Comparison of Different Methods Making Use of Backup Copies for Fault-Tolerant Scheduling on Embedded Multiprocessor Systems

Petr Dobiáš, Emmanuel Casseau, Oliver Sinnen

Comparison of Different Methods Making Use of Backup Copies for Fault-Tolerant Scheduling on Embedded Multiprocessor Systems

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

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

Abstract

As transistors scale down, systems are more vulnerable to faults. Their reliability consequently becomes the main concern, especially in safety-critical applications such as automotive sector, aeronautics or nuclear plants. Many methods have already been introduced to conceive fault-tolerant systems and therefore improve the reliability. Nevertheless, several of them are not suitable for real-time embedded systems since they incur significant overheads, other methods may be less intrusive but at the cost of being too specific to a dedicated system. The aim of this paper is to analyse a method making use of two task copies when on-line scheduling tasks on multiprocessor systems. This method can guarantee the system reliability without causing too much overhead and requiring any special hardware components. In addition, it remains general and thus applicable to large amount of systems. Last but not least, this paper studies two techniques of processor allocation policies: the exhaustive search and the first found solution search. It is shown that the exhaustive search is not necessary for efficient fault-tolerant scheduling and that the latter search significantly reduces the computation complexity, which is interesting for embedded systems.

Index Terms

Embedded real-time systems, Fault tolerance, Multiprocessor systems, On-line scheduling, Primary/backup approach

I. Introduction

Each system component is liable to fail and it will cease running correctly sooner or later. As a result, the system can exhibit a malfunction. There are applications where a system failure can have catastrophic consequences like advanced driver-assistance systems, air traffic control or medical equipment. In order to solve this problem, systems should be fault-tolerant, i.e. more robust to be able to properly work even if faults occur. The fault-tolerant design is thus a topical issue, especially at the present time when transistors scale down to fulfil stern constraints on energy-consumption and area, which goes hand in hand with higher vulnerability to fault occurrence. It is therefore necessary to ensure the system reliability, in particular for safety-critical applications.

The main aim of this paper is to compare different techniques intended for the primary/backup approach, which is an approach to fault-tolerant scheduling on embedded multiprocessor systems making use of two tasks copies: primary and backup ones [1]. The primary/backup approach is a commonly used technique when conceiving fault-tolerant systems owing to its easy application and minimal system overheads. Several additional enhancements [1]–[3] to this approach have been already presented but few studies dealing with overall comparisons have been carried out.

The remainder of this paper is organised as follows. The related work is summarised in Section II. Section III presents our assumptions and it describes the scheduling model and tested techniques. The experiment framework is introduced in Section IV and results are then analysed in Section V.

II. Related Work

Since systems are more vulnerable to faults and the reliability becomes the main concern, Naithani et al. [4] emphasized the necessity to consider the reliability during scheduling. They showed that it is better to make use of reliability-aware scheduling rather than performance-optimised scheduling. On average, although the reliability-aware scheduling degrades performance by 6%, it improves the system reliability by 25.4% compared to the performance-optimised scheduling.

The next question is which possibilities are available to conceive fault-tolerant systems.

The first solution is to make the system more robust by adding additional components but, regarding that extra logic requires changes in hardware, this possibility is not applicable to already existing systems.

This work was funded by the Reliasic project granted by the CominLabs Excellence Laboratory (ANR-10-LABX-07-01), French ”Investissements d’Avenir” program.

The second solution is to modify the code in order to verify its correct execution. Some error detection code can be inserted in high level code [5] or in instruction-level one [6]. Goloubeva et al. [7] introduced additional executable assertions to check the correct execution of the program control flow. Their technique mainly aims at safety-critical applications. However, depending on the application and program, the obtained overheads are considerable: memory ones (minimum: 124%, average: 283%, maximum: 630%) and performance ones (minimum: 107%, average: 185%, maximum: 426%).

A hybrid solution putting the first and second solutions together was proposed by Bernardi et al. [8]. Their approach combined software-based techniques with an Infrastructure IP to detect transient faults in processor-based SoCs and it was tested using several benchmarks. The results showed significant overheads in execution time (minimum: 78%, average: 126%, maximum: 209%), in code size (minimum: 68%, average: 162%, maximum: 270%) and in data size (minimum: 102%, average: 107%, maximum: 113%).

