International Journal of Computing and Digital Systems
2024, VOL. 17, NO. 1, 1–10

http://dx.doi.org/10.12785/ijcds/XXXXXX

A Novel Fault Tolerant Scheduling Approach with Energy Optimization for Real-Time Embedded Systems

NEDJOUA LOUCHENE [1], RIADH HOCINE [1], MALIKA BACHIR [2] and SALIM KALLA [1]

[1]*Department of Computer Science - University of Batna2, LaSTIC Laboratory, BATNA, ALGERIA*
[2]*Department of Common Core in Mathematics and Computer Science - University of Batna2, LaSTIC Laboratory, BATNA, ALGERIA*

Received 12 March 2024, Revised 15 June 2024, Accepted 10 July 2024

Abstract: In this paper, we address two critical yet conflicting aspects of real-time embedded systems: fault tolerance and energy consumption. We propose an Optimizing Energy consumption and Fault-Tolerant Scheduling Algorithm (OE_FTS), designed to achieve simultaneous minimization of energy consumption and maximization of system reliability. OE_FTS is based on a hybrid combination of active and passive replication and the Dynamic Voltage and Frequency Scaling (DVFS) technique. Active and passive replication are employed for tolerating multiple transient faults to improve reliability; therefore, Dynamic Voltage and Frequency Scaling (DVFS) technique is used for minimizing power consumption. We classify real-time tasks into critical and non-critical categories. Critical tasks are characterized by tight, stringent deadlines and limited execution time, while non-critical tasks have more flexible and less strict deadlines. This classification is used to decide which type of redundancy to apply and how to exploit the available slack time to lower the CPU frequency and voltage, thereby reducing energy consumption. Our technique is applied in a system designed to manage and execute several dependent tasks scheduled on a set of homogeneous processors linked by a multi-point connection. Our experimental findings confirm the effectiveness of the proposed approach which improves reliability while managing energy consumption.

Keywords: Real-time embedded systems, Active Redundancy, Passive Redundancy, Fault tolerance, Dynamic Voltage Scaling, Minimizing Energy Consumption

1. INTRODUCTION

Whether we realize it or not, embedded systems are an integral part of our daily lives. They represent the core of technological innovation, contributing to the improvement of our environment and our interconnected communication.

Developing real-time embedded systems requires meeting multiple conditions such as satisfying time constraints, ensuring fault tolerance, and minimizing energy consumption [1]. In the design of real-time embedded systems, fault tolerance and energy consumption are critical factors that require careful consideration. However, low energy usage and high fault tolerance are contradictory objectives and are often at odds with temporal constraints.

Fault tolerance is the capacity of a system to operate continuously and properly even when faults are present [2], and is ensured by several methods, with redundancy being the most widely used. Redundancy can be applied to either software or hardware; in the context of software redundancy, it can be temporal or spatial. Spatial redundancy, in turn, can be classified into three types: active, passive, or hybrid. As we focus on embedded systems, our attention is directed specifically towards software redundancy, particularly using spatial redundancy. Faults can be categorized as permanent, intermittent, or transient, depending on their duration. Permanent faults persist until the defective component is exchanged or fixed, while transient faults occur once and then disappear. Intermittent faults are identified by the occurrence of a fault, its disappearance, and then its reappearance, repeating the pattern [3]. Transient faults have recently gained more interest compared to permanent faults, as they represent the most frequent type of failure in such systems [4]. For this reason, we have focused on transient faults.

Energy management is influenced by multiple factors. Increasing the frequency speeds up task execution but also raises energy consumption. Conversely, lowering the frequency extends execution time, which can make it difficult to meet deadlines. Thus, a compromise occurs between runtime and energy consumed [5]. Additionally, reducing the supply voltage to save energy negatively impacts circuit reliability by increasing the occurrence of transient faults [4] [6]. Various energy management strategies have been

E-mail address: n.louchene@univ-batna2.dz, riadh.hocine@univ-batna2.dz, m.bachir@univ-batna2.dz, s.kalla@univ-batna2.dz

developed to lower energy consumption in real-time systems, including the standby technique and dynamic voltage and frequency scaling (DVFS). DVFS is the most popular of these techniques and is supported by the majority of today's processors.

As our objective is to find a fault-tolerant methodology for embedded and distributed real-time systems while guaranteeing time constraints and minimizing energy consumption, we propose in this study a new heuristic that combines active and passive replication to handle a fixed number of arbitrary transient faults, while ensuring timing constraints (deadlines) and utilizing the the DVFS technique to lower energy consumption.

We classify real-time tasks into critical and non-critical categories. Critical tasks are characterized by close and stringent deadlines with very limited execution time, whereas non-critical tasks have deadlines that are less strict and more flexible. Active replication is used for critical tasks to ensure that their deadlines are met, while passive replication is employed for non-critical tasks. In the latter case, we exploit the available slack time to lower the CPU frequency and voltage, thereby reducing energy consumption. This is why we have integrated the dynamic voltage and frequency scaling (DVFS) technique to achieve our goal of reducing energy consumption.

Our technique is applied within a system designed to manage and execute several dependent tasks. These tasks are scheduled on a set of homogeneous processors that are linked via a communication bus.

