Comparison of Enhancing Methods for Primary/Backup Approach Meant for Fault Tolerant Scheduling

Petr Dobiáš, Emmanuel Casseau, Oliver Sinnen

Research Report
Comparison of Enhancing Methods for Primary/Backup Approach Meant for Fault Tolerant Scheduling

Tuesday 26th October, 2021

Petr Dobiáš, Emmanuel Casseau
Univ Rennes, Inria, CNRS, IRISA
Lannion, France
petr.dobias@esiee.fr

Oliver Sinnen
Department of Electrical and Computer Engineering
University of Auckland
Auckland, New Zealand

Abstract

This report explores algorithms aiming at reducing the algorithm run-time and rejection rate when online scheduling tasks on real-time embedded systems consisting of several processors prone to fault occurrence. The authors introduce a new processor scheduling policy and propose new enhancing methods for the primary/backup approach and analyse their performances. The studied techniques are as follows: (i) the method of *restricted scheduling windows* within which the primary and backup copies can be scheduled, (ii) the method of *limitation on the number of comparisons*, accounting for the algorithm run-time, when scheduling a task on a system, and (iii) the method of *several scheduling attempts*. Last but not least, we inject faults to evaluate the impact on scheduling algorithms.

Thorough experiments show that the best proposed method is based on the combination of the limitation on the number of comparisons and two scheduling attempts. When it is compared to the primary/backup approach without this method, the algorithm run-time is reduced by 23% (mean value) and 67% (maximum value) and the rejection rate is decreased by 4%. This improvement in the algorithm run-time is significant, especially for embedded systems dealing with hard real-time tasks. Finally, we found out that the studied algorithm performs well in a harsh environment.

Index Terms

Embedded systems, Fault tolerance, Multiprocessor systems, Online scheduling, Primary/backup approach, Real-time systems, Schedulability

I. Introduction

At present, demand on multiprocessor embedded systems for high performances and low energy consumption still increases in order to satisfy requirements to perform more and more complex computations. Moreover, the transistor size gets smaller and their operating voltage is lower, which goes hand in glove with higher susceptibility to system failure. Since systems are more vulnerable to faults, reliability becomes the main concern [1]. There are different methods to provide systems with fault tolerance [2]–[4] and the choice of their design depends on the application. For multiprocessor embedded systems, one of promising methods makes use of the redundancy and reconfigurable computing. Furthermore, multiprocessor systems are less vulnerable than a standalone processor because, in case of a processor failure, other processors remain operational.

The primary/backup (PB) approach is a method of fault tolerant scheduling on multiprocessor embedded systems making use of two task copies: the primary copy and backup copy [5]. It is a commonly used technique based on the redundancy to provide fault tolerance owing to its easy application and minimal system overheads. The PB approach can be adapted for satellites [6], evolutionary algorithms [7], employed in the virtualised cloud [8] or put into practice in energy-efficient scheduling algorithms for periodic real-time tasks [9]. Several additional enhancements [5], [10]–[12] to this approach have been already presented but few studies dealing with overall comparisons have been published. Moreover, the resiliency of the PB approach has been discussed in only few studies and with several restrictive assumptions [13]–[17].w In this report, we compare enhancing methods for PB approach when online scheduling tasks onto embedded systems dealing with hard real-time tasks and we analyse its resiliency.

Our motivation is to propose a scheduling algorithm for multiprocessor embedded systems with limited energy resources and subject to real-time and reliability constraints. Our main objective is to choose the method, which significantly reduces the algorithm run-time (in order to provide a solution more quickly and consume less energy) without worsening system performances when scheduling tasks on such embedded systems. The contributions of this report are as follows:

- Introduction of a new processor scheduling policy (called *first found solution search: slot by slot*) and its comparison with already existing ones;
- Introduction and analysis of new enhancing techniques based on the PB approach: (i) the method of *limitation on the number of comparisons*, accounting for the algorithm run-time, when scheduling a task on a system, and (ii) the method of *several scheduling attempts*;
- Evaluation of the overhead of the PB approach;
- Trade-off between the algorithm run-time (measured by the number of comparisons to find a free slot) and system performances (assessed by the rejection rate, i.e. the ratio of rejected tasks to all arriving tasks);
- Assessment of the fault tolerance of the PB approach;
- Mathematical programming formulation of the scheduling problem and comparison with the optimal solution delivered by CPLEX solver.

The remainder of this report is organised as follows. Section II summarises the related work and Section III presents our task, system and fault models and decribes the principle of the PB approach. Section IV then introduces our devised enhancements to the PB approach. Next, the experiments are described in Section V and analysed in Section VI. The discussion is in Section VII and Section VIII concludes the report.

II. Related Work

The aim of this section is to give a brief overview of methods meant for the PB approach.

The PB approach belongs to the category of task replication, which is used to improve the system reliability [18]–[22]. It consists in scheduling two identical task copies: primary one and backup one. The approach operates combined with a fault detection mechanism, which detects a fault for example using acceptance tests, like timing, coding, reasonableness or structural checks [2]. When a fault occurs during the execution of a primary copy, the results are provided by the corresponding backup copy. Nevertheless, if correct results are delivered by a primary copy, the corresponding backup copy is useless and can be deallocated to free its slot for new tasks [5], [23]. Manimaran et al. [24] considered that task copies may finish earlier than planned and reserved resources can be consequently released.

Some other method to improve the system schedulability is to overload several task copies on the same processor. While the *primary-backup overloading* of two different tasks, i.e. that a primary copy of one task overlaps with a backup copy of another task, [25] has not been put into practice in many algorithms due to its restrictions, the *backup-backup overloading*, i.e. that two or more backup copies of different tasks can overlap each other, [5] has been widely used. For example, Zheng et al. [12] devised the algorithm called *Minimum Replication Cost with Early Completion Time*. Their aim is (i) to overload as much as possible to reduce replication cost and (ii) to avoid continuous evaluation of replication costs on each processor. As for the latter objective, they only consider slots, which start and/or finish at the same time as already placed task copies.

Sometimes, it may happen that there is no free slot large enough to accommodate a task. In order to avoid its rejection, a *primary slack* was introduced [10]. This method shifts forward (if possible) already scheduled primary copies to create larger free slot for new task. Another possibility is to try scheduling later. Naedele [10] added a new task attribute called *decision deadline* defining the time instant before which the task must be treated, i.e. accepted or rejected.

In general, primary and backup copies of the same task do not overlap each other in time, which is called the *passive primary/backup approach*. However, when systems deal with tasks having tight deadlines, there may not be enough time to separately schedule both copies. Therefore, the *active approach* authorises the overlap of two copies of the same task on two different processors [11], [23] but at the cost of higher system load overheads because the system entirely or partially executes the backup copy while running the corresponding primary copy. This approach is not efficient when the backup deallocation is enabled.

When searching for a free slot, the chosen processor scheduling policy plays an important role. Whereas the *exhaustive search* on all processors [5], [23] finds the best solution, i.e. the primary and backup slots are respectively situated as soon and as late as possible within task window, other searches provide quicker solutions. These may not be the best ones because all processors are not tested. For example, Naedele [10] presented the *load-based processor selection* or the *sequential search*. Another possibility how to get a solution quickly is to make use of *restricted scheduling windows* [26], which reduce the number of tested slots.

To evaluate the resiliency against transient and permanent faults, the authors of [5] used a metric called the *Time To Second Fault* (TTSF), which is defined as the time it takes for the system to be able to tolerate a second fault after the first fault occurs. It was shown that the larger the task window, i.e. the difference between the deadline and the arrival time, the larger the TTSF. While there may be at most one transient or permanent fault at the same time in the system described in [5], the algorithm presented in [24] can tolerate more than one fault at any point of time (but in both cases at most one task copy can fail). To improve the schedulability in case of transient and intermittent faults, it was assumed in [13] that there is no additional fault during the minimum inter-fault time after one fault occurs. Consequently, the algorithm does not schedule backup copies for all primary copies. As for improvements related to permanent faults, an enhancement to find a schedule

reducing the overall reliability cost was proposed in [14], [27]. In general, it was shown that the higher the failure rate, the lower the system reliability and the lower the rejection rate.

Beitollahi et al. [15] injected faults based on a value of the mean time to failure (MTTF). Since this research considered only a uniprocessor system, a transient fault was considered. They concluded that the larger MTTF, i.e. the lower the failure rate, the lower the number of lost tasks. As faults are in general a random phenomenon in nature, the authors of [16] used a stochastic process to model transient faults. The faults were generated by the Poisson distribution and injected at task level. Zhu et al. [17] proposed a QoS-aware (Quality-of-Service) fault tolerant scheduling algorithm dealing with transient and permanent independent faults. At the same time, there is at most one fault. Their results showed that the reliability cost computed for the system is almost independent of the number of nodes, task arrival time, task deadline, task heterogeneity and system heterogeneity.

This section summed up the related work concerning the enhancements of the PB approach. A great deal of research has been conducted on scheduling algorithms using this approach to improve the schedulability and diminish the rejection rate but few studies have been carried out to evaluate and reduce the algorithm run-time. This is a major concern as far as embedded systems are targeted.

III. Assumptions and Scheduling Model

First of all, we give definitions of employed terms. We call *mapping*, a placing in space of a task onto one of the system processors, and *scheduling*, a placing in time of a task onto one particular system processor. Thus, in this report, scheduling implies both time and space from now on.

A. Task and System Models

In our task model, we assume aperiodic tasks. Every task has three attributes: arrival time a_i, computation time c_i and deadline d_i. The task window tw_i is defined as $d_i - a_i$ and can be also expressed as a multiple α of the computation time c_i.

Arriving tasks are online scheduled on the system without preemption. The system consists of P interconnected identical processors[1]. The processors share the same memory. In the case of distributed memory, delays of data transfers need to be considered but the principle of the method remains the same.

B. Fault Model

The fault model considers independent faults, which are mainly due to bit flips or temporary malfunction of functional units. Although not all faults generate an error, we consider the worst case in our research, i.e. that every fault causes an error. We assume that a fault detection mechanism exists and promptly informs when a permanent and/or transient fault occurs. A fault can be detected for example by acceptance tests, such as timing, coding, reasonableness or structural checks [2]. While the processor impacted by a transient fault recovers and is operational again, in the case of a permanent fault, the faulty processor is not used anymore.

We consider that only one processor failure (unless we carry out fault injection experiments as in Section VI-H) can arise at any instant of time. We also assume that the scheduler is made robust, for instance using a spare scheduler if necessary. Current processors have a failure rate of $1/120\,h^{-1}$ [28] and even though the reliability of P-processor system is lower our assumption holds. In fact, the authors in [3] proved that a system consisting of identical processors has its reliability (measured by mean time between faults (MTBF)) equal to the processor MTBF divided by the number of processors.

C. Primary/Backup Approach

The studied algorithm is based on the *primary/backup (PB) approach* [5], which is commonly used for its minimal resources utilisation and high reliability. Its principal rule is that, when a task arrives, two identical copies, the *primary copy* (PC) and the *backup copy* (BC), are created. The primary copy is scheduled as soon as possible and the backup one as late as possible in order to avoid idle processors just after the task arrival time and possible high processor load later. A *slot* is a time interval on processor schedule.

In order to improve the schedulability and to minimise resources usage, we consider the backup deallocation and the backup overloading [5].

*Definition 1 (**BC deallocation**):* Let t_i be a task having two task copies PC_i and BC_i. If PC_i was correctly executed, then BC_i can be deallocated and free its slot for new arriving tasks.

*Definition 2 (**BC overloading**):* Let P_x be a processor and t_i and t_j be two tasks. Backup copies BC_i and BC_j can overlap each other on the same processor unless $PC_i \in P_x$ and $PC_j \in P_x$ because, when a fault occurs on the processor P_x, both backup copies BC_i and BC_j may need to be executed.

Using the notation summarised in Table I, the conditions for the PB approach are as follows.

[1]In this report, we consider systems with only homogeneous processors for the sake of simplicity. Nevertheless, the studied model can be easily extended to systems with heterogeneous processors, e.g. [12].

Table I: Notations and definitions

Notation	Definition
P_x	Processor x
t_i	Task i
a_i	Arrival time of task t_i
c_i	Computation time of task t_i
d_i	Deadline of task t_i
tw_i	Task window of task t_i
f	Fraction of task window tw_i
s_i	Slack of task t_i
ps_i	Percentage of s_i within the tw_i
PC_i	Primary copy of task t_i
BC_i	Backup copy of task t_i
xC_i	PC or BC of task t_i
$start(xC_i)$	Start of the execution of PC_i or BC_i
$end(xC_i)$	End of the execution of PC_i or BC_i

*Condition 1 (**No overlap between primary and backup copies of the same task**):* Let t_i be a task having two task copies PC_i and BC_i. BC_i cannot start its execution before the end of PC_i, i.e. $end(PC_i) \leqslant start(BC_i)$. Otherwise BC_i needs to be executed (at least during the overlap with PC_i), which causes system load overheads when the backup deallocation is authorised.

*Condition 2 (**Respect of task deadline**):* Let t_i be a task having two task copies PC_i and BC_i and Condition 1 applies. None of task copies cannot start before task arrival and they must be executed prior to task deadline, i.e. $a_i \leqslant start(PC_i) < end(PC_i) \leqslant start(BC_i) < end(BC_i) \leqslant d_i$. Otherwise the input data may not be available and the results may not be useful anymore.

*Condition 3 (**Primary copy and backup copy processor constraint**):* Let t_i be a task having two task copies PC_i and BC_i. PC_i and BC_i cannot be scheduled on the same processor P_x, i.e. $PC_i \in P_x \Rightarrow BC_i \notin P_x$. Otherwise, if a fault occurs during the PC_i execution, the processor P_x may not recover and the BC_i execution may be faulty too.

*Condition 4 (**No overlap of primary copies on the same processor**):* Let PC_i and PC_j be respectively primary copies of t_i and t_j. A processor P_x can execute only one primary copy at the same time, i.e. (PC_i and PC_j) $\in P_x \Rightarrow end(PC_i) \leqslant start(PC_j)$ or $end(PC_j) \leqslant start(PC_i)$.

D. Mathematical Programming Formulation

In this section, we define the mathematical programming formulation of the studied scheduling problem. This formulation is used as a baseline to get the theoretical optimum and to compare it against the solution provided by our algorithms. It is as follows:

$$\max \sum_{i}^{\text{Set of tasks}} \text{task } t_i \text{ is accepted}$$

subject to

$$\begin{cases} \textbf{1)}\ PC_i \text{ scheduled } \Leftrightarrow BC_i \text{ scheduled} \\ \textbf{2)}\ a_i \leqslant start(PC_i) < end(PC_i) \leqslant start(BC_i) < \\ \quad end(BC_i) \leqslant d_i \\ \textbf{3)}\ PC_i \in P_x \Rightarrow BC_i \notin P_x \\ \textbf{4)}\ (PC_i \text{ and } BC_j) \in P_x \Rightarrow \\ \quad end(PC_i) \leqslant start(BC_j) \text{ or } end(BC_j) \leqslant start(PC_i) \\ \textbf{5)}\ (PC_i \text{ and } PC_j) \in P_x \Rightarrow \\ \quad end(PC_i) \leqslant start(PC_j) \text{ or } end(PC_j) \leqslant start(PC_i) \\ \textbf{6)}\ (BC_i \text{ and } BC_j) \in P_x \Rightarrow \\ \quad end(BC_i) \leqslant start(BC_j) \text{ or } end(BC_j) \leqslant start(BC_i) \end{cases}$$

The objective function is to maximise the number of accepted tasks, which is equivalent to minimise the task rejection rate. The first two constraints are related to the principle of the PB approach, i.e. every task has two no overlapping copies, which are delimited by the arrival time and deadline. The third constraint forbids primary and backup copies of the same task to be scheduled on the same processor. The last three constraints account for no overlap among task copies on one processor,

i.e. only one task copy can be scheduled per processor at the same time. Whereas the fourth and fifth constraints must be respected all the time, the last constraint is used only when the BC overloading is not authorised.

The problem is solved in CPLEX optimizer[2], which is a high-performance mathematical programming solver for linear programming, mixed-integer programming and quadratic programming. Since tasks aperiodically arrive and backup copies are deallocated once their corresponding primary copies were correctly executed, a dynamic aspect needs to be modelled in CPLEX solver. Therefore, at each task arrival, the main function updates task data (new task arrivals and deallocated backup copies) and launches a new resolution using current data set.

E. Processor Scheduling Policies

Three processor scheduling policies are presented in this report: the *exhaustive search* [5], the *first found solution search processor by processor* [10] and the *first found solution search slot by slot*. Algorithm 1 summarises main scheduling steps independent of the processor scheduling policy.

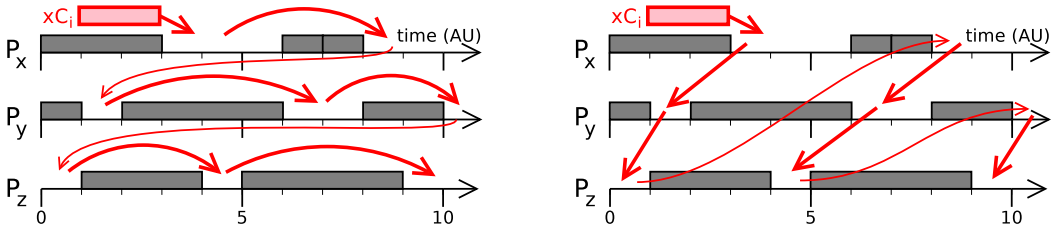
Algorithm 1 Primary/backup scheduling

Input: Task t_i
Mapping and scheduling of already scheduled tasks
Output: Updated mapping and scheduling

```
1: if new task t_i arrives then
2:     Map and schedule PC_i
3:     Map and schedule BC_i
4:     if PC_i and BC_i slots exist then
5:         Commit the task t_i
6:     else
7:         Reject the task t_i
8:     end if
9: end if
```

The *exhaustive search* (ES) tests all (P) processors to find a primary copy slot and $P-1$ processors to search for a backup copy slot in order to respect Condition 3. After such a search, the algorithm provides the best solution, i.e. the one having its primary copy scheduled as soon as possible and backup copy placed as late as possible.

The second and third processor scheduling policies are mainly meant for real-time systems, which may not have time to search for a solution on all processors, assess all possibilities and opt for the best one. The idea is to find a solution as quickly as possible and not necessarily the best one. Naedele [10] presented the *sequential search*, which we call the *first found solution search - processor by processor* (FFSS PbP). The algorithm goes through processors, one by one, until it finds a slot large enough to place a copy or until it scours all processors, as it is depicted in Fig. 1a. The arrows illustrate how the search is carried out in space (on processors) and in time (on a given processor) when a task copy xC_i arrives. There is no restriction on scheduling, which means that a primary copy can be scheduled rather late within the task window. This decreases the chance to place the backup copy within the remaining scheduling window and subsequently increases the task rejection rate.



(a) Processor by Processor (PbP) (b) Slot by Slot (SbS)

Figure 1: Principle of the First Found Solution Search (FFSS)

In order to improve the previous method and favour placing primary copies as soon as possible, we propose the algorithm called the *first found solution search - slot by slot* (FFSS SbS). It starts to check the first free slot on each processor and then, if solution was not found, it continues with next slots (second, third, ...) until a solution is obtained or it searches all free slots on all processors. The principle of this algorithm is illustrated in Fig. 1b.

[2]https://www.ibm.com/analytics/cplex-optimizer

The selection of the processor on which the search for a slot starts plays an important role in the system schedulability and workload distribution among processors [10]. To avoid non-uniformity of processor load for both PbP and SbS, the FFSS for a primary copy slot starts on the processor following the processor on which the primary copy of the previous task was successfully scheduled. The search then continues in increasing order of the processors until a slot is found or all processors are tested [10]. If the primary copy of new task is obtained on processor P_x, a search for a backup copy slot is carried out. It starts on processor P_{x-1} and it proceeds in decreasing order of the processors till a slot is found or no more processor is available.

IV. Proposed Enhancements

In this section, we introduce our devised enhancements to the PB approach.

A. Limitation on the Number of Comparisons

When scheduling a task, the simplest idea aiming at reducing the algorithm run-time is to limit the number of comparisons between slot duration and computation time c_i [10]. This number is computed for every task until it is definitely accepted or rejected. Every arriving task is assigned the maximum number of comparisons to search for its PC and BC slots. If this threshold is exceeded, the task is rejected. Otherwise, it is normally scheduled, i.e. accepted or rejected according to the baseline algorithm.

To justify this idea, we found out that accepted tasks require less comparisons than rejected tasks (in terms of mean values) and that the mean number of comparisons is significantly lower than the maximum number of comparisons, as shown in Fig. 2. This figure represents the mean and maximum numbers of comparisons for the PB approach with BC deallocation using FFSS SbS ($P = 14$, $TPL = 1.0$)[3] without limitation on the number of comparisons. Qualitatively similar results were obtained for the PB approach with BC deallocation and BC overloading. Consequently, when scheduling a new task, the probability that it will be successfully scheduled is lower when the number of comparisons is already high.

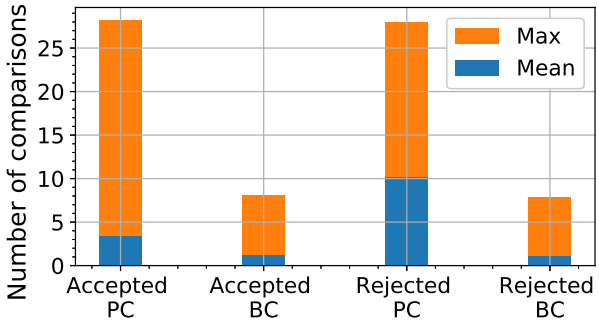


Figure 2: Mean and maximum numbers of comparisons for the PB approach with BC deallocation (FFSS SbS, $P = 14$, $TPL = 1.0$, no limitation on the number of comparisons)

The detailed analysis of the numbers of comparisons for accepted and rejected copies (not presented in this report) showed that the number of comparisons when scheduling a primary copy depends on the number of processors, whilst the one for backup copy is almost independent of the number of processors. In general, the mean number of comparisons for backup copy is between 1 and 2 and the maximum number of comparisons can exceed 10. In our simulations, we set the threshold at 5 to avoid that a task is often rejected due to missing free slot for BC. Therefore, we define the theoretical maximum value of run-time as reads:

$$rt_{limit} = rt_{limit}(PC) + rt_{limit}(BC) = \gamma \cdot P + 5 \tag{1}$$

where γ is the limitation coefficient for primary copies expressed in our simulation framework as a function of the number of processors.

To illustrate Equation 1, Fig. 3 plots the theoretical limitation on the maximum number of comparisons for the PB approach with BC deallocation as a function of the number of processors (FFSS SbS, $TPL = 1.0$). As a baseline, we consider our experimental results when a limitation is not used. Qualitatively similar results were obtained for the PB approach with BC deallocation and BC overloading.

[3]The *Targeted Processor Load* (TPL), defined in Section V-A, is a parameter related to the theoretical processor load when generating task arrivals. If $TPL = 1.0$, arrival times are generated so that every processor is considered to be working all the time at 100%.

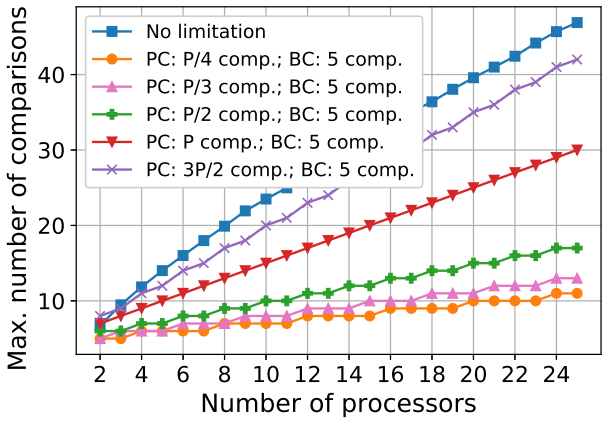


Figure 3: Theoretical limitation on the maximum number of comparisons for the PB approach with BC deallocation as a function of the number of processors (FFSS SbS, $TPL = 1.0$)

B. Restricted Scheduling Windows

The second method, studied in this report to diminish the algorithm run-time when scheduling a task, is called the *restricted scheduling windows*. This method has been already published in our report [26]. Its aims are 1) to avoid the mutual scheduling interference between primary and backup copies of the same task, 2) to reduce the run-time (measured by the number of comparisons carried out before definitely accepting or rejecting a task) and 3) to encourage to place the primary copies as soon as possible and the backup ones as late as possible, which increases schedulability when the BC deallocation is enabled. A *scheduling window* for primary or backup copy, respectively, is a time interval (subinterval of the task window) within which the respective copy can be scheduled. The size of scheduling window is defined by a parameter f representing the *fraction of task window*. The PC window of task t_i is thereby delimited by a_i and $a_i + f \cdot tw_i$ and the BC window by $d_i - f \cdot tw_i$ and d_i. In our algorithm, the fraction is within $0 < f \leqslant 1$, whereas it equals 1 for the conventional algorithm. An example of restricted scheduling windows is depicted in Fig. 4.

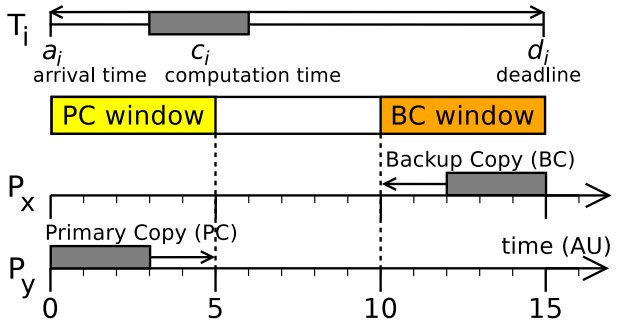


Figure 4: Primary/backup approach with restricted scheduling windows ($f = 1/3$)

The theoretical maximum value of the run-time rt when placing tasks with restricted scheduling windows was presented in [26] and is expressed as follows:

$$
\begin{aligned}
rt(PC) &= P \cdot N_{ps}(PC) \cdot \max\left(\frac{1}{\alpha}; \min\left(1 - \frac{1}{\alpha}; f\right)\right) \\
rt(BC) &= (P-1) \cdot N_{ps}(BC) \cdot \max\left(\frac{1}{\alpha}; \min\left(1 - \frac{1}{\alpha}; f\right)\right) \\
rt &= rt(PC) + rt(BC)
\end{aligned}
\tag{2}
$$

where N_{ps} is the number of all possible slots where a copy can be scheduled onto a processor and $\alpha = \frac{tw_i}{c_i}$.

Fig. 5 shows a trend of theoretical maximum run-time. It can be observed that, when the fraction of task window decreases, the run-time, expressed as the number of comparisons, is reduced because there are less possible slots to test.

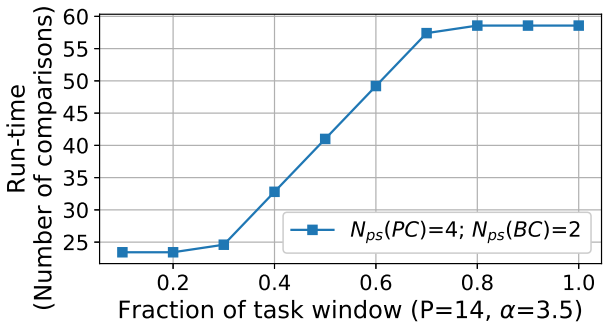


Figure 5: Example of theoretical maximum run-time

C. Several Scheduling Attempts

The previous two techniques mainly dealt with the reduction in the algorithm run-time, whereas the method described in this section focuses on the decrease in the rejection rate. Up to now, the algorithm had only one attempt to schedule a task carried out at arrival time a_i. However, it may sometimes happen that a task is rejected at the task arrival even though several time units later there is a slot freeing up and large enough to accommodate a task copy thanks to the BC deallocation. The aim of the proposed method is to retry the scheduling later, at the percentage ω of task window tw_i, and thus increase the chance for a task to be accepted.

An example for $\omega = 25\%$ is given in Fig. 6 and Algorithm 2 sums up the main scheduling steps of this method.

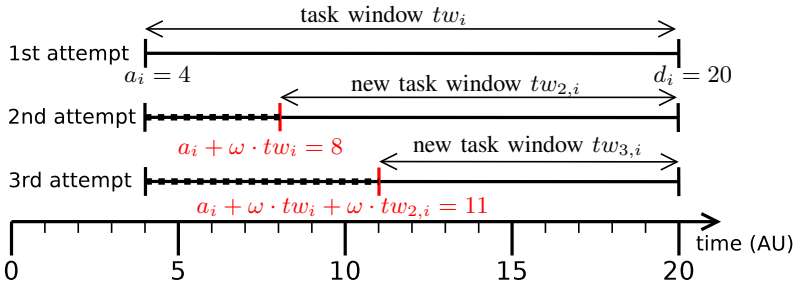


Figure 6: Three scheduling attempts at $\omega = 25\%$

Algorithm 2 Algorithm for several scheduling attempts

Input: Task t_i
Mapping and scheduling of already scheduled tasks
Output: Updated mapping and scheduling

```
if new task t_i arrives then
    Search for PC_i and BC_i slots for the first time
    if PC and BC slots exist then
        Commit the task t_i
    else
        while task unscheduled and new attempt authorised do
            Compute the time of new attempt to schedule t_i
            Search for PC_i and BC_i slots
            if PC and BC slots exist then
                Commit the task t_i
            else
                Reject the task t_i
            end if
        end while
    end if
end if
```

Table II: Simulation parameters

Parameter	Distribution	Value(s) in ms
Number of processors P		2 – 25
Computation time c	Uniform	1 – 20
Arrival time a	Poisson	$\lambda = \frac{average\ c}{TPL \cdot P}$
Deadline d	Uniform	$[\![a + 2c; a + 5c]\!]$

To evaluate the efficiency of this method, we define the *percentage of slack within the task window* ps_i of task t_i as follows:

$$ps_i = \frac{s_i}{tw_i} = \frac{tw_i - 2 \cdot c_i}{tw_i} \tag{3}$$

where s_i is the *slack*, i.e. the remaining time within the task window tw_i after subtracting twice the computation time c_i necessary for PC and BC to be executed. The higher the percentage ps_i, the higher the chance to schedule a task later than at its arrival time. Nonetheless, the higher the number of scheduling attempts for one task, the lower the probability for new scheduling attempts to be successful because there is less and less slack.

V. Experiment Framework

In this section, we describe our simulation scenario and define metrics used to evaluate our algorithms.

A. Simulation Scenario

Table II sums up simulation parameters. For each simulation scenario, 100 simulations of 10 000 tasks were treated and the obtained values were averaged. Unless otherwise stated (see Section VI-H), we consider that no fault occurs during our simulations and therefore all backup copies are deallocated when their respective primary copies finish.

Arrival times are generated using the Poisson distribution with parameter λ depending on the computation time, the number of processors and the targeted processor load. If the *Targeted Processor Load* (TPL) equals 1.0, arrival times are generated so that every processor is considered to be working all the time at 100%.

The simulations were conducted in a dicrete simulator that we wrote in Python.

To compare our results, we defined mathematical programming formulation as described in Section III-D and carried out resolutions in CPLEX optimizer using the same data set. Due to computational time constraints, only 16 resolutions were conducted and the results were averaged.

For simulations with faults, we take into account that the estimated processor fault rate is $1/120\,h^{-1} = 2.3 \cdot 10^{-6}$ fault/s [28], which corresponds to $5.8 \cdot 10^{-5}$ fault/s for 25-processor system. Therefore, we randomly inject faults at the level of task copies with fault rate for each processor between $1 \cdot 10^{-5}$ and $5 \cdot 10^{-2}$ fault/ms in order to assess algorithm performances not only in real conditions but also in harsher environment. Consequently, our assumption about only one processor failure at the same time may not be respected for higher fault rates[4], which may cause that a task having both primary and backup copies impacted does not contribute to the system throughput, defined in Section V-B. For the sake of simplicity, we consider only transient faults and that one fault can impact at most one task copy.

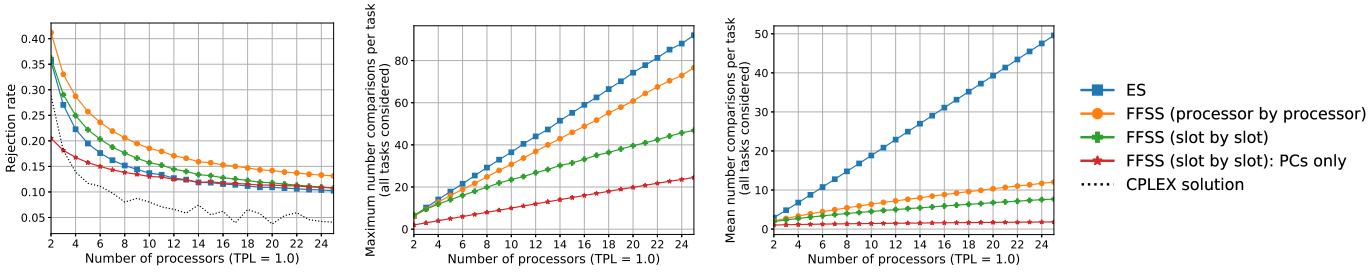
B. Metrics

The performances of our algorithms were evaluated based on the following metrics. The *rejection rate* is defined as the ratio of rejected tasks to all arriving tasks to the system. The *system throughput* counts the number of correctly executed tasks. In a fault-free environment, this metric is equal to the number of tasks minus the number of rejected tasks.

The algorithm run-time is evaluated by the *number of comparisons* accounting for the number of tested slots. One comparison accounts for evaluation whether a free slot is large enough to accommodate a task copy (PC or BC) on a given processor. All tasks are taken into account, no matter whether they are finally accepted or rejected. This metric is essential for embedded systems because it is related to rapidity of scheduling and energy consumption.

As our algorithm is meant for embedded systems dealing with hard real-time tasks, we try to reduce the algorithm run-time as much as possible without worsening system performances. Thus, our aim is to cut down on the number of comparisons first and then decrease the rejection rate.

[4]Inspired by [5], we use a metric, which we called the *Time To Next Fault* (TTNF), to evaluate the resiliency. It is defined as the time elapsed between a chosen time instant and the time when a new fault may occur without violating the assumption of only one fault in the system all at once. The worst-case value is obtained when a fault occurs at the beginning of PC having the longest c and the largest tw. In our scenario $TTNF_{\text{worst-case}} = c_{max} \cdot tw_{max} = 20 \cdot 5 = 100ms$, which implies fault rate of $1 \cdot 10^{-2}$ fault/ms for 25-processor system. Our results (not presented in this report) show that the mean value of TTNF is less than one half of the worst-case TTNF no matter the chosen method.



(a) Rejection rate (b) Maximum number of comparisons (c) Mean number of comparisons

Figure 7: Experimental results for the PB approach with BC deallocation comparing three processor scheduling policies and evaluating system overheads as a function of the number of processors ($TPL = 1.0$)

VI. Results

In this section, we first compare different processor scheduling policies. Then, we present the results showing the overhead of the PB approach and the comparison with the optimal solution from CPLEX solver. Afterwards, we separately analyse three enhancing methods (limitation on the number of comparisons, restricted scheduling windows and several scheduling attempts) in order to determine their parameters satisfying the best our objective. These methods are then combined together and their performances are assessed. Finally, we evaluate the fault tolerance of the PB approach using the best choice of enhancing techniques.

A. Comparisons of Processor Scheduling Policies

One of our contributions is a new processor scheduling policy called the *first found solution search - slot by slot*. In this section, it is compared to two already existing policies: the *exhaustive search* [5] and the *first found solution search - processor by processor* [10]. The results of the rejection rate, the maximum and the mean numbers of comparisons for the PB approach with BC deallocation as a function of the number of processors are depicted in Fig. 7.

Fig. 7a representing the rejection rate shows that the FFSS SbS achieves better results than the FFSS PbP. The ES is the best because this search tests all possibilities and chooses the solution placing the primary copy as soon as possible and the backup one as late as possible, which contributes to higher schedulability. Nevertheless, it can be seen that the gap between the FFSS SbS and the ES becomes smaller when the number of processors augments.

As for the maximum number of comparisons plotted in Fig. 7b, the FFSS SbS has notably lower values compared to the ES (for example 41% for 14-processor system) and the FFSS PbP. When considering the mean number of comparisons shown in Fig. 7c, it can be seen that the FFSS SbS requires significantly less comparisons than the ES (for instance 80% for 14-processor system).

Similar results were obtained for the PB approach with BC deallocation and BC overloading.

Since our new processor scheduling policy (FFSS SbS) performs well, it is chosen for further experiments.

B. Overhead of the Fault Tolerant System

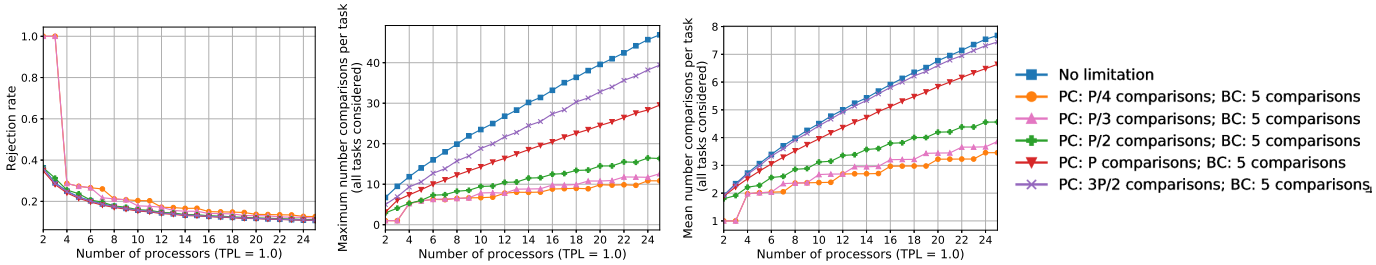
As the PB approach makes use of two task copies, there are system overheads, which are assessed in this section. Fig. 7 also compare the PB approach with BC deallocation to scheduling of only primary copies (using FFSS SbS), i.e. to the scheduling without fault tolerance.

Even if the BC deallocation is performed when a primary copy finishes, fault tolerant systems based on the PB approach and having only a few processors have higher rejection rate and higher number of comparisons when compared to systems not providing the fault tolerance. The ES of fault tolerant systems achieves slightly better results in terms of rejection rate than scheduling of only primary copies because the latter makes use of the FFSS SbS, which chooses the first found solution and not necessarily the earliest one. The more processors, the wider the gap in the number of comparisons (because there are more possibilities to test for the PB approach) and the narrower the gap in the rejection rate.

C. Comparison with the Optimal Solution

We compare our new proposed processor scheduling policy (FFSS SbS) in terms of the rejection rate to the optimal results provided by CPLEX solver, which explored all possible solutions and chose the one minimising the number of rejected tasks. Fig. 7a shows that the rejection rate of the FFSS SbS is approximately higher about 5% than the optimal solution and the algorithm using the FFSS SbS is 2-competitive. This represents a good result taking into account that our proposed technique chooses the first found solution.

The explanation of the difference between the optimal solution from CPLEX solver and the ES is as follows. At time t, the algorithm using the ES deallocates backup copies (if possible) and then it schedules new tasks one by one. The ES tests all



(a) Rejection rate (b) Max. # of comparisons (c) Mean # of comparisons

Figure 8: Method of limitation on the number of comparisons for the PB approach with BC deallocation as a function of the number of processors (FFSS SbS, $TPL = 1.0$)

processors for a current task in order to provide a solution, where the primary copies are scheduled as soon as possible and the backup ones are placed as late as possible. On the contrary, the CPLEX solver tests all schedules at the same time knowing all tasks available at time t, i.e. new task arrivals and backup copies, which can be deallocated. It means that primary copies are not necessarily scheduled as soon as possible and backup copies as late as possible.

D. Limitation on the Number of Comparisons

In this section, we focus on the limitation on the number of comparisons as described in Section IV-A.

The presented results are related to the PB approach with BC deallocation and similar results were obtained for the PB approach with BC deallocation and BC overloading. We consider the FFSS SbS and $TPL = 1.0$. Fig. 8 depict the rejection rate, the maximum and the mean numbers of comparisons as a function of the number of processors. The tested value for backup copies is always set at 5 comparisons and the ones for primary copies are: $P/4$, $P/3$, $P/2$, P and $3P/2$ comparisons, where P is the number of processors.

Fig. 8a representing the rejection rate shows that there is almost no difference if more than $P/2$ comparisons for primary copies are authorised. For systems with 2 and 3 processors and less than $P/2$ comparisons for primary copies, the rejection rate is 100% because there are not enough comparisons authorised to schedule a task. As for the maximum and mean numbers of comparisons represented respectively in Fig. 8b and 8c, the limitation on the number of comparisons significantly reduces their values as expected.

To find a trade-off between the rejection rate and the algorithm run-time, Fig. 11a plots improvements in the rejection rate and the maximum and mean numbers of comparisons for 14-processor system[5]. The values of studied metrics are compared to the PB approach without proposed enhancing method(s) and the higher improvement in %, the better the method.

It can be noticed that, if $P/2$ comparisons for PC is chosen, the rejection rate is deteriorated only by 1.5% when compared to the PB approach without this technique and the maximum and mean numbers of comparisons are respectively reduced by 62% and 34%.

E. Restricted Scheduling Windows

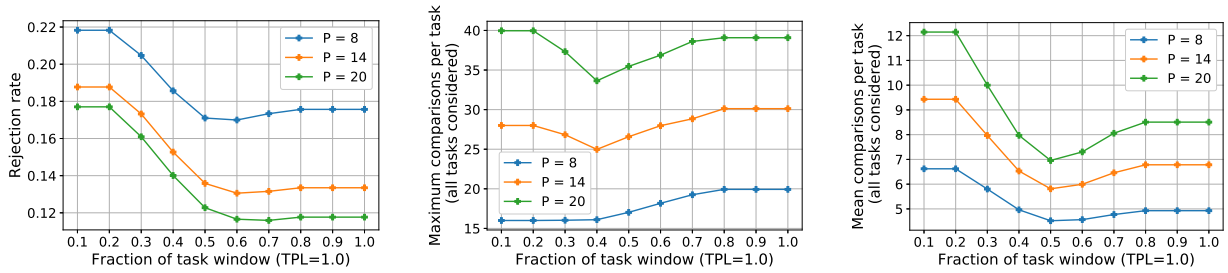
In this section, we analyse the method of restricted scheduling windows. We consider the FFSS SbS, TPL set at 1.0 and the PB approach with BC deallocation. Once again, similar results are obtained for the PB approach with BC deallocation and BC overloading. As an example, we show results respectively conducted for 8, 14 and 20 processors.

Fig. 9 depict the rejection rate, the maximum and the mean numbers of comparisons as a function of the fraction of task window. It can be seen that represented curves remain constant from $f = 0.1$ to $f = 0.2$ and from $f = 0.8$ to $f = 1.0$. These constant values are due to minimal considered ratio of computation times to task window in our experiment framework, which is $2c_i \leqslant d_i \leqslant 5c_i$ and thus $c_i/d_{max,i} = 1/5$. Furthermore, the trend for a given metric is similar no matter the number of processors. In conformity with the results depicted in Fig. 7, the more processors, the lower the rejection rate and the more comparisons.

Fig. 9a representing the rejection rate shows that, if the fraction f drops below 0.5, the rejection rate climbs because the scheduling windows become too restrictive. Therefore, we focus on f between 0.5 and 1 and observe a minimum for $f = 0.6$.

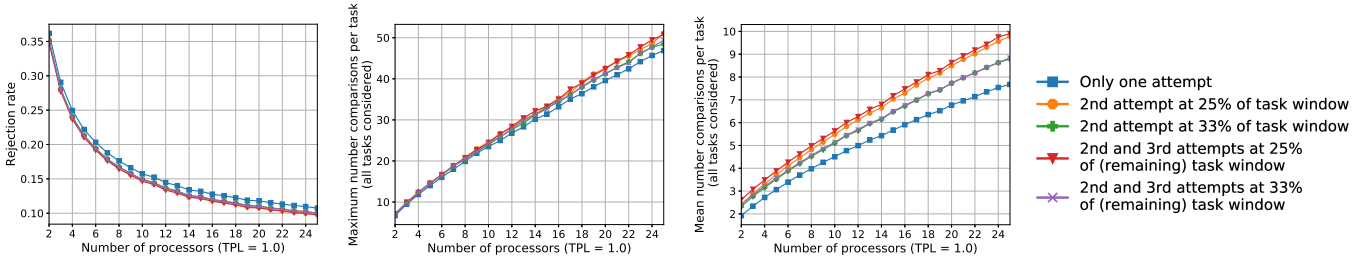
The algorithm run-time is depicted in Fig. 9b and 9c showing the maximum and mean numbers of comparisons, respectively. When zeroing in on f between 0.5 and 1, the evolution of the maximum number of comparisons grows with the fraction f of task window due to more possibilities to test. We notice that, if the restricted scheduling windows are fixed at $f = 0.5$, the maximum number of comparisons is reduced by 11% compared to the value for $f = 1$. As for the mean number of

[5]Fig. 7a shows that the rejection rate as a function of the number of processors does not considerably vary when the number of processors is greater than 12. Thus, a 14-processor system is taken as an example.



(a) Rejection rate (b) Maximum # of comparisons (c) Mean # of comparisons

Figure 9: Experimental results for the PB approach with BC deallocation using restricted scheduling windows as a function of the fraction of task window (FFSS SbS, $TPL = 1.0$)



(a) Rejection rate (b) Maximum number of comparisons (c) Mean number of comparisons

Figure 10: Method of several scheduling attempts for the PB approach with BC deallocation as a function of the number of processors (FFSS SbS, $TPL = 1.0$)

comparisons, it is reduced by about 17% when compared to values for $f = 1$. The value of the mean number of comparisons is up to 5.5 times lower compared with the maximum number of comparisons, which demonstrates that the FFSS SbS does not need to scour all processors all the time to schedule a task.

To sum up, the method of restricted scheduling windows diminishes the algorithm run-time, measured by means of the number of comparisons, without notably worsening the system performances, such as the rejection rate. Fig. 11b represents improvements in the rejection rate and the maximum and mean numbers of comparisons for different values of f. It is shown that the reasonable trade-off between the rejection rate and the number of comparisons is obtained for $f = 0.5$ or $f = 0.6$.

F. Several Scheduling Attempts

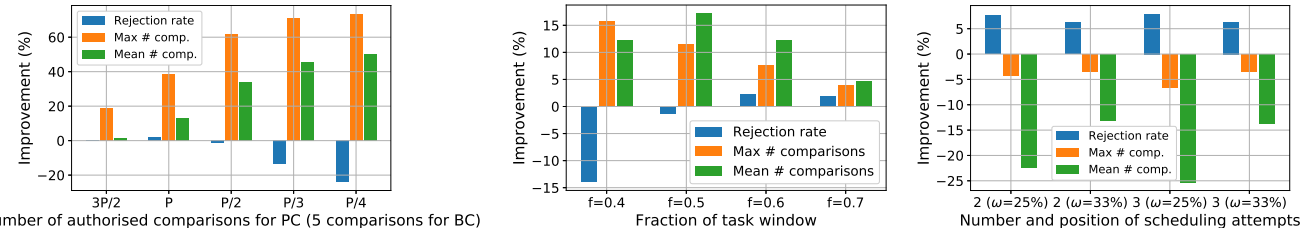
To evaluate performances of several scheduling attempts, we consider the FFSS SbS and $TPL = 1.0$. Based on mean values of simulation parameters from Table II, the value of the percentage of slack within the task window ps, as defined in Equation 3, is 43%, which means that there is a high chance of successful scheduling a task later than at its task arrival. We present results for the PB approach with BC deallocation but it should be noticed that results remain valid for the PB approach with BC deallocation and BC overloading as well. Fig. 10 depict the rejection rate, the maximum and the mean numbers of comparisons as a function of the number of processors when $\omega = 25\%$ and $\omega = 33\%$.

As it can be seen in Fig. 10a, two or three scheduling attempts are always beneficial and the decrease in the rejection rate is about 6% or 7%. The maximum and mean numbers of comparisons, depicted in Fig. 10b and 10c respectively, are worsened when compared to the algorithm with only one scheduling attempt because each new attempt requires additional comparisons. Fig. 11c showing the improvement for 14-processor system sums up the results. It can be noticed, that it does not worth trying more than two attempts for there is hardly any improvement in the rejection rate and the number of comparisons is higher. The reasonable trade-off between the rejection rate and the number of comparisons is the use of two scheduling attempts at 33% of task window.

G. Combination of Enhancing Methods

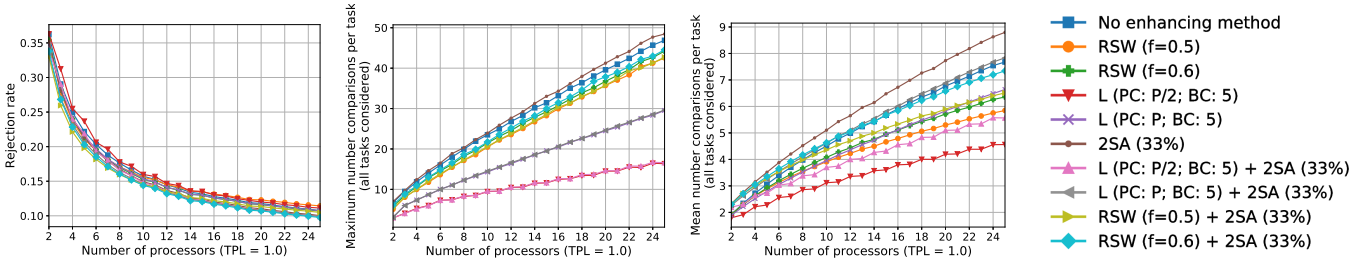
We remind the reader that our aim is to significantly reduce the number of comparisons without worsening the rejection rate. Consequently, we analyse the aforementioned methods (and their combinations) with parameters achieving the best performances based on the results from Sections VI-D, VI-E and VI-F and summarised for 14-processor system in Fig. 11. The chosen methods (with their abbreviations in square brackets) use the FFSS SbS, $TPL = 1.0$ and are as follows:

- Limitation on the number of comparisons (PC: $P/2$ comparisons; BC: 5 comparisons) [L (PC: $P/2$; BC: 5)]



(a) Limitation on the number of comparisons (b) Restricted scheduling windows (c) Several scheduling attempts

Figure 11: Improvements to 14-processor system compared to the PB approach without proposed enhancing methods (PB approach with BC deallocation, FFSS SbS, $TPL = 1.0$)



(a) Rejection rate (b) Max. # of comparisons (c) Mean # of comparisons

Figure 12: Comparison of different methods for the PB approach with BC deallocation as a function of the number of processors (FFSS SbS, $TPL = 1.0$)

- Limitation on the number of comparisons (PC: P comparisons; BC: 5 comparisons) [L (PC: P; BC: 5)]
- Restricted scheduling windows ($f = 0.5$) [RSW ($f = 0.5$)]
- Restricted scheduling windows ($f = 0.6$) [RSW ($f = 0.6$)]
- Two scheduling attempts at 33% [2SA (33%)]
- Limitation on the number of comparisons (PC: $P/2$ comparisons; BC: 5 comparisons) and two scheduling attempts at 33% [L (PC: $P/2$; BC: 5) + 2SA (33%)]
- Limitation on the number of comparisons (PC: P comparisons; BC: 5 comparisons) and two scheduling attempts at 33% [L (PC: P; BC: 5) + 2SA (33%)]
- Restricted scheduling windows ($f = 0.5$) and two scheduling attempts at 33% [RSW ($f = 0.5$) + 2SA (33%)]
- Restricted scheduling windows ($f = 0.6$) and two scheduling attempts at 33% [RSW ($f = 0.6$) + 2SA (33%)]

These methods are then compared to the baseline method, i.e. the FFSS SbS without any proposed enhancing techniques. The obtained results are depicted in Fig. 12 respectively showing the rejection rate, the maximum and the mean numbers of comparisons as a function of the number of processors when $TPL = 1.0$. Although only the results for the PB approach with BC deallocation are represented, the PB approach with BC deallocation and BC overloading achieves similar performances.

From Fig. 12a, it can be seen that the lowest rejection rate is attained by two scheduling attempts ($\omega = 33\%$) with the limitation on the number of comparisons (PC: P comparisons; BC: 5 comparisons) or with the restricted scheduling windows ($f = 0.6$).

Fig. 12b illustrates the reduction in the maximum number of comparisons when the method of limitation on the number of comparisons is put into practice. The mean number of comparisons, represented in Fig. 12c, is diminished for all methods except when the method of two scheduling attempts is separately put into practice.

To facilitate a comparison among studied techniques, improvements (compared to the PB approach without described techniques) in the rejection rate and in the maximum and mean number of comparisons are depicted in Fig. 13. Due to space constraints, only the results for 14-processor system are plotted but similar improvements were obtain for systems having different number of processors. All methods (except when the technique of two scheduling attempts is put into practice separately or in conjunction with the restricted scheduling windows) reduce the number of comparisons and all methods (except the restricted scheduling windows ($f = 0.5$) and the limitation on the number of comparisons (PC: $P/2$ comparisons; BC: 5 comparisons)) lessen the rejection rate. The best methods to reduce both, the rejection rate and number of comparisons, and

no matter whether the PB approach with BC deallocation and with or without BC overloading is used, are as follows: (i) the limitation on the number of comparisons (PC: $P/2$ comparisons; BC: 5 comparisons) and two scheduling attempts at 33%, and (ii) the limitation on the number of comparisons (PC: P comparisons; BC: 5 comparisons). Moreover, we also compare these two methods to the approach with the exhaustive search (ES) presented in [5] as it is the method which provides the lowest rejection rate (see Section VI-A). Whereas the rejection rate is respectively deteriorated by -4.6% and -4.0%, the improvement in the maximum (77% and 64%, respectively) and mean (84% and 79%, respectively) numbers of comparisons are significant and interesting for embedded systems.

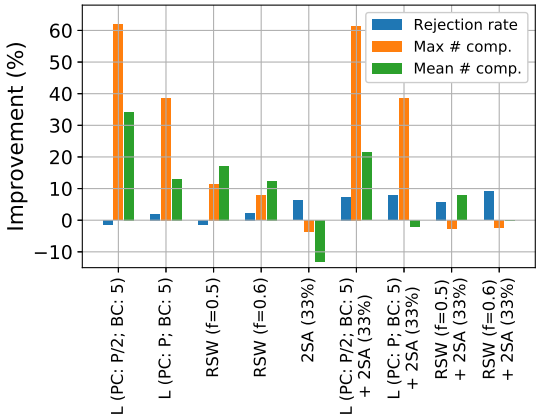


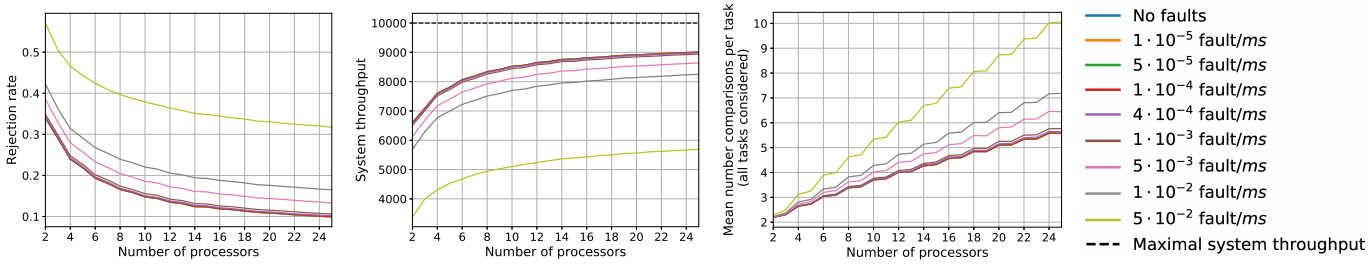
Figure 13: Improvements to 14-processor system compared to the PB approach without proposed enhancing methods (PB approach with BC deallocation, FFSS SbS, $TPL = 1.0$)

H. Fault Injection

This section evaluates the fault tolerance performances of the PB approach with BC deallocation (FFSS SbS) with limitation on the number of comparisons (PC: $P/2$ comparisons; BC: 5 comparisons) and two scheduling attempts at 33%, which is the best combination of the enhancing methods studied previously. It should be noticed that the conclusions made for this case are also valid for other techniques with and without BC deallocation even if they are not detailed in this report.

Fig. 14 respectively depict the rejection rate, the system throughput and the mean number of comparisons as a function of the number of processors at different fault rates. As expected, the higher the fault rate, the higher the rejection rate, the lower the system throughput and the higher the mean number of comparisons. The maximum number of comparisons remains almost unchanged compared to the fault-free simulations (figure not presented in the report). The rejection rate accounts for the schedulability as described in Section V-B, which means that both primary and backup copies are successfully scheduled. Nevertheless, it may happen that a backup copy is impacted by fault too. In this case, such a task does not contribute to the system throughput because it was not correctly executed.

We conclude that algorithm performances do not significantly change up to $1 \cdot 10^{-3}$ fault/ms. This fault rate is higher than the estimated processor fault rate in standard conditions ($2.3 \cdot 10^{-9}$ fault/ms [28]) and even higher than the worst estimated fault rate in a harsh environment ($1 \cdot 10^{-5}$ fault/ms [29]).



(a) Rejection rate (b) System throughput (c) Mean # of comparisons

Figure 14: PB approach with fault injection and with BC deallocation (FFSS SbS) with limitation on the number of comparisons (PC: $P/2$ comparisons; BC: 5 comparisons) and two scheduling attempts at 33%

VII. Discussion

The results show different aspects of fault tolerant scheduling of aperiodic tasks based on the PB approach. The main outcomes are discussed in this section.

First, our new proposed processor scheduling policy was compared to two already published ones: the *exhaustive search* (ES) [5], the *first found solution search - processor by processor* (FFSS PbP) [10], and the *first found solution search - slot by slot* (FFSS SbS). On the one hand, it was found that the ES achieves the lowest rejection rate compared to the two other searches but it has the highest values for the maximum and mean numbers of comparisons. On the other hand, the FFSS SbS performs better than the FFSS PbP. While its rejection rate is higher about 12% compared to the ES (14 processors), the maximum number of comparisons is significantly lower than the one for the FFSS PbP (about 30% for 14-processor system) and the ES (about 41% for 14-processor system). Moreover, the mean number of comparisons is only slightly dependent on the number of processors, which is advantageous to systems with many resources.

The overhead of the algorithm based on the PB approach was also assessed. Since this approach reserves slots for its primary and backup copies (even if the BC deallocation is put into practice), the higher the number of processors, the more comparisons to find slots for both copies and consequently the wider the gap in the number of comparisons between fault tolerant and non fault tolerant systems. It was also shown that the more processors, the narrower the gap in the rejection rate between fault tolerant system using the PB approach and non-fault tolerant one.

Next, the method of *limitation on the number of comparisons* was presented. This very simple method provides interesting results. For example, when the threshold for primary copies is set at $P/2$ comparisons (where P is the number of processors) and the one for backup copies is fixed at 5 comparisons, the maximum and mean numbers of comparisons are respectively cut down by 62% and 34%, whereas the rejection rate is higher by only 1.5% compared to the approach without this technique.

Another method aiming at reducing the algorithm run-time is the technique of *restricted scheduling windows*. It diminishes the algorithm run-time, measured again by means of the number of comparisons, without worsening the system performances, such as the rejection rate. The reasonable trade-off between the rejection rate and the number of comparisons is obtained for the fraction of task window equal to 0.5 or 0.6.

The method of *several scheduling attempts* focuses on the reduction in the rejection rate. The results showed that it is useless to carry out more than two scheduling attempts because the rejection rate is not notably better and the number of comparisons increases too much. The reasonable trade-off between the rejection rate and the number of comparisons is achieved for two scheduling attempts at 33% of task window. In such a case, the rejection rate is reduced by 6.2%.

Then, we analysed combinations of aforementioned methods. It was found that almost all proposed methods diminish the number of comparisons and lessen the rejection rate. The best methods to reduce both, the rejection rate and the number of comparisons, are (i) the limitation on the number of comparisons (PC: $P/2$ comparisons; BC: 5 comparisons) combined with two scheduling attempts at 33%, and (ii) the limitation on the number of comparisons (PC: P comparisons; BC: 5 comparisons). The algorithm run-time of the former technique is reduced by 23% (mean value) and 67% (maximum value) and its rejection rate is decreased by 4% compared to the PB approach without any enhancing method.

Last but not least, the results showed that fault rates up to $1 \cdot 10^{-3}$ fault/ms have a minimal impact on algorithm performances. This value is higher than the estimated fault rate in standard conditions ($2.3 \cdot 10^{-9}$ fault/ms [28]) and also in severe conditions ($1 \cdot 10^{-5}$ fault/ms [29]). Therefore, our algorithm can perform well in a harsh environment.

VIII. Conclusion

This report is concerned with the online fault tolerant primary/backup (PB) approach-based scheduling of aperiodic tasks on multiprocessor embedded systems dealing with hard real-time tasks. It describes and analyses an algorithm with several enhancing techniques to reduce the algorithm run-time without worsening system performances.

First, our new proposed processor scheduling policy was compared to two already published ones. Although the *exhaustive search* (ES) [5] has lower rejection rate than the *first found solution search - processor by processor* (FFSS PbP) [10], and the *first found solution search - slot by slot* (FFSS SbS), its number of comparisons, accounting for the algorithm run-time, is significantly higher.

Then, our three devised techniques (*limitation on the number of comparisons*, *restricted scheduling windows* and *several scheduling attempts*) and their combinations are analysed in term of performances. The results show that the best methods, which reduce both the rejection rate and the number of comparisons, are (i) the limitation on the number of comparisons combined with two scheduling attempts at 33%, and (ii) the limitation on the number of comparisons.

Finally, it was found that the studied algorithm performs well also in a harsh environment.

Regarding the future work, we intend to design a new fault tolerant online algorithm for real-time embedded systems, which will take into account energy constraints.

REFERENCES

[1] A. Naithani, S. Eyerman, and L. Eeckhout, "Reliability-Aware Scheduling on Heterogeneous Multicore Processors," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 397–408.

[2] E. Dubrova, *Fault-Tolerant Design*. Springer, 2013.

[3] T. Herault and Y. Robert, *Fault-Tolerance Techniques for High-Performance Computing*, 1st ed. Springer Publishing Company, Incorporated, 2015.

[4] I. Koren and C. M. Krishna, *Fault-Tolerant Systems*. Morgan Kaufmann Publishers, Elsevier, 2007.

[5] S. Ghosh, R. Melhem, and D. Mosse, "Fault-Tolerance Through Scheduling of Aperiodic Tasks in Hard Real-Time Multiprocessor Systems," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 3, March 1997, pp. 272–284.

[6] X. Zhu, J. Wang, J. Wang, and X. Qin, "Analysis and Design of Fault-Tolerant Scheduling for Real-Time Tasks on Earth-Observation Satellites," in *43rd International Conference on Parallel Processing*, 2014, pp. 491–500.

[7] A. Kumar, S. Panda, S. K. Pani, V. Baghel, and A. Panda, "Aco and Ga Based Fault-Tolerant Scheduling of Real-Time Tasks on Multiprocessor Systems - A Comparative Study," in *IEEE 8th International Conference on Intelligent Systems and Control (ISCO)*, 2014, pp. 120–126.

[8] X. Zhu, J. Wang, H. Guo, D. Zhu, L. T. Yang, and L. Liu, "Fault-Tolerant Scheduling for Real-Time Scientific Workflows with Elastic Resource Provisioning in Virtualized Clouds," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 12, 2016, pp. 3501–3517.

[9] Y. Guo, D. Zhu, H. Aydin, J.-J. Han, and L. T. Yang, "Exploiting Primary/Backup Mechanism for Energy Efficiency in Dependable Real-Time Systems," vol. 78, 2017, pp. 68–80.

[10] M. Naedele, "Fault-Tolerant Real-Time Scheduling under Execution Time Constraints," in *Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA)*, 1999, pp. 392–395.

[11] T. Tsuchiya, Y. Kakuda, and T. Kikuno, "A New Fault-Tolerant Scheduling Technique for Real-Time Multiprocessor Systems," in *Proceedings Second International Workshop on Real-Time Computing Systems and Applications*, 1995, pp. 197–202.

[12] Q. Zheng, B. Veeravalli, and C.-K. Tham, "On the Design of Fault-Tolerant Scheduling Strategies Using Primary-Backup Approach for Computational Grids with Low Replication Costs," in *IEEE Transactions on Computers*, vol. 58, no. 3, 2009, pp. 380–393.

[13] H. Kim, S. Lee, and B.-S. Jeong, "An improved feasible shortest path real-time fault-tolerant scheduling algorithm," in *Proceedings Seventh International Conference on Real-Time Computing Systems and Applications*, Dec 2000, pp. 363–367.

[14] A. Amin, R. Ammar, and A. El Dessouly, "Scheduling real time parallel structures on cluster computing with possible processor failures," in *Proceedings. ISCC 2004. Ninth International Symposium on Computers And Communications (IEEE Cat. No.04TH8769)*, vol. 1, July 2004, pp. 62–67.

[15] H. Beitollahi, S. G. Miremadi, and G. Deconinck, "Fault-Tolerant Earliest-Deadline-First Scheduling Algorithm," in *2007 IEEE International Parallel and Distributed Processing Symposium*, March 2007, pp. 1–6.

[16] R. Sridharan and R. Mahapatra, "Analysis of Real Time Embedded Applications in the Presence of a Stochastic Fault Model," in *20th International Conference on VLSI Design held jointly with 6th International Conference on Embedded Systems (VLSID'07)*, Jan 2007, pp. 83–88.

[17] X. Zhu, X. Qin, and M. Qiu, "QoS-Aware Fault-Tolerant Scheduling for Real-Time Tasks on Heterogeneous Clusters," in *IEEE Transactions on Computers*, vol. 60, no. 6, June 2011, pp. 800–812.

[18] A. Syed and G. Fohler, "Efficient offline scheduling of task-sets with complex constraints on large distributed time-triggered systems," in *Real-Time Systems*, vol. 55, no. 2, Apr 2019, pp. 209–247.

[19] S. Wang, K. Li, J. Mei, G. Xiao, and K. Li, "A Reliability-aware Task Scheduling Algorithm Based on Replication on Heterogeneous Computing Systems," in *Journal of Grid Computing*, vol. 15, no. 1, 03 2017, pp. 23–39.

[20] A. Das, A. Kumar, B. Veeravalli, C. Bolchini, and A. Miele, "Combined DVFS and Mapping Exploration for Lifetime and Soft-Error Susceptibility improvement in MPSoCs," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2014, pp. 1–6.

[21] A. Benoit, M. Hakem, and Y. Robert, "Fault Tolerant Scheduling of Precedence Task Graphs on Heterogeneous Platforms," in *IEEE International Symposium on Parallel and Distributed Processing*, 2008, pp. 1–8.

[22] A. Girault, H. Kalla, M. Sighireanu, and Y. Sorel, "An Algorithm for Automatically Obtaining Distributed and Fault-Tolerant Static Schedules," in *International Conference on Dependable Systems and Networks*, 2003.

[23] P. Dobiáš, E. Casseau, and O. Sinnen, "Comparison of Different Methods Making Use of Backup Copies for Fault-Tolerant Scheduling on Embedded Multiprocessor Systems," in *2018 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, Oct 2018, pp. 100–105.

[24] G. Manimaran and C. S. R. Murthy, "A Fault-Tolerant Dynamic Scheduling Algorithm for Multiprocessor Real-Time Systems and its Analysis," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 11, 1998, pp. 1137–1152.

[25] R. Al-Omari, A. K. Somani, and G. Manimaran, "Efficient Overloading Techniques for Primary-Backup Scheduling in Real-Time Systems," in *Journal of Parallel and Distributed Computing*, vol. 64, no. 5, 2004, pp. 629–648.

[26] P. Dobiáš, E. Casseau, and O. Sinnen, "Restricted Scheduling Windows for Dynamic Fault-Tolerant Primary/Backup Approach-Based Scheduling on Embedded Systems," in *Proceedings of the 21st International Workshop on Software and Compilers for Embedded Systems*, ser. SCOPES '18. New York, NY, USA: ACM, 2018, pp. 27–30.

[27] X. Qin, H. Jiang, and D. R. Swanson, "An efficient fault-tolerant scheduling algorithm for real-time tasks with precedence constraints in heterogeneous systems," in *Proceedings International Conference on Parallel Processing*, Aug 2002, pp. 360–368.

[28] S. Du, E. Zio, and R. Kang, "A New Analytical Approach for Interval Availability Analysis of Markov Repairable Systems," in *IEEE Transactions on Reliability*, vol. 67, no. 1, March 2018, pp. 118–128.

[29] R. M. Pathan, "Real-Time Scheduling Algorithm for Safety-Critical Systems on Faulty Multicore Environments," in *Real-Time Systems*, vol. 53, no. 1, 2017, pp. 45–81.