a list scheduling heuristics, it satisfies real-time constraints and minimizes energy consumption even in the presence of faults by exploring the multiprocessor architecture. Simulation results reveal that the proposed algorithm can save a significant amount of energy while preserving the required fault-tolerance of the system and outperforms other related approaches in energy savings.*

***Keywords*: *Fault tolerance, Transient faults, Checkpointing, Active replication, Dynamic Voltage Frequency Scaling (DVFS), Energy minimization.*

1. Introduction

Energy consumption and fault tolerance have attracted a lot of interest in the design of modern embedded real-time systems. Fault tolerance is fundamental for these systems to satisfy their real-time constraints even in the presence of faults. Transient faults are most common, and their number is dramatically increasing due to the high complexity, smaller transistors sizes, higher operational frequency, and lowering voltages [1-5].

Dynamic power/energy management is an active area of research and many techniques have been proposed to minimize energy consumption under a large diversity of system and task models [6, 7]. Dynamic Voltage and Frequency Scaling (DVFS) is an energy saving technology enabled on most current processors. It enables a processor to operate at multiple voltages where each corresponds to a specific frequency. Because the energy consumption of a processor is proportional to voltage squared, the processor's energy consumption can be considerably reduced by lowering CPU voltage and processing speed [8].

Addressing energy and fault-tolerance simultaneously is a challenge because lowering the voltage to reduce energy consumption has been shown to increase the number of transient faults [4, 11, 20]. Furthermore, reducing working frequency increases task execution time, which can lead to no guarantee of task deadlines.

This paper presents first a novel fault-tolerance approach to tolerate a fixed number of transient faults. Our approach combines active replication which provides space-redundancy and checkpointing with rollback recovery which provides time-based redundancy. Based on this approach and DVFS technique, we propose a fault-tolerant DVFS scheduling heuristic, which generates, from a given hard real-time application and a given multiprocessor architecture, a task allocation scheme that minimizes energy consumption and tolerates *k* arbitrary transient faults.

The rest of the paper is organized as follows. An overview of related work is provided in Section 2. The system models considered in this work are introduced in Section 3. The proposed fault-tolerance approach is explained in Section 4. The strategy that utilizes this approach and DVFS technique to minimize energy is provided in Section 5. The proposed DVFS_FTS algorithm is presented in Section 6. Simulation results are discussed in Section 7, and finally, the conclusion is given in Section 8.

2. Related works

Several papers have been published that are closely related to our research, these researches differ in many aspects, such as task models (dependent or independent tasks, hard or soft deadlines, periodic or aperiodic tasks), multiprocessor or uniprocessor platforms, online or offline scheduling and the fault-tolerance technique adopted.

Authors in [9] proposed a scheduling heuristic to minimize the schedule length, the global system failure rate and the power consumption of the generated schedule. Active replication of tasks and data dependencies is used to increase the system reliability and dynamic voltage scaling DVS is used for energy minimization. The primary-backup (passive replication) approach is used by S a m a l, M a l l and T r i p a t h y [10] as a fault-tolerant scheduling technique to guarantee real-time task constraints in the presence of permanent or transient fault. Authors proposed a scheduling algorithm using a hybrid genetic algorithm. G a n et al. [11] proposed a synthesis approach to decide the mapping of hard real-time applications on distributed heterogeneous systems, such that multiple transient faults are tolerated, and the energy consumed is minimized. For recovery from faults, they used replication technique.

The replication technique is effective to tolerate multiple spatial faults (permanent or transient) and it is more preferable for safety-critical systems.

However, scheduling multiple replicas of each task on different processors may not be affordable due to cost constraints.

Checkpointing with rollback recovery [7, 12-15] and re-execution [16] are classified by Motaghi and Zarandi [17] as time based-redundancy methods. These methods try to deal with transient faults by serial executions on the same processor of the faulty task.

Djosic and Jevtic [1] developed a fault-tolerant DVFS algorithm for real-time application of independent tasks. This algorithm combines DVFS for optimizing energy consumption and re-execution recovery for fault tolerance, but their scope is restricted to single processor systems. In [18], authors introduced an efficient method to determine the checkpointing scheme that can tolerate *k* transient faults on a single processor. They also proposed a task allocation scheme to reduce energy consumption.

The combination of replication and time-based redundancy techniques to tolerate multiple transient faults with low overhead in terms of energy consumption and total execution time has been studied in few works related to our research [19, 20].

Authors in [19] have proposed a fault-tolerance policy assignment strategy to decide which fault-tolerance technique, for instance checkpointing, active replication or their combination, is the best suited for a particular process in the application but energy consumption is not studied in their proposition. Tavana et al. [20] have proposed a standby-sparing scheme which addressed simultaneously reliability and energy consumption. The proposed scheme by employing both hardware redundancy (standby-sparing) and time redundancy (re-execution) in some cases, can tolerate many transient faults. To reduce energy consumption, they applied two techniques DPM (Dynamic Power Management) used by the spare unit and DVS (Dynamic Voltage Scaling) used by the primary processor.

This paper attempts to solve the following problem "Given a set Γ of hard real-time dependent tasks and a set $\mathcal{P}$ of homogeneous processors which support L frequency levels, find the scheduling for all tasks in Γ such that the total energy consumption is reduced without any deadline miss while ensuring fault-tolerance requirement".

The main contributions of this paper are summarized as follows:

- Tolerating multiple transient fault occurrences with respect to application time-constraints.
- Combine two different policies: checkpointing and active replication to propose an efficient fault-tolerance approach that explores hardware resources and timing constraints.
- Extend the proposed fault-tolerance approach to incorporate it with DVFS to achieve more energy saving.
- Efficient fault-tolerant scheduling heuristic DVFS_FTS of precedence-constrained applications based on the earliest-deadline-first (EDF) algorithm and the proposed fault-tolerance approach is presented to minimize the system energy consumption while tolerating *k* transient faults.