The organization of this paper is as follows: Section 2 offers an overview of related work. The system models discussed in this paper are presented in Section 3. The proposed fault tolerance methodology is detailed in Section 4, followed by a discussion on the fault tolerance methodology utilizing the dynamic voltage and frequency scaling (DVFS) technique in Section 5. Section 6 introduces the proposed OE_FTS algorithm. Sections 7 and 8 cover simulation parameters and results, respectively. Finally, the paper concludes in Section 9.

2. RELATED WORKS

The optimization of scheduling strategies based on two criteria -meeting temporal constraints and reliability- has been the subject of several published studies. Additionally, some studies focus on optimizing scheduling rules using three criteria, including power consumption.

Assayad et al [7] have proposed a technique that is a list scheduling heuristic. This heuristic uses a bi-criteria compromise function to prioritize the tasks that need to be scheduled and identify the subset of processors where these tasks should be allocated. The approach employs active redundancy of tasks.

Alain Girault et al [8] proposed an approach based on active redundancy that can tolerate a specified number of failed communication links and arbitrary processors. The process involves two steps: the first step transforms a non-redundant graph specification into one that incorporates redundant software components. Following this, the software components of the new redundant graph are allocated in terms of space and time.

The focus of the study in [9] was hardware faults, particularly in the area of communication. The author proposed a strategy based on both active and passive replicas. The fault-tolerant data scheduling optimization problem, considering two types of backup copies, is formulated using linear programming with the aim of reducing scheduling length.

The study in [10] has proposed fault-tolerant scheduling heuristics to tolerate multiple transient faults. These heuristics are based on an active replication strategy and checkpointing technique, aiming to maximize reliability. Additionally, the approach utilizes dynamic voltage frequency scaling to minimize energy consumption.

Salim Kalla et al. [11] have proposed an approach that tolerates transient faults and reduces energy consumption by employing graceful degradation, which incorporates the dynamic voltage scaling technique. This approach aims to decrease battery life variability, consequently reducing overall energy consumption.

The strategy presented by Yeganeh-Khaksar et al. [12] attempts to fulfill the power consumption limit at the chip level while achieving system reliability. The tasks are initially assigned according to a reliability-aware lowest utilization strategy; after that, they are scheduled with the maximum power taken into consideration and the Earliest Deadline First (EDF) policy. In the end, the DVFS technique is applied, accounting for peak power and dependability, to meet thermal design power (TDP) requirements.

The study presented in [13] introduces two fault-tolerant scheduling algorithms that are energy-aware and based on the primary-backup approach known as 'Fault-Tolerant Energy-Efficient Task Scheduling with Delayed and Overloaded Backups (FEED-O)' and 'FEED-O with Dynamic Deferring (FEED-OD). In this approach, backup copies are allocated on an secondary processor using dynamic power management (DPM) to reduce energy consumption, whereas the primary tasks are executed on a compatible dynamic voltage scaling (DVS) processor. This research proposes a method for minimizing energy consumption in scheduling periodic tasks employing a monotonic strategy.

Tavana et al. [6] have proposed an approach that addresses reliability and energy consumption. by achieving reliability and tolerating transient faults through the use of standby-sparing and re-execution techniques. Additionally, they employ dynamic voltage scaling (DVS) for the main processor and dynamic power management (DPM) for the secondary unit to reduce energy consumption.

In Table I, we show a summary and overview of the proposed approaches.

This paper aims to address the scheduling problem a software architecture composed of dependent tasks on a hardware architecture consisting of homogeneous processors while guaranteeing tolerance to multiple transient faults and optimizing energy consumption while respecting temporal constraints, the primary contributions provided by this article are as follows:

- We have proposed a scheduling algorithm capable of tolerating K transient faults.

- To minimize energy, we take advantage of non-critical tasks to apply the DVFS technique. While with critical tasks, task replicas require execution at the highest possible frequency.

- Our approach relies on the AAA (Algorithm Architecture Adequation) methodology that aims to optimize the length of scheduling.

3. SYSTEM MODELS

A. Algorithm model

The architecture of the algorithm used is represented through a data flux graph [14], specifically a directed hypergraph known as the algorithm graph ALG. The vertexes of the algorithm's graph indicate the task components of the system, denoted as $T = \{T_i, T_j, \ldots, T_n\}$, and the arcs denote the data dependencies between tasks. These tasks are not preemptive, meaning that other tasks are unable to stop them. They are connected by precedence dependencies, where a task can be executed only after receiving data from its preceding tasks.

Tasks without any predecessors are known as input operations and act as the algorithm's entry points for data flow. Output operations, indicating the end outcomes, are tasks that have no successors after them.

A tuple (C_i, D_i) that includes the execution time of the task (C_i) and its deadline (D_i) is used to characterize each task. The utilization of the T_i task can be defined as shown below:

$$U_i = \frac{C_i}{D_i}, where \quad 0 \leq U_i \leq 1 \tag{1}$$

Figure 1 illustrates an algorithm graph example formed of eight dependent tasks $\{T_1, T_2, ...T_8\}$: T_1 and T_2 (resp. T_8) are input (resp. output) tasks; $T_3 - T_7$ are regular tasks.

B. Architecture model

The architecture consists of homogeneous processors, called Arc which is represented using a non-directed graph in which the nodes represent the processors and the edges indicate the links that connect them physically. Homogeneous processors mean that each task has a similar execution time on each processor. We have considered a bus network (multi-point link).

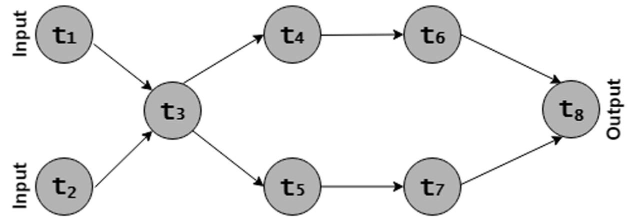

Figure 1. Algorithm graph example

The example of an algorithm graph illustrated in Figure 2.

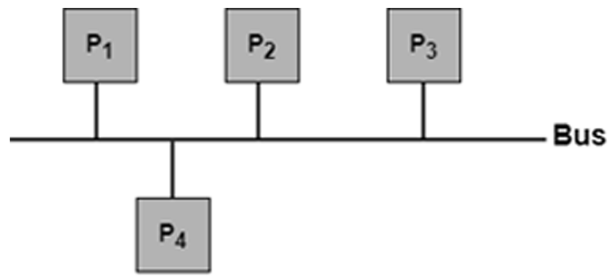

Figure 2. Architecture example

C. Fault model

This paper primarily focuses on transient faults rather than permanent faults because, when the processor voltage is reduced to save energy, there is a risk of transient failure triggered by even extremely low-energy particles producing critical charges [15]. We assume the capability to tolerate k processor transient faults. These faults may vary among different processors or occur within a single processor.

We use the Shatz and Wang [16] fault model, in which the maximum duration of fault only affects the current task that is executing on the faulty processor and does not have any impact on subsequent tasks.

D. Power model

In this study, we follow the energy model used in [5] [17] [18], Where the system's power usage P is determined by

$$P = P_s + h(P_{ind} + P_d) \tag{2}$$

$$P_d = C_{ef} V^2 f \tag{3}$$

Where:

P_s :is the static power (can only be removed by turning off the all system)

P_{ind}:is the frequency independent active power, constant, generally known as the CPU processing speed independent power.

TABLE I. summary and overview of the proposed approaches

Ref.	Model for Application	Model for Architecture	Fault-Tolerant Technique	Aims Constraints	Energy Management Technique
[7]	Acyclic oriented graph	Heterogeneous processors	Active replication	Timing, Reliability	-
[8]	Data-flow graph	Heterogeneous processors	Active redundancy	Timing, Reliability	-
[9]	Directed Acyclic Graph	Heterogeneous Communication links	Active,passive backup	Timing, Reliability	-
[10]	Directed Acyclic Graph	Homogeneous processors	Checkpointing and active replication	Energy, Timing, Reliability	DVFS
[11]	Acyclic oriented graph	Heterogeneous processors	Re-execution	Energy, Timing, Reliability	DVFS
[12]	Periodic	Homogeneous multi-core	Replicas of periodic real-time tasks	Energy, Timing, Reliability	RPPA-DVFS
[13]	Periodic	Homogeneous Multi-core	Standby Sparing Primary Backup	Energy, Timing, Reliability	DVS, DPM
[6]	Directed Acyclic Graph	Homogeneous Multi-core	Standby-sparing re-execution	Energy, Timing, Reliability	DVS, DPM

P_d:is the frequency dependent active power, which includes the dynamic power of the processor and any power that depends on the speed of the CPU [17]. It is equivalent to 1 when the system is in an active state and 0 when in an inactive state.

C_{ef}:is the switch capacitance, is the supply voltage, and f is the operating frequency.

Only the frequency-dependent power P_d is considered and $P_s = 0$. Hence, the expression for the power usage is as follows:

$$P = C_{ef} V^2 f \tag{4}$$

Since $f \propto V$, and in [19] a polynomial of frequency with a degree of 3 can be used to express the dynamic power. Hence, we reformulate the power usage P in (5) as

$$P = C_{ef} f^3 \tag{5}$$

The energy consumed by task T_i is

$$E = P C_i \tag{6}$$

$$E(f_i) = C_{ef} f_i^2 C_i \tag{7}$$

Where C_i represents the execution time of a task under frequency f_i .

The total energy consumption E_{total} of processors while performing a set of tasks is as follows:

$$E_{total} = \sum_{i=1}^{n} E_i(f_i) \tag{8}$$

This study considers only processor energy consumption. Before presenting our new approach, which combines two new techniques for fault tolerance and energy savings, we first describe each of them in detail in Sections 4 and 5.

4. PROPOSED FAULT TOLERANCE METHODOLOGY

We propose a novel fault-tolerance methodology that combines a hybrid approach of active and passive redundancy to tolerate K transient faults of processors. Our primary objective is to meet temporal constraints and enhance reliability, even when faults are present.

We employ passive replication for non-critical tasks; it relies on replicating the software components of the algorithm in several copies. However, only one copy, called the primary copy of each component, is executed, and the other copies, called backups, will be executed only if a fault causes an error [20]. With passive replication, we need to ensure a fault detection mechanism [21]. For critical tasks, we use active replication when passive redundancy cannot satisfy the task deadline. Active replication is achieved by replicating the algorithm's software components in several copies and the execution of the same task simultaneously on multiple separate processors [20].

Based on the work presented by Motaghi and Zarandi [22], we determine the task utilization U_i of each task T_i to decide whether it is critical or noncritical.

The task would be considered critical when U_i is closer to 1; in this situation, the scheduler is unable to delay task execution. When faults occur, time-consuming fault-tolerant methods, like passive replication, will not be appropriate for critical tasks.

A threshold θ is used to determine if the task is critical or not as follows:

$$T_i = \begin{cases} Noncritical, & if U_i < \theta, \\ Critical, & if U_i > \theta, \end{cases} \quad where 0 < \theta < 1 \quad (9)$$

θ varies from one task to another. In Section 6, we explain how to calculate the criticality threshold for each task T on processor P.

In the following, we describe the different basics of our methodology:

A. Passive redundancy

Costs and energy consumption can be reduced by using a passive technique, which involves executing secondary copies of a task only when a failure of its processor is detected; and for this reason, an error detection technique is required. For this purpose, we insert an additional task referred to as a watchdog, denoted as W, appended as a successor to every task in the architectural graph [23].

Starting from an algorithmic graph called ALG and applying the principle of passive redundancy, we obtain a new transformed graph called ALGnew where:

- Every task is duplicated in K+1 replicas, where K signifies the maximum number of transient faults on processors. Among these K+1 replicas, only one, designated as the primary task, is actively executed. The other K replicas designated as the secondary tasks are on standby, ready to be activated when the primary task fails. These replicas are distributed on distinct processors.

- For each primary task, K watchdog (W) tasks are inserted. Each task W is positioned between the main task and its copy. The task W and its successor must be placed on the same processor.

The task W is responsible for receiving a signal indicating that its predecessor (primary task) has been well executed. Therefore, if it doesn't receive any signal after a certain time denoted Δ, which is owner to each task, it detects a failure in the processor P responsible for executing the primary task. Consequently, each task W activates its successor; the first replica which finishes its execution sends a blocking signal to the other replicas. Below, we summarize the main steps involved in transforming the algorithm graph using passive replication.

For each task T_i

- Make k secondary copies T_i^1 ,T_i^2,...T_i^K of the primary task T_i, (T_i must be placed on a different processor from its replicas).

- Place each T_i^j (j ranging from 1 to k) on k distinct processors.

- Place k wachdog tasks(W_i^j), each W_i^j is placed between T_i and T_i^j (j running from 1 to k).

- Execute only the primary copy T_i,

- If W_i fails to receive a signal of primary copy T_i following a certain period Δ then

- Each task W_i^j wakes up its successor (T_i^j)

- The first T_i^j that finishes execution blocks the others in progress

An example of passive replication is presented in Figure 3.

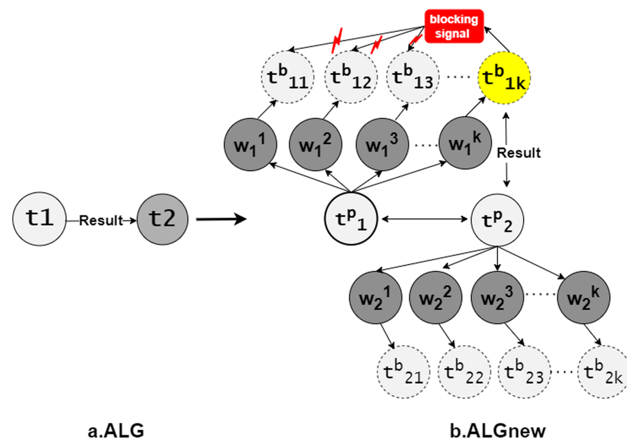

Figure 3. Transformation scheme of ALG with passive replication

The time out Δ is given by the following formula:

$$\Delta = C_i(T_{best}, P_{best}) + N \quad (10)$$

where:

T_{best}:is the best task selected for scheduling

P_{best}:is the best processor where task T_{best} will be placed

N: is an estimated value which determines a certain waiting time.

The worst-case response time named WRT_i of taskT_i employing passive redundancy when faults occur is

$$WRT_i = C_i + \Delta \quad (11)$$

Where C_i is the execution time of task T_i and Δ is the fault detection time of task T_i

$$WRT_i = 2C_i + N \quad (12)$$

Note that a fault affects only one task, so K faults mean

(K faulty tasks).

B. Active redundancy

To eliminate error detection time and optimize time execution for critical tasks, we use active replication. To apply active replication in our methodology, the starting graph ALG must be transformed into a new duplicated graph ALGnew in which all tasks are replicated in K+1 copies, placed on different processors, and are distributed for simultaneous execution. The first replica, which ends its execution, blocks the execution of the other replicas by sending a blocking signal. It also sends the result to its successor as well as its replicas. In the following, we summarize the main steps involved in transforming the algorithm graph using active replication.

For each task T_i

- Make k secondary copies T_i^1, T_i^2,... T_i^k of the primary task T_i, (T_i must be placed on a different processor from its replicas)

- Place each T_i^j (j ranging from 1 to k) on k separate processors

- Simultaneously execute the primary copy Ti and all its replicas,

- The first task that finishes execution blocks the others and sends the result to all replicas of its successor.

Figure 4 shows an example of active replication.

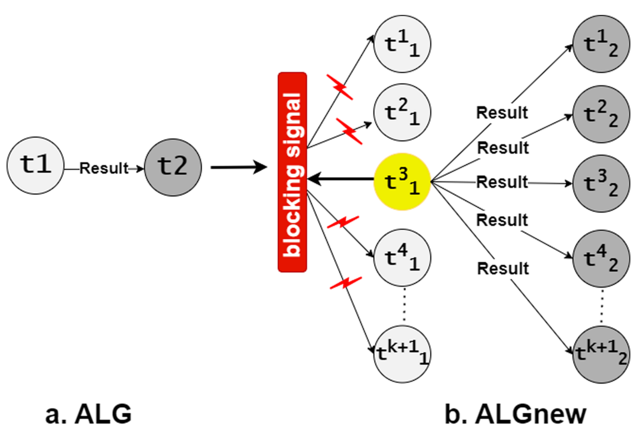

Figure 4. Transformation scheme of ALG with active replication

We have used the AAA (Adequation Algorithm Architecture) methodology for distributing tasks among processors, which is considered to be another important basis for our methodology. AAA relies on a cost function known as the scheduling pressure denoted $\sigma^{(n)}_{T_i,P_j}$, whose aim is to minimize the distribution/scheduling length [24]. The schedule pressure σ is calculated as follows for each T_i on each processor P_j :

$$\sigma^{(n)}_{T_i,P_j} = ST^{(n)}_{T_i,P_j} + \overline{st}^{(n)}(T_i) - R^{(n-1)} \tag{13}$$

$ST^{(n)}_{T_i,P_j}$: represents the earliest time when the task T_i on the processor P_j can begin to be executed.

$\overline{st}^{(n)}(T_i)$: represents the latest start time from end of T_i

$R^{(n-1)}$: is the critical path length of the partial schedule, which consists of tasks that have already been scheduled.

5. FAULT TOLERANCE METHODOLOGY UTILIZING DVFS

Dynamic voltage and frequency scaling (DVFS) is a common feature in the majority of contemporary processors, an energy-saving feature that permits a processor to operate at various voltage levels, with each voltage setting associated with a distinct operating frequency. Because of the direct proportionality between a processor's energy consumption and the square of its voltage, it is possible to significantly reduce processor energy consumption by lowering the CPU voltage and subsequently reducing its processing speed [25]. We have integrated DVFS into our proposed methodology to make use of the available slack time for greater energy savings.

In our proposed methodology, we employ active replication to satisfy temporal constraints and ensure a high level of reliability, particularly in situations where deadlines are close and time is limited. Task replicas with critical timing requirements should be executed at the highest possible frequency.

We assume the use of DVFS in passive replication for non-critical tasks, where their deadlines are not tight. We calculate the optimal frequency that can be assigned to each task. Assuming that each task has a deadline, the optimal frequency allows it to be completed before this deadline while minimizing energy consumption and being able to tolerate K faults thanks to passive replication, this leads to ensuring the reliability and energy efficiency of the tasks.

The optimal frequency that ensures task completes its execution within its deadline and should achieve the following requirements:

$$ST_i = \frac{2C_i}{f_i^{opt}} + N \leq D_i \tag{14}$$

$$f_i^{opt} \geq \frac{2C_i}{D_i - ST_i - N} \tag{15}$$

If $f_i^{opt}$is not found in F, we select neighboring frequencies

f_{L+1} ¡ f_i^{opt} ¡ f_L such that f_{L+1} , $f_L \in F$.

As a result, the energy consumption during the execution of task (T_i) is expressed as:

$$E(f_i^{opt}) = C_{ef} f_i^{opt2} \frac{C_i}{f_i^{opt}} = C_{ef} f_i^{opt} C_i = C_{ef} \frac{2C_i^2}{D_i - ST_i - N} \quad (16)$$

In the following, we present the algorithm of our new approach to fault-tolerant scheduling with energy optimization for real-time embedded systems which is called OE_FTS.

6. OE_FTS PROPOSED ALGORITHM

Our OE_FTS algorithm is designed to meet temporal constraints, minimize energy consumption, and maximize system reliability. It uses the AAA methodology to minimize the schedule length of the distribution and the scheduling of the algorithm on the hardware architecture while meeting temporal constraints.

To maximize system reliability, the algorithm employs active replication for critical tasks, executing them at the highest possible frequency to meet deadlines.

For non-critical tasks, passive replication is employed with the integration of dynamic voltage and frequency scaling (DVFS) to optimize energy consumption.

To determine the criticality of a task, the scheduler calculates the threshold "θ" of the best tasks selected by AAA. It is crucial to remember that, in the case of passive replication in the presence of faults, the maximum response time of a task must be less than its deadline ("D_i"). Therefore, we have:

$$ST_i + WRT_i < D_i \quad (17)$$

$$WRT_i < D_i - ST_i \quad (18)$$

Where WRT_i is worst-case response time of task T_i using passive replication, and Equation (12) is used to replace WRT_i for computing θ :

$$2C_i + N < D_i - ST_i \quad (19)$$

$$\frac{C_i}{D_i} < \frac{D_i - ST_i - N}{2D_i} \quad (20)$$

Finally, we can get the criticality

$$U_i < \frac{D_i - ST_i - N}{2D_i} = \theta \quad (21)$$

In Figure 5, we briefly present our optimizing energy and fault-tolerant scheduling in the OE_FTS Algorithm. The input for our method includes the ALGnew application, the variable K representing the tolerated number of transient faults, the architecture Arc, and the set of frequency levels F, as well as the constraints in real time. First we present its main outlines:

- The algorithm initiates by initializing the list of candidate tasks, which comprises tasks without predecessors (Line 1).
- For each task T_i in this candidate task list, we calculate the scheduling pressure $\sigma^{(n)}_{T_i,P_j}$ on each processor P_j in the Arc based on equation (13) (Line 5).
- Subsequently, we select the processor (P_{best}) that minimizes this scheduling pressure (Line 6).
- Among the pairs (T_i, P_{best}), we choose the one maximizing the scheduling pressure, resulting in (T_{best}, P_{best}) (Line 7).
- Task T_{best} is then placed and scheduled on processor P_{best} (Line 8).
- The scheduler computes the criticality of task T_{best} (Line 9). If task T_{best} is noncritical, passive redundancy and dynamic voltage and frequency scaling (DVFS) policy are applied (Lines 10-11), and T_{best} is executed under the calculated frequency f^{opt} based on (15) (Lines 13-14).
- If task T_{best} is critical, active replication is employed, and T_{best} is executed at the maximum frequency (Lines 16-17).
- After the execution of T_{best} , energy consumption is calculated (Line 18), and total energy is updated accordingly (Line 19). Finally, T_{best} is removed from the list of candidate tasks (Line 21).
- This allocation process iterates for all remaining tasks until none are left.

7. SIMULATIONS PARAMETERS

To evaluate our approach, we have implemented our heuristic on a set of software graphs (graph algorithm) randomly generated by a random graph generator, inspired by the work of H.Kalla in [24] using a set of parameters that influence our results. The parameters that we considered effective to study the performance of our methodologies are the number of faults, the number of processors, and the number of tasks. A description of the parameters and their values is presented in the Table II.

TABLE II. Parameters for simulation

Parameter	Value
Number of tasks	$T = \{5, 10, 15, 20, 25, 30\}$
Number of processors	$P = \{3, 5, 7, 9, 15\}$
Execution time (ms)	$EXT = \{10, 100\}$
Number of faults k	$K = \{1, 3, 5\}$
Operating frequencies	$F = \{0.1, 0.2, \ldots, 1\}$
C_{ef}	assume = 1

The general goal of our simulations is to examine how

Inputs:
$T = \{T_1, T_2, \ldots, T_n\}$ (Set of tasks)
$P = \{P_1, P_2, \ldots, P_m\}$ (Set of processors)
$F = \{f_1, f_2, \ldots, f_L\}$ (Set of frequencies)
ALGnew transformed algorithm graph
K transient faults
Temporal constraints (*the deadline*)

Initialization
Initialize the list of candidate tasks and the list of already placed tasks:
1. $T_{cand}^{(1)}$= {tasks from ALGnew without predecessors};
2. $T_{fin}^{(1)} = \varnothing$;
3. $E_{total} = 0$

DISTRIBUTION AND SCHEDULING LOOP
4. While $T_{cand}^{(n)} \neq \varnothing$ do
SELECTION
5. Calculate for each candidate T_i from $T_{candidat}^{(n)}$ and each processor $P_j \in$ Arc scheduling pressure (Equation (13));
6. Select for each candidate T_i the processor P_{best} that minimizes the scheduling pressure;
7. Select the best pair (T_{best} , P_{best}) that maximizes the scheduling pressure.

DISTRIBUTION AND SCHEDULING
8. Place this candidate T_{best} on the processor P_{best} (spatial allocation, temporal allocation);
9. Calculate the utilization of task U_i(equation (1)) and the criticality threshold θ (Equation (20))
10. if $U_i < \theta$ then
{
11. Apply passive replication
12. Apply DVS
13. Compute f_i^{opt} based on (Equation (15))
14. Execute T_{best} under f_i^{opt} frequency
}
15. Else
{
16. Apply Active replication
17. Execute T_{best} under *f* maximum frequency
}
18. Compute the energy consumption $E_i(f_i)$
19. $E_{total} = E_{total} + E_i\ (f_i)$

UPDATE
Update the list of candidate tasks and already placed tasks:
20. $T_{fin}^{(n+1)} = T_{fin}^{(n)}\ U \{ T_{best} \}$
21. $T_{cand}^{(n+1)} = T_{cand}^{(n)} - \{ T_{best} \}\ U \{ T \in succ\ (T_{best}^{p}) \mid pred\ (T_{best}^{p})\ T_{cand}^{(n+1)} \}$
22. **End While**
23. **END OF ALGORITHM**

Figure 5. The proposed OE_FTS Algorithm

changes in the number of faults and tasks affect energy consumption. Additionally, we aim to analyze the effects of the number of processors on both overall execution time and energy consumption. We compare our approach with the approach introduced in [10], called DVFS fault-tolerant scheduling (DVFS_FTS).

8. RESULTS

In the initial simulation, as illustrated in Figure 6, we plotted the energy consumption while maintaining the number of tasks at T =10, and the number of processors at P= 4, with the value of K varying from 1 to 5.

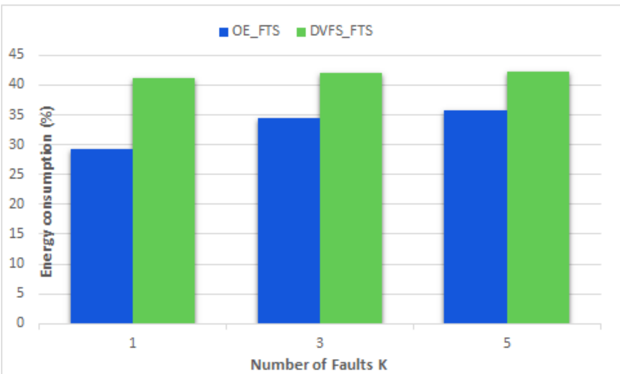

Figure 6. Energy consumption in function of number of faults

As seen in Figure 6, our approach OE_FTS, outperforms DVFS_FTS in terms of energy consumption. With 1 fault, the energy consumption of OE_FTS (29.18%) is lower than that of DVFS_FTS (41.2%). Similarly, with 5 faults, our approach consumes (35.75%) less than DVFS_FTS, which consumes (42.3%), we can clearly see that energy consumption increases with the increase in the number of transient faults. This is explained by the replication of tasks triggered by transient failures, resulting in more processor activity and therefore higher energy consumption. This was expected, as the two criteria maximizing reliability (tolerating a large number of faults) and minimizing energy consumption are contradictory.

In the second simulation, shown in Figure 7, we have varied the number of tasks within the interval of 5 to 30 tasks.These tasks were scheduled on an architecture graph with 5 processors, and 2 probabilistic faults with a value of K = 2.

In this case, it's clear that each time the number of tasks executed increases, energy consumption also increases, and this is due to processor occupancy rates (the workload). When comparing energy consumption between OE_FTS and DVFS_FTS, we observe OE_FTS consumes less energy than DVFS_FTS. For instance, with 10 tasks, energy consumption of OE_FTS (29,25%) is lower than that of DVFS_FTS (43%). Similarly, with 20 tasks, energy consumption of OE_FTS (33,62%) is lower than that of DVFS_FTS (42%) and with 30 tasks, energy consumption

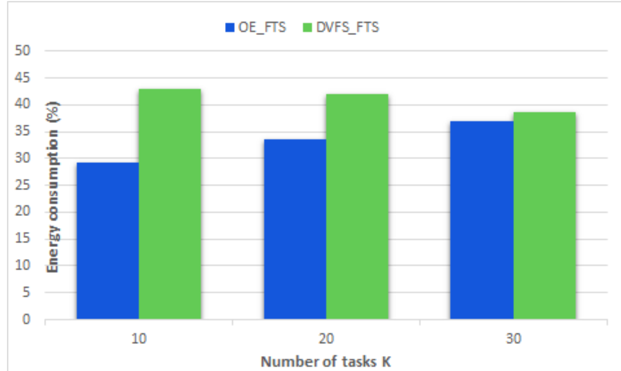

Figure 7. Effect of number of tasks on Energy consumption

of OE_FTS (36,9%) is lower than that of DVFS_FTS (38,7%) .

The findings indicate that our approach surpasses the performance of DVFS_FTS.

In the third simulation, illustrated in Figure 8 and Figure 9, we have plotted energy consumption and schedule length while varying the number of processors (P from 3 up to 15), while keeping the application size T=20 and number of faults fixed K=2.

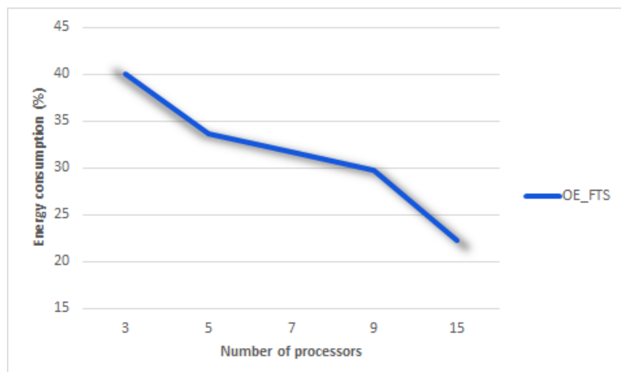

Figure 8. Effect of number of processors on Energy consumption

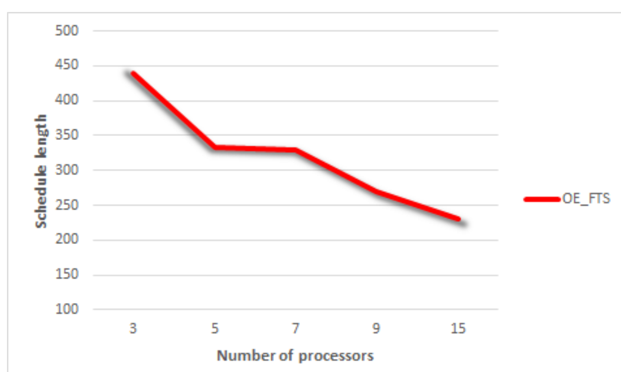

Figure 9. Effect of number of processors on schedule length

It is clear that with each increase in the number of processors, energy consumption correspondingly decreases. This is because the overall execution time is reduced by the simultaneous rather than sequential operation of all the processors.

As a result, there are two advantages: shorter execution times and lower overall energy consumption.

9. CONCLUSIONS

In this article, we have presented a novel approach to address the trade-off in real time embedded systems between antagonistic criteria such as fault tolerance and energy consumption. Our OE_FTS Algorithm combines active and passive replication of tasks tasks to enhance reliability while utilizing dynamic voltage and frequency scaling to lower energy consumption. The study takes into consideration factors such as the number of faults, processors, and tasks in the system. The results of the simulation demonstrate the effectiveness of our proposed approach, a correlation is observed between the energy consumption and and the rise in the number of faults, processors, and tasks. The proposed approach achieves improved reliability while managing energy consumption. As a direction for future work, and given the dynamic and continually evolving nature of the real time embedded systems field, we propose to test our heuristics on a heterogeneous architecture, incorporating additional parameters such as temperature, permanent faults, etc.

References

[1] S. Safari, M. Ansari, H. Khdr *et al.*, "A survey of fault-tolerance techniques for embedded systems from the perspective of power, energy, and thermal issues," *IEEE Access*, vol. 10, p. 12229, 2022.

[2] P. Jalote, "Fault tolerance in distributed systems," 1994.

[3] C. Wilwert, "Influence des fautes transitoires et des performances temps réel sur la sûreté des systèmes x-by-wire," 2005.

[4] J. Gan, F. Gruian, P. Pop, and J. Madsen, "Energy/reliability trade-offs in fault-tolerant event-triggered distributed embedded systems," 2011.

[5] I. Assayad, A. Girault, and H. Kalla, "Scheduling of real-time embedded systems under reliability and power constraints," 2012.

[6] M. K. Tavana, N. Teimouri, M. Abdollahi, and M. Goudarzi, "Simultaneous hardware and time redundancy with online task scheduling for low energy highly reliable standby-sparing system," *ACM Transactions on Embedded Computing Systems*, vol. 13, no. 4, p. 1, 2014.

[7] I. Assayad, A. Girault, and H. Kalla, "A bi-criteria scheduling heuristic for distributed embedded systems under reliability and real-time constraints," 2004.

[8] A. Girault, H. Kalla, and Y. Sorel, "An active replication scheme that tolerates failures in distributed embedded real-time systems: Processors and communication links failures," pp. 83–92, August 2004.

[9] C. Arar and M. S. Khireddine, "An efficient fault-tolerant multi-bus data scheduling algorithm based on replication and deallocation," *Cybernetics and Information Technologies*, vol. 16, no. 2, pp. 69–84, 2016.

[10] B. Kada and H. Kalla, "An efficient fault-tolerant scheduling approach with energy minimization for hard real-time embedded systems," pp. 102–117, 2019.

[11] S. Kalla, R. Hocine, H. Kalla, and A. Chouki, "Graceful degradation for reducing jitter of battery life in fault-tolerant embedded systems," *International Journal of Systems Science*, vol. 49, no. 11, pp. 2353–2361, 2018.

[12] A. Yeganeh-Khaksar, M. Ansari, and A. Ejlali, "Remap: Reliability management of peak-power-aware real-time embedded systems through task replication," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 1, p. 312, 2022.

[13] S. Bansal, R. K. Bansal, and K. Arora, "Energy efficient backup overloading schemes for fault-tolerant scheduling of real-time tasks," *Journal of Systems Architecture*, vol. 113, p. 101901, 2021.

[14] P. Lopez, "Approche par contraintes des problèmes d'ordonnancement et d'affectation: structures temporelles et mécanismes de propagation," 2003.

[15] D. Zhu, R. Melhem, and D. Mossé, "The effects of energy management on reliability in real-time embedded systems," pp. 35–40, November 2004.

[16] S. M. Shatz and J.-P. Wang, "Models and algorithms for reliability-oriented task-allocation in redundant distributed-computer systems," *IEEE Transactions on Reliability*, vol. 38, no. 1, p. 16, 1989.

[17] S. Djosic and M. Jevtic, "Dynamic voltage and frequency scaling algorithm for fault-tolerant real-time systems," *Microelectronics Reliability*, vol. 53, no. 7, p. 1036, 2013.

[18] A. Mahmood, S. A. Khan, F. Albalooshi, and N. Awwad, "Energy aware real-time task scheduling in multiprocessor systems using a hybrid genetic algorithm," *Electronics*, vol. 6, no. 2, p. 40, 2017.

[19] H. E. Zahaf, "Energy efficient scheduling of parallel real-time tasks on heterogeneous multicore systems," 2016.

[20] M. Bachir, R. Hocine, N. Louchene, and H. Kalla, "A fault-tolerant scheduling heuristics for distributed real-time embedded system," pp. 1–6, 2021.

[21] C. B. Holroyd, N. Yeung, M. G. Coles, and J. D. Cohen, "A mechanism for error detection in speeded response time tasks," *Journal of Experimental Psychology: General*, vol. 134, no. 2, p. 163, 2005.

[22] M. H. Motaghi and H. R. Zarandi, "Dfts: A dynamic fault-tolerant scheduling for real-time tasks in multicore processors," *Microprocessors and Microsystems*, vol. 38, no. 1, p. 88, 2014.

[23] M. Bachir and H. Kalla, "A fault-tolerant scheduling heuristics for distributed real-time embedded systems," *Cybernetics and Information Technologies*, vol. 18, no. 3, pp. 48–61, 2018.

[24] H. Kalla, "Génération automatique de distributions/ordonnancements temps réel, fiables et tolérants aux fautes," 2004.

[25] X. Zhu, R. Ge, J. Sun, and C. He, "3e: Energy-efficient elastic scheduling for independent tasks in heterogeneous computing systems," *Journal of Systems and Software*, vol. 86, no. 2, p. 302, 2013.