The third possibility is to take advantage of the redundancy on multiprocessor systems, which can be effectively used to design fault-tolerant systems. While the redundancy in space, such as N-modular redundancy [9] incurs system overheads even during normal operation, the redundancy in time does not cause any of them in fault-free environment. An example of such redundancy is a primary/backup approach [1], which makes use of two task copies: primary and backup ones and which is commonly used for its high-reliability. It can be found in satellites [10] or employed in the virtualised cloud [11]. Furthermore, its uncomplicated principle makes it compatible with different implementations, such as the genetic algorithm [12], [13].

III. Assumptions and Scheduling Model

The studied system consists of P identical[1] processors sharing the same memory[2]. All tasks, characterised by their arrival time a, computation time c and deadline d, are considered to be aperiodic and independent. They are on-line and immediately mapped and scheduled on the system without preemption. We assume that only one processor failure (transient or permanent) can occur at any instant of time and that a fault detection mechanism[3] exists to promptly inform about a fault occurrence.

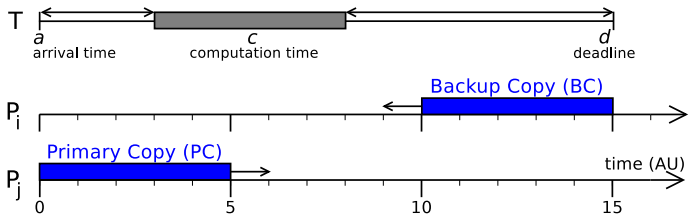


Figure 1: Principle of the primary/backup approach

The principle of the primary/backup (PB) approach [1] is that, when a task arrives, the system creates two identical task copies. A time interval on which a task copy is placed is called a *slot*. The first task copy is called a *primary copy* and denoted PC, the second one is a *backup copy* labelled BC. These copies are then scheduled on two different processors within the task window[4] without overlapping each other, as depicted in Fig. 1. The aim is to keep system overheads down when no fault occurs. Thus, if the primary copy of a task cannot provide correct results due to a fault, the scheduled backup copy is run and its results are taken into account. In general, when scheduling a task, the algorithm first places its primary copy as soon as possible and then its backup copy as late as possible, as summarised in Algorithm 1. This strategy avoids idle processors just after the task arrival and possible high processor load later.

In order to increase the system schedulability and minimise resources utilisation, i.e. system overheads, two enhancing techniques exist: the *backup deallocation* and the *backup overloading* [1]. The former technique deallocates the backup copy BC_i, if the primary copy PC_i was correctly executed. The advantage of this technique is not to unnecessarily occupy resources and thus free the occupied slot for new arriving tasks. The latter technique overloads several backup copies, except ones having their primary copies placed on the same processor, and therefore saves up processor resources.

A. Processor Allocation Policies

The subsequent question is how a search for PC and BC slots are carried out. In this paper, we analyse two possibilities for processor allocation policies: the *exhaustive search* (ES) and the *first found solution search* (FFSS). The exhaustive search checks all processors for available slots and chooses the best solution, i.e. the one having its primary copy scheduled as soon as possible and the backup copy as late as possible. In contrast, the first found solution search assesses processors in turn and

[1]To simplify, a system with homogeneous processors is considered in this paper. Nonetheless, the studied model can be easily extended to a system consisting of heterogeneous processors, such as in [14].

[2]In the case of distributed memory, it is necessary to take delays of data transfers into account but the principle of the method remains the same.

[3]A fault can be detected by acceptance tests, such as timing, coding, reasonableness or structural checks [9].

[4]A *task window* tw is defined as the difference between the deadline d and the arrival time a.

Algorithm 1 Primary/backup approach

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

```
if new task T_i arrives then
    Map and schedule PC_i (ES or FFSS policy)
    Map and schedule BC_i (ES or FFSS policy)
    if PC_i and BC_i slots exist then
        Commit the task T_i
    else
        Reject the task T_i
    end if
end if
if fault detection mechanism reports a fault then
    Do not take into consideration the result from PC_i
    Wait for the result from BC_i
else
    Consider the result from PC_i
end if
```

it looks only for the first suitable slot for PC and then for the first BC one no matter their positions within the task window. In this case, the best solution can be consequently missed.