3. System models

3.1. Application model

The real-time application considered in this paper consists of n hard aperiodic dependent tasks, denoted as $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Tasks are non-preemptive and cannot be interrupted by other tasks. Tasks send their output values in messages, when terminated. All required inputs have to arrive before activation of the task. The dependence $\tau_i \rightarrow \tau_j$ means that τ_i execution precedes τ_j execution. So we say that τ_j is a successor of τ_i and symmetrically that τ_i is a predecessor of τ_j. Each task τ_i is characterised by a tuple (C_i, D_i), where C_i is the worst case execution time of the task at the maximum frequency/voltage in a fault free condition and D_i is the deadline of the task. The utilization of task τ_i is

(1) $$U_i = \frac{C_i}{D_i}, \quad \text{where } 0 \le U_i \le 1.$$

The system utilization is therefore calculated according to next equation:

(2) $$U = \sum_{i=1}^{n} U_i.$$

We model an application A as a Directed Acyclic Graph (DAG). Each node represents one task. An edge e_{ij} indicates data-dependency between two tasks τ_i and τ_j.

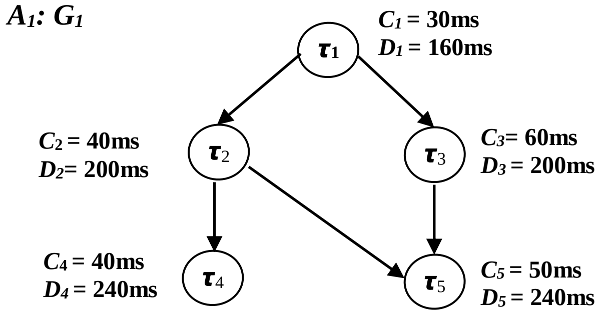

Fig. 1. Hard real time application example

An example of an application A_1 composed of five dependant tasks $\{\tau_1, \tau_2, \ldots, \tau_5\}$ is represented as a DAG G_1 shown in Fig. 1.

3.2. Hardware model

The architecture is considered as a set of m homogeneous processors denoted as: $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$. Each processor is connected with the others through communication links. As so, our architecture is homogeneous and fully connected.

3.3. Fault model

During the execution of an application, faults may be hard to avoid due to different reasons, such as hardware failure, software errors, devices exposed to intense temperatures, and external impacts [22]. As a result, transient faults are more frequent than permanent ones. Hence, authors in this paper are interested in tolerating transient faults as the number of these faults has been dramatically higher.

3.4. Energy model

We assume that there are m processors, each of them is DVFS enabled with a set of L operating frequencies. We denote with $F = \{f_1, f_2, \dots, f_L\}$ with $0 \le f_L \le f_{L-1} \le \dots \le f_1 = f_{\max}$. We assume the frequency values are normalized with respect to $f_{\max}$, i.e., $f_{\max} = 1$.

The energy model used in this work is the same to the one, used in the literature [1, 6, 9, 22], where the power consumption P of a system is given by

(3) $$P_s + h(P_{\text{ind}} + C_{\text{ef}} V^2 f), \quad P = P_s + h(P_{\text{ind}} + P_d),$$

where P_s is the static power, P_{ind} is the frequency-independent power and P_d is the frequency-dependent power. The parameter $h = 1$ when the system is in the working state. Otherwise, when the system is in the standby state, $h = 0$. C_{ef} is the effective loading capacitance, f is the operating frequency and V is the supply voltage. The static power can be removed only by turning off the whole system, P_{ind} is a constant independent of operating frequency. As the energy consumption due to frequency scaling is independent of P_s, we take into account only the frequency-dependent power P_d and we set $P_s = 0$. Hence, the power consumption P can be written as:

(4) $$P = C_{\text{ef}} V^2 f.$$

Since $f \propto V$, and according to (4), the dynamic power P can be expressed as a polynomial of frequency of degree α, where α has been set to 3 in most of the published papers on energy consumption [22, 23]. Hence, we reformulate P in (5) as

(5) $$P = C_{\text{ef}} f^3.$$

The energy consumed by task τ_i is

(6) $$E_i(f_i) = C_{\text{ef}}\, C_i\, f_i^2,$$

where C_i *is* the execution time of task τ_i under frequency f_i. The total energy E_{total} consumed by processors during the execution of a task set is

(7) $$E_{\text{total}} = \sum_{i=1}^{n} E_i(f_i).$$

In this study, we consider only processor energy consumption.

4. The proposed fault-tolerance approach

We propose a mixed fault-tolerance approach, which combines software replication and time-based redundancy for tolerating k transient faults. We use these two techniques in order to meet time constraints and to increase the reliability of hard real-time applications even in the presence of faults.

As time-based redundancy, we use uniform checkpointing with rollbacks. Once a fault is detected, the application rolls back to the last saved checkpoint and re-executes the faulty interval [19]. Inserting one checkpoint to task τ_i refers to save its current state in memory for recovery. As software replication, we use active replication in case that checkpointing with rollbacks cannot satisfy task deadline.

4.1. Uniform Checkpointing with Rollback Recovery

The time overhead for re-execution can be reduced with more complex techniques such as rollback recovery with checkpointing [15]. By using this technique, once a fault is detected during the execution of the task τ_i, it needs to restore the saved state to continue task execution. We consider the following assumptions:

- The checkpointing is uniform (checkpoint intervals are equal for the same task).
- Faults are detected as soon as they occur.
- The checkpoint saving and rollback recovery are themselves fault-tolerant.

The fault-free execution time of task τ_i using uniform checkpointing is a function of the number of checkpoints m_i and is formulated as

(8) $$\mathsf{C}_i(m_i) = \mathsf{C}_i + m_i O_i,$$

where O_i is the time overhead for saving one checkpoint.

The recovery time of τ_i with m_i checkpoints under a single failure is formulated as

(9) $$R_i(m_i) = r_i + \frac{\mathsf{C}_i}{m_i},$$

where r_i is the time overhead to rollback to the latest checkpoint.

In general, in the presence of *k* faults, the Worst-Case Response Time WCRT$_i$ of task τ_i using uniform checkpointing with rollback recovery is given by:

(10) $$\text{WCRT}_i(m_i) = \mathsf{C}_i(m_i) + kR_i(m_i).$$

P o p et al. [19] showed that the optimal number of checkpoints m_i^* to minimize the worst case response time WCRT$_i$ considering *k* faults can be calculated as:

(11) $$m_i^* = \begin{cases} m_i^- = \left\lfloor \sqrt{\frac{k\mathsf{C}_i}{O_i}} \right\rfloor & \text{if } \mathsf{C}_i \leq m_i^-(m_i^- + 1)\frac{O_i}{k}, \\ m_i^+ = \left\lceil \sqrt{\frac{k\mathsf{C}_i}{O_i}} \right\rceil & \text{if } \mathsf{C}_i > m_i^-(m_i^- + 1)\frac{O_i}{k}, \end{cases}$$

where O_i is the time overhead for saving one checkpoint and C_i is the worst case execution time of task τ_i. As the number of checkpoints is an integer, thus we use m_i^- (the floor) or m_i^+(the ceiling) as a value. If $\mathsf{C}_i \leq m_i^-(m_i^- + 1)\frac{O_i}{k}$, we use the floor value. Otherwise, the ceiling value is used.

For the sake of easy presentation, m_i^* is simply denoted by (12)

(12) $$m_i^* = \left\| \sqrt{\frac{k\mathsf{C}_i}{O_i}} \right\|.$$

An example of uniform checkpointing with rollback recovery is presented in Fig. 2.

We consider task τ_1 with worst execution time $\mathsf{C}_1 =$ 60 ms in Fig. 2a. In Fig. 2b two equidistant checkpoints are inserted. Thus, task τ_1 is composed of two execution intervals $\tau_{1(1)}$ and $\tau_{1(2)}$. In Fig. 2c, a fault affects the second execution interval $\tau_{1(2)}$. This faulty interval is re-executed again starting from the second checkpoint.

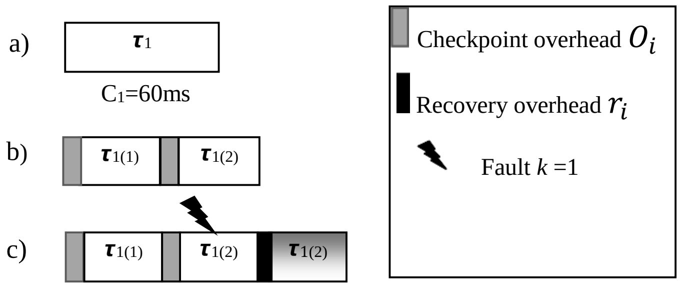

Fig. 2. Uniform checkpointing with rollback recovery

4.2. Collaborative active replication

Uniform checkpointing with rollback recovery technique cannot explore the available processors in the architecture to reduce the schedule length [24]. If the task experiences a fault, then it has to recover on the same processor. In contrast, software replication techniques (active and passive replication) can utilize the spare capacity of the other processors. Software replication has the ability to execute task replicas simultaneously on different processors. With active replication, all the task replicas are executed independent of fault occurrences [25]. However, with passive replication, backup replicas are executed only if the primary replica is faulty [29, 30].

In our work, we are interested in active replication. If there is enough time to rollback to the last saved checkpoint in the presence of faults, we use active replication to guarantee and respect task τ_i deadline. The task τ_i is replicated on two collaborative replicas; τ_i^1 and τ_i^2, both of which are be executed on different processors at the same time. We also introduce collaboration between replicas to tolerate multiple faults and respect task τ_i deadline.

For the sake of uniformity and clarity, we will consider the original task τ_i as the primary replica τ_i^1 and its replica as the backup replica τ_i^2. We consider the following assumptions:

- All checkpoints are assumed to be fault-free, i.e., no faults can occur during checkpoint saving.
- Each task's primary copy and backup copy must not be assigned to the same processor.
- Each task's primary copy and backup copy cannot be faulty at the same time.
- Faults are detected as soon as they occur, and the recovery will be with the no faulty replica.

Our goal is to tolerate k faults with respect to task τ_i deadline. To achieve this goal, we use active replication technique. However, it is possible that both primary and backup replicas are faulty due to multiple fault occurrence. Therefore, our goal will be missed, and active replication alone will be infeasible. This is the case in the work presented in [17].

As a solution, we introduce collaboration between replicas to tolerate each coming fault in the primary or the backup replicas (τ_i^1, τ_i^2) to achieve the feasibility of our approach. For computation purpose, we add an extra virtual processor to the architecture, noted *P#*.

Once the active replication approach is decided for a task τ_i, we execute the following steps:

Step1. τ_i has to be scheduled on virtual processor *P#* (**τ**$_i$#) at Start Time ST_i as illustrated in Fig. 3a;

Step2. We insert in (**τ**$_i$#) the appropriate m_i^* checkpoints obtained with (12);

Step3. τ_i is replicated, which will result in two replicas τ_i^1 and τ_i^2 which must be scheduled on two different processors;

Step4. The initial checkpoints of the task **τ**$_i$# are projected onto τ_i^1 and τ_i^2 alternatively, as illustrated in Fig. 3b.

The alternative checkpointing idea of the two replicas τ_i^1 and τ_i^2 is to ensure the collaboration between replicas and to minimize the number of checkpoints of the original task τ_i. In this case, we can meet the task deadline even in the presence of faults.

In Fig. 3b, Δ represents the difference between the start times of the two replicas τ_i^1 and τ_i^2 (the start time of each replica depends on the availability of processors). It can be written as

(13) $$\Delta = \mathrm{ST}(\tau_i^1) - \mathrm{ST}(\tau_i^2).$$

To ensure the success of our alternative checkpointing idea, Δ should be less or equal than the checkpointing interval, so we have

(14) $$0 \le \Delta \le \frac{C_i}{m_i}.$$

With this approach the start time ST$_i$ of a task τ_i can be given by

(15) $$\mathrm{ST}_i = \min_{1 \le j \le 2} \left(\mathrm{ST}(\tau_i^j) \right),$$

where $\mathrm{ST}(\tau_i^j)$ is the start time of the replica τ_i^j.

Consequently, the actual Finish Time FT_i of task τ_i^j is given by

(16) $$\mathrm{FT}_i = \mathrm{ST}_i + \mathrm{WCRT}_i.$$

In case of fault occurrence in the execution of one of the replicas (τ_i^1 or τ_i^2), the results produced by the no faulty replica must be sent to the faulty replica at checkpoint with Send/Receive communication to continue the execution. As shown in Fig. 3c, when fault affects the first execution interval $\tau_i^1(1)$, the no faulty replica τ_i^2 sends at checkpoint the correct state to the faulty task via communication step.

With alternative checkpointing scheme, the number of checkpoints in each replica is equal to $\left\lfloor \frac{m_i}{2} \right\rfloor$ or $\left\lceil \frac{m_i}{2} \right\rceil$. Hence, in the fault free condition ($k = 0$), the worst-case response time WCRT_i of the task τ_i is given by the term ($C_i + \left\lfloor \frac{m_i}{2} \right\rfloor O_i$). Where O_i is the time overhead for saving one checkpoint.

In case of fault occurrence, the recovery from fault is provided with communication step between the replicas τ_i^1 and τ_i^2. We denote the communication overhead by $\mathrm{com}(\tau_i^1, \tau_i^2)$.

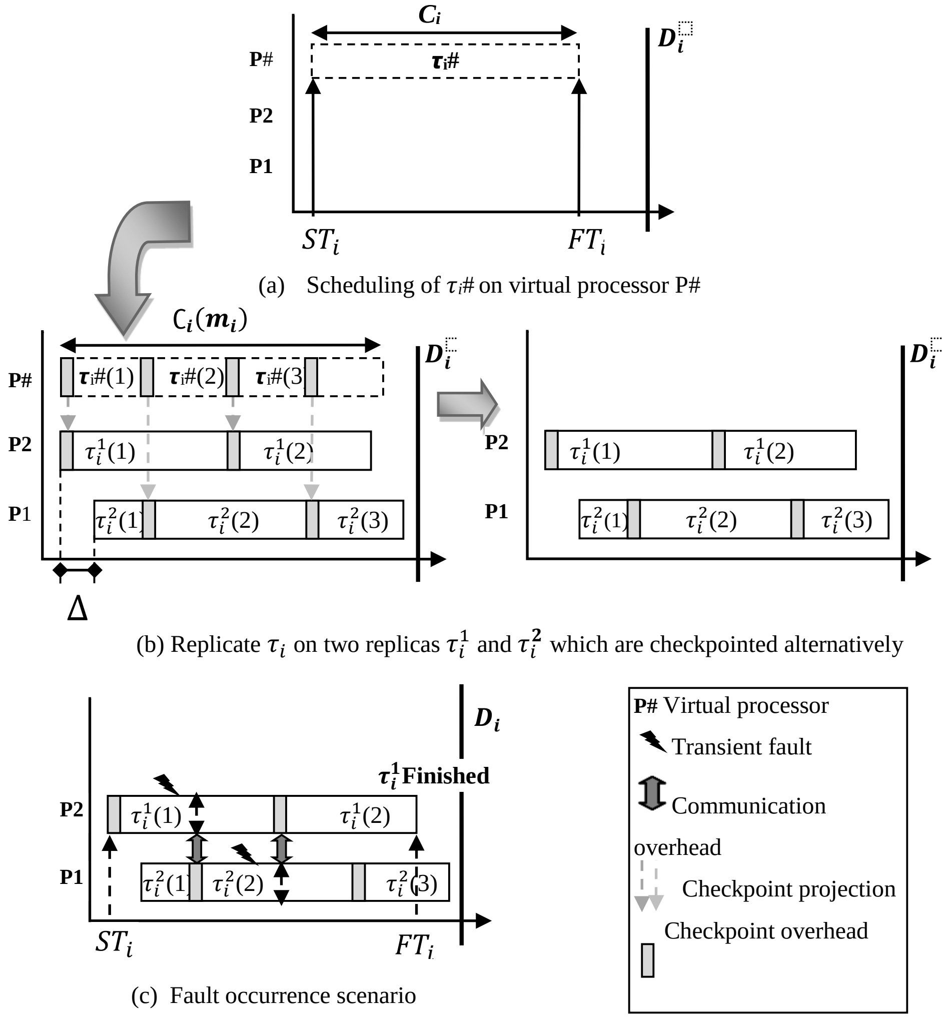

Fig. 3. Illustration of different steps of collaborative active replication

In general, in the presence of k faults, the worst case response time WCRT_i of the task τ_i using the new active replication with checkpointing is formulated in next equation:

$$\mathrm{WCRT}_i = \begin{cases} \mathrm{C}_i + \left\lfloor \frac{m_i}{2} \right\rfloor O_i & \text{if } \ k = 0, \\ \mathrm{C}_i + \left\lceil \frac{m_i}{2} \right\rceil O_i + k\mathrm{com}\left(\tau_i^1, \tau_i^2\right) & \text{, otherwise, or} \\ \mathrm{C}_i + \left\lfloor \frac{m_i}{2} \right\rfloor O_i + k\mathrm{com}\left(\tau_i^1, \tau_i^2\right). \end{cases} \tag{17}$$

The best response time $\mathrm{WCRT}_{\mathrm{best}}$ of the task τ_i can be written as:

$$\mathrm{WCRT}_{\mathrm{best}}(\tau_i) = \min_{1 \le j \le 2} \mathrm{WCRT}\left(\tau_i^j\right), \tag{18}$$

where $\mathrm{WCRT}\left(\tau_i^j\right)$ is the worst case response time of the replica τ_i^j and is calculated with (17).

5. DVFS Based Fault-Tolerance Approach

The DVFS technique can assign different frequencies to each task, which gives us a useful way to minimize energy consumption of applications [26]. We extend the proposed fault-tolerance approach to incorporate it with DVFS to exploit the released slack time to achieve more energy saving.

According to the proposed fault-tolerance approach, we adopt active replication to meet timing constraints and provide high reliability even when deadlines are tight. However, task replicas must be performed at the maximum frequency given the probability of failure is low. We assume that DVFS is used during uniform checkpointing with rollback technique.

Similar to [18], we assume that checkpointing is not affected by processor frequency. We focus on the fault-free execution and like [2] and [27], we aim to reduce the fault-free energy consumption because recovery executions have a small probability of being performed, and for this reason their energy consumption is a negligible fraction of the total energy consumption. The recovery time of a faulty task is always performed at the maximum frequency to preserve its original reliability.

5.1. Optimal frequency assignments

In this section, we search the optimal frequency assignments assuming all tasks their deadlines. In the existence of precedence constraints, a task may have to complete well before its deadline to ensure that all its successor tasks can finish in time. Therefore, as in [21], we can define the effective deadline of a task τ_i as follows:

$$D_i^{\text{ef}} = \begin{cases} D_i\,, & \text{succ}(\tau_i) = \emptyset, \\ \min\left(D_i, D_j^{\text{ef}} - \text{C}_j\right), & \tau_j \in \text{succ}(\tau_i), \end{cases} \tag{19}$$

where $\text{succ}(\tau_i)$ is the set of successor tasks of τ_i.

The frequency f_i^{opt} that allows task τ_i to successfully complete execution before its deadline D_i^{ef} while minimizing energy consumption and tolerating *k* faults with checkpointing with rollback should satisfy the following:

$$\text{ST}_i + \frac{\text{C}_i(m_i)}{f_i^{\text{opt}}} + kR_i(m_i) \le D_i^{\text{ef}}, \tag{20}$$

where ST_i and $\frac{\text{C}_i(m_i)}{f_i^{\text{opt}}}$ are respectively the start time and the fault-free execution time of task τ_i with m_i checkpoints performed at frequency f_i^{opt}. $R_i(m_i)$ is the recovery time of τ_i under a single failure performed at the maximum frequency $f_{\max}$ ($\text{C}_i(m_i)$ and $R(m_i)$ were defined with (7) and (8) respectively).

After evaluation of (20), we obtain the following solution:

$$f_i^{\text{opt}} \ge \frac{\text{C}_i(m_i)}{D_i^{\text{ef}} - \text{ST}_i - kR_i(m_i)} \tag{21}$$

If $f_i^{\text{opt}} \nexists F$, we choose neighboring frequencies $f_L < f_i^{\text{opt}} < f_{L-1}$ and $f_{L-1}, f_L \in F$. Hence, the minimized energy consumed during the execution of task τ_i is given by:

$$E_i\left(f_i^{\text{opt}}\right) = \text{C}_{\text{ef}} \frac{\text{C}_i(m_i)}{f_i^{\text{opt}}} {f_i^{\text{opt}}}^2 = \text{C}_{\text{ef}} \text{C}_i(m_i)\, f_i^{\text{opt}} = \text{C}_{\text{ef}} \frac{\text{C}_i(m_i)^2}{D_i^{\text{ef}} - \text{ST}_i - kR_i(m_i)}. \tag{22}$$

6. The proposed DVFS fault-tolerant scheduling algorithm

Our DVFS fault-tolerant schedule is presented in DVFS_FTS Algorithm. The algorithm takes as input the application *A*, the number *k* of transient faults that have to be tolerated, the architecture $\mathcal{P}$, the set of frequency levels F and the real-time constraints.

```
    DVFS_FTS Algorithm
    Inputs:
              $\Gamma = \{\tau_1, \tau_2, \dots \tau_n\}$
              $\mathcal{P} = \{P_1, P_2, \dots, P_m\}$
              $F = \{f_1, f_2, \dots, f_L\}$
    k transient faults for each task
    Real time constraints
1.  TReady = $\{\tau_i \in \Gamma \mid \mathrm{pred}(t_i) = \emptyset\}$
2.  Schedulable = True
3.  $E_{\mathrm{total}} = 0$
4.  While TReady $\neq \emptyset$ do
5.  {  Select $\tau_i \in$ TReady having the minimum deadline $D_i$
6.  compute $\mathrm{WCRT}_i$ with (10) under maximum frequency
7.  compute the start time $\mathrm{ST}_{ij}$ of $\tau_i$ on all processor $P_j$ in $\mathcal{P}$
8.  $\mathrm{ST}_i = \min_{j=1..m} \mathrm{ST}_{ij}$
9.  If $D_i - \mathrm{ST}_i \geq \mathrm{WCRT}_i$ then
10. { Schedule $\tau_i$ on $P_j$ at the earliest start time    /* $P_j$ is the processor with min
    $\mathrm{ST}_i$*/
11. Apply checkpointing for $\tau_i$
12. compute $f_i^{\mathrm{opt}}$ based on (21)
13. Perform $\tau_i$ under $f_i^{\mathrm{opt}}$ frequency }
14. Else
15. { compute $\mathrm{WCRT}_i$ with (17) under maximum frequency
16. If $D_i - \mathrm{ST}_i \geq \mathrm{WCRT}_i$ then
17. { Schedule both $\tau_i$ on $P_j$ and its replica on another processor $P_k$ at the earliest
    start time.
18. Apply collaborative active replication for $\tau_i$ }
19. Else
20. { Schedulable = False
21. break  }}
22. compute the energy consumption $E_i(f_i)$
23. $E_{\mathrm{total}} = E_{\mathrm{total}} + E_i(f_i)$
24. TReady = Tready – $\{\tau_i\} \cup \{\tau_j \in \mathrm{succ}(\tau_i) \mid \mathrm{pred}(\tau_j) \nexists \mathrm{TReady}\}$
25. } End DVFS_FTS
```

Fig. 4. The proposed DVFS_FTS Algorithm

Our scheduling algorithm is a list scheduling based heuristics, which uses the concept of ready task and ready list. By ready task τ_i, we mean that all τ_i's predecessors have been scheduled. The heuristic initializes the list TReady with tasks without predecessors in line 1 and is looping while TReady isn't empty (line 4-25). At first, the ready task τ_i with minimum deadline is selected for placement in the schedule (line 5). Then, the maximum response time of the task τ_i will be calculated with (10) under maximum frequency (line 6). The checkpointing with rollback policy will be applied if the task deadline can be satisfied on the processor P_j at the earliest start time (line 10-13). In this case, the task τ_i will be performed under the frequency f_i^{opt} calculated based on (21) (line 12-13). Otherwise, the task τ_i will be replicated and the proposed new active replication will be applied. In this case, the maximum response time of the task τ_i will be calculated with (17) under the maximum frequency (line 14-18). After execution of the task τ_i, its energy consumption will be calculated and the total energy will be updated in lines 22-23. Finally, the task τ_i will be removed from the ready list TReady and all its successors are added to the list in line 24.

7. Performance evaluation

In this section, we evaluate the performance of the proposed DVFS_FTS algorithm. For comparison, we have implemented our algorithm and the following schemes:

EXH_FTS: Fault tolerant scheduling algorithm with energy minimization using exhaustion method.

DVFS_CH: Fault tolerant scheduling algorithm that uses checkpointing with roll back technique for fault tolerance and DVS for reduce energy. This algorithm is extended from JFTT scheme [15] for tasks with precedence constraints (application DAG).

The performance is measured in term of normalized total energy saving. We formulate the parameter energy saving ES:

(23) $$\mathrm{ES} = 100 \times \frac{E_{\mathrm{FTS}} - E}{E_{\mathrm{FTS}}},$$

where E_{FTS} is the energy consumption of the proposed algorithm with all tasks are executed at the highest frequency and E is the energy consumption of an algorithm being compared with DVFS scheme.

7.1. Simulation parameters

Before presenting our experimental results, we present the simulation parameters as follows: The method of generating random graphs is the same as [28]. We have generated a set of DAG applications with 10, 20, 30, 40 and 50 tasks. Within a task set, the worst-case execution time on maximum operating frequency C_i for each task is randomly generated with values uniformly distributed in the range of [10 ms, 100 ms]. We assume $\mathrm{C_{ef}} = 1$ and the operating frequencies are set as $F = \{0.1, 0.2, \ldots, 1\}$. The parameters and the values used in our simulation are summarized in Table 1.

Table 1. Parameters for simulation

Parameter	Value(fixed-varied)
Number of processors	4
Application size (Number of tasks)	(10, 20, 30, 40, 50)
Execution time (ms)	[10 , 100]
Normalized frequency	[0.1 – 1] with a step of 0.1
Checkpoint overhead *O*	(1%, 2%, 5%, 10%, 15%, 20%)
Number of faults *k*	(1, 2, 3, 4, 5)

7.2. Experiment results

The first set of experiments compares the energy savings of algorithms with respect to number of transient faults (Fig. 5). In this experiments, we set application size $\Gamma = 10$ tasks, the checkpoint overhead *O* = 2% and vary *k* from 1 up to 5. As can be seen clearly from the figure that the performance on energy saving of DVFS_FTS algorithm outperforms both DVFS_CH and EXH_FTS schemes. For instance, when the number of transient faults is 5 faults, the ES of DVFS_FTS is greater than DVFS_CH and EXH_FTS by 7.17% and 6.34% respectively. Furthermore, we can observe that the energy savings of the three algorithms decreases with the increase of the number of transient faults.

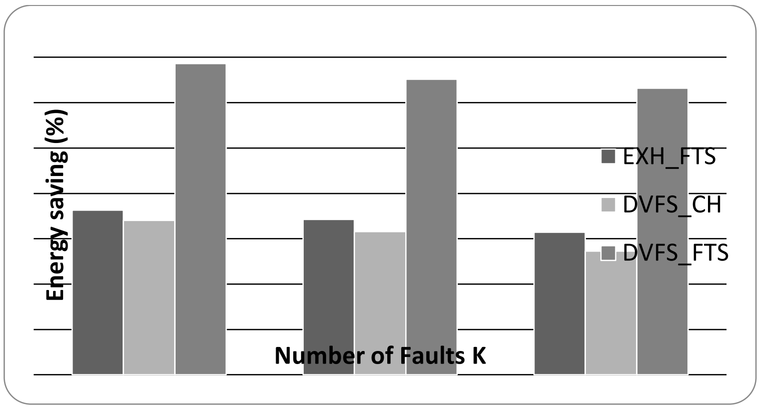

Fig. 5. The impact of number of faults on energy saving

The second set of experiments is to investigate the performance of the different approaches with respect to application size (Fig. 6). In this set of experiments, we set the checkpoint overhead *O* = 2% and *k* =3 and vary the application size Γ from 10 tasks to 50 tasks. We can see that the energy saving increases when the number of tasks increases. The energy saving of DVFS_FTS is greater than DVFS_CH and EXH_FTS schemes by: (6.73%, 6.18%), (6.76%, 5.8%), (7.68%, 6.75%), (8.74%, 8.45%), (8.61%, 8.8%) for number of tasks of 10, 20, 30, 40 and 50, respectively. The results of our proposed algorithm always outperform those of the others, which show the efficiency of the DVFS_FTS Algorithm.

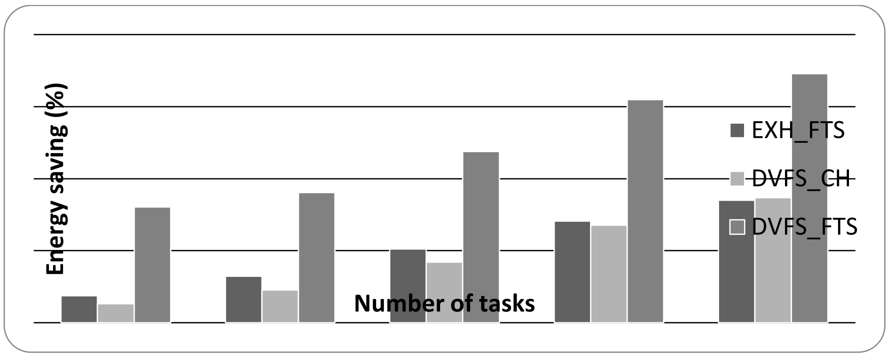

Fig. 6. The impact of application size on energy saving considering *k*=3 faults

In the third set of experiments, we show the impact of checkpointing overhead on the performance of algorithms (Fig. 7). In this set of experiments, we set application size $\Gamma = 20$ tasks, *k* =3 faults and vary *O* from 1 up to 20%. As can be seen from the figure, the energy saving of the three schemes decreases when *O* increases. However, the ES of DVFS_FTS decreases about 5.87% when *O* increases from 1 up to 20% and less than the ES of DVFS_CH and EXH_FTS decrease about 6.5% and 6.76%, respectively.

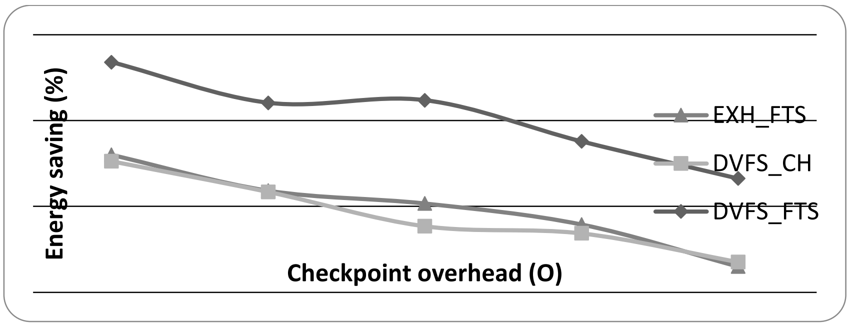

Fig. 7. The impact of checkpoint overhead on energy saving considering *k*=3 faults

From these experiments, we can resume that the proposed algorithm DVFS_FTS outperforms the other two algorithms.

8. Conclusion

Fault-tolerance and energy minimization are two major concerns in today's real-time embedded system designs. In this paper, we have studied the trade-off between fault tolerance and energy minimization in hard real-time systems running on multiprocessor platforms. This trade-off arises from the fact that fault tolerance and energy conservation exploit the released slack time to achieve more energy saving and to improve reliability, respectively. Thus, we first propose an efficient fault-tolerance approach that combines uniform checkpointing with rollback policy and collaborative active replication to explore hardware resources and timing constraint. We then present our fault-tolerant scheduling algorithm DVFS_FTS that exploits

DVFS technology to reduce energy consumption and the proposed fault-tolerance approach to tolerating K transient faults for applications that can be modeled with a DAG (precedence-constrained applications). Simulation results have shown that the proposed algorithm achieves a considerable amount of energy saving compared to EXH_FTS and DVFS_CH algorithms.

Our work remains opening to future contributions like extend the proposed algorithm to heterogeneous multiprocessor platforms and improve the proposed collaborative active replication to achieve more energy saving. Furthermore, we will study another checkpointing strategy to minimize the optimal number of checkpoints for reducing scheduling length and energy consumption.

References

1. D j o s i c, S., M. J e v t i c. Dynamic Voltage and Frequency Scaling Algorithm for Fault Tolerant Real-Time Systems. – Microelectronics Reliability Journal of Elsevier, Vol. **53**, 2013, pp. 1036-1042.
2. S a l e h i, M., M. K. T a v a n a, S. R e h m e n, M. S h a f i q u e, A. E j l a l i, J. H e n k e l. Two-State Checkpointing for Energy-Efficient Fault Tolerance in Hard Real-Time Systems. – IEEE Trans. on Very Large Scale Integration (VLSI) Systems, Vol. **24**, 2016, pp. 2426-2437.
3. L i, Z., L. W a n g, S. R e n, G. Q u a n. Energy Minimization for Checkpointing-Based Approach to Guaranteeing Real-Time Systems Reliability. – In: Proc. of 16th IEEE Int. Symp. Object/Compon./Service-Oriented Real-Time Distrib. Comput. (ISORC'13), 2013, pp. 1-8.
4. L i, Z., S. R e n, G. Q u a n. Energy Minimization for Reliability-Guaranteed Real-Time Applications Using DVFS and Checkpointing Techniques. – Journal of Systems Architecture, Vol. **61**, 2015, pp. 71-81.
5. K r i s h n a, C. M. Fault-Tolerant Scheduling in Homogeneous Real-Time Systems. – ACM Computing Surveys, Vol. **46**, March 2014, No 4, Article 48. 34 p.
6. M a h m o o d, A., S. A. K h a n, F. A l b a l o o s h i, N. A w w a d. Energy-Aware Real-Time Task Scheduling in Multiprocessor Systems Using a Hybrid Genetic Algorithm. – Electronics, Vol. **6**, 2017, No 2. 40 p.
7. W e i, T., P. M i s h r a, K. W u c, J. Z h o u. Quasi-Static Fault Tolerant Schemes for Energy-Efficient Hard Real-Time Systems. – Systems and Software Journal of Elsevier, Vol. **85**, 2012, pp. 1386-1399.
8. Z h u, X., R. G e, J. S u n c, C. H e. 3E: Energy-Efficient Elastic Scheduling for Independent Tasks in Heterogeneous Computing Systems. – Systems And Software Journal of Elsevier, Vol. **86**, 2013, pp. 302-314.
9. A s s a y a d, I., A. G i r a u l t, H. K a l l a. Scheduling of Real-Time Embedded Systems under Reliability and Power Constraints. – In: Proc. of International Conference on Complex Systems (ICCS'12) , IEEE, November 2012.
10. S a m a l, A. K., R. M a l l, C. T r i p a t h y. Fault Tolerant Scheduling of Hard Real-Time Tasks on Multiprocessor System Using a Hybrid Genetic Algorithm. – Swarm and Evolutionary Computation Journal of Elsevier, 2013.
11. G a n, J., F. G r u i a n, P. P o p, J. M a d s e n. Energy/Reliability Trade-Offs in Fault-Tolerant Event-Triggered Distributed Embedded Systems. – In: Proc. of 16th Asia South Pacific Design Automation Conference ASP-DAC, 2011, pp. 731-736.
12. K u m a r, A., B. A l a m. Improved EDF Algorithm for Fault Tolerance with Energy Minimization. – In: Proc. of IEEE International Conference on Computational Intelligence & Communication Technology (CICT'15), Ghaziabad, India, February 2015.
13. H a n, Q., M. F a n, G. Q u a n. Energy Minimization for Fault Tolerant Real-Time Applications on Multiprocessor Platforms Scheduling Using Checkpointing. – In: IEEE International Symposium on Low Power Electronics and Design (ISLPED), Beijing, China, September 2013, pp. 76-81.

14. I z o s i m o v, V., P. P o p, P. E l e s, Z. P e n g. Scheduling and Optimization of Fault-Tolerant Embedded Systems with Transparency/ Performance Trade-Offs. – ACM Trans. Embedded Computing Systems, Vol. **11**, 2012, No 3. 61 p.
15. Z h a n g, Y., K. C h a k r a b a r t y. A Unified Approach for Fault Tolerance and Dynamic Power Management in Fixed-Priority Real-Time Embedded Systems. – IEEE Trans. Computer-Aided Design of Integrated Circuits And Systems, Vol. **25**, 2006, pp. 111-125.
16. I z o s i m o v, V., P. P o p, P. E l e s, Z. P e n g. Scheduling of Fault Tolerant Embedded Systems with Soft and Hard Timing Constraints. – In: Proc. of 2008 Design, Automation and Test in Europe Conference (DATE'08), 2008, pp. 915-920.
17. M o t a g h i, M. H., H. R. Z a r a n d i. DFTS: Dynamic Fault-Tolerant Scheduling for Real-Time Tasks in Multicore Processors. – Microprocessors and Microsystems Journal of Elsevier, Vol. **38**, 2014, pp. 88-97.
18. H a n, Q., M. F a n, L. N i u, G. Q u a n. Energy Minimization for Fault Tolerant Scheduling of Periodic Fixed-Priority Applications on Multiprocessor Platforms. – In: Proc. of 2015 Design, Automation and Test in Europe Conference and Exhibition (DATE'15), 2015, pp. 830-835.
19. P o p, P., V. I z o s i m o v, P. E l e s, Z. P e n g. Design Optimization of Time-and- Cost-Constrained Fault-Tolerant Embedded Systems with Checkpointing and Replication. – IEEE Trans. Very Large Scale Integration Systems, Vol. **17**, 2009, 389-340.
20. T a v a n a, M. K., N. T e i m o u r i, M. A b d o l l a h i, M. G o u d a r z i. Simultaneous Hardware and Time Redundancy with Online Task Scheduling for Low Energy Highly Reliable Standby-Sparing System. – ACM Trans. Embedded Computing Systems, Vol. **13**, 2014, No 4. 86 p.
21. Z h a o, B., H. A y d i n, D. Z h u. Shared Recovery for Energy Efficiency and Reliability Enhancements in Real-Time Applications with Precedence Constraints. – ACM Trans. Des. Autom. Electron. Syst., Vol. **18**, March 2013, No 2, Article 23. 21 p.
22. Z h a n g, L., K. L i, Y. X u, J. M e i, F. Z h a n g, K. L i. Maximizing Reliability with Energy Conservation for Parallel Task Scheduling in a Heterogeneous Cluster. – Information Sciences, Vol. **319**, 2015, pp. 113-131.
23. Z a h a f, H. E. Energy Efficient Scheduling of Parallel Real-Time Tasks on Heterogeneous Multicore Systems. Ph.D. Université de Lille 1, Sciences et Technologies, 2016.
24. E l e s, P., V. I z o s i m o v, P. P o p, Z. P e n g. Synthesis of Fault-Tolerant Embedded Systems. – In: Proc. of 2008 Design, Automation and Test in Europe Conference (DATE'08), 2008, pp. 1117-1122.
25. G i r a u l t, A., H. K a l l a. A Novel Bicriteria Scheduling Heuristics Providing a Guaranteed Global System Failure Rate. – IEEE Trans. on Dependable and Secure Computing, Vol. **6**, 2009, pp. 241-254.
26. H u, Y., C. L i u a, K. L i, X. C h e n a, K. Li. Slack Allocation Algorithm for Energy Minimization in Cluster Systems. – Future Generation Computer Systems, Vol. **74**, 2016, pp. 119-131.
27. M e l h e m, R., D. M o s s e, E. E l n o z a h y. The Interplay of Power Management and Fault Recovery in Real-Time Systems. – IEEE Trans. Comput., Vol. **53**, 2004, pp. 217-231.
28. Q a m h i e h, M. Scheduling of Parallel Real-Time DAG Tasks on Multiprocessor Systems. Ph.D. Paris-Est University, 2015.
29. A r a r, C., M. S. K h i r e d d i n e. An Algorithm Based on Replication and Deallocation Efficient Fault-Tolerant Multi-Bus Data Scheduling Algorithm Based on Replication and Deallocation. – Cybernetics and Information Technologies, Vol. **16**, 2016, No 2, pp. 69-84.
30. B a c h i r, M., H. K a l l a. A Fault Tolerant Scheduling Heuristics for Distributed Real Time Embedded Systems. – Cybernetics and Information Technologies, Vol. **18**, 2018, No 3, pp. 48-61.

Received: 14.06.2019; Second Version: 15.11.2019; Accepted: 22.11.2019