Moreover, in order to avoid that the first processor has higher charge than other system processors, the FFSS starts its search for a PC slot on the processor following the processor on which the primary copy of previous task was successfully scheduled. The search then continues in ascending order until a free slot is found or no more processor is available [3]. Then, if the primary copy of the new task is found on processor P_i, a search for a BC slot can be conducted. It starts on processor P_{i-1} and it continues in decreasing order.

B. Active Primary/Backup Approach

Until now, the *passive* primary/backup approach was considered, i.e. that primary and backup copies of the same task cannot overlap each other on two different processors. Nevertheless, this approach may be too restrictive for some real-time systems since the deadline may be earlier than two times the computation time and, therefore, the *active* primary/backup approach should be considered [2]. On the one hand, it allows the primary and backup copies to overlap each other in space and thus helps to schedule tasks with tight deadlines. On the other hand, this approach gives rise to system overheads because the system entirely or partially executes the backup copy (during the execution of the corresponding primary copy). Besides, the active approach adds more schedulability constraints: backup copies scheduled by means of this method cannot overload other backup copies and cannot be overloaded as they always need to be executed (in total or in part).

IV. Experiments Framework

To evaluate the system performance using aforementioned techniques, simulations were carried out based on the parameters summarised in Table I. Each simulation contains 10 000 tasks and the results present the average of 100 simulations. We assume that there is no fault occurrence and all backup copies can be consequently deallocated once their respective primary copies finish.

In order to generate task arrival times, we define the *Targeted Processor Load* (TPL). If this parameter is equal to 1.0, task attribute a is generated so that every processor is theoretically charged all the time at 100%. In practice, the processor load can be less than 100% when tasks are rejected due to missing free slots.

Table I: Standard simulation parameters

Parameter	Symbol	Distribution	Value(s)
Number of processors	P	-	2 – 25
Targeted processor load	TPL	-	0.5 – 1.0
Computation time	c	Uniform	1 – 20
Arrival time	a	Poisson	$\lambda = \frac{average\ execution\ time}{TPL \cdot P}$
Deadline	d	Uniform	$(a + 2c)$ – $(a + 5c)$

The performance of realised simulations were then assessed by means of the following criteria. The *rejection rate* represents the ratio of rejected tasks to all arriving tasks. The *processor load* is the effective system load taking into account the backup

deallocation and rejected tasks. This metric allows us to evaluate the resources utilisation: when it equals 100%, the scheduling efficiency is maximal. The *number of comparisons*, which is necessary to carry out before finally accepting or rejecting a task, accounts for the algorithm complexity. One comparison corresponds to evaluation whether a free slot is large enough to accommodate a task copy on a given processor.

V. Results

In this section, we first present the baseline results in order to observe the improvement of the backup deallocation technique presented in the subsequent section. Afterwards, we compare two processor allocation policies and evaluate the use of the active PB approach.

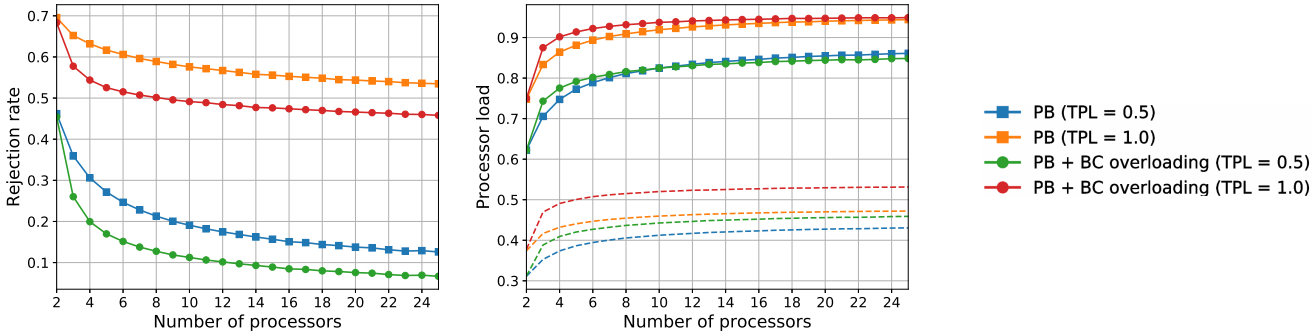
A. Baseline Results

Fig. 2 represent the rejection rate and the processor load as a function of the number of processors for the passive PB approach without any additional technique and the passive PB approach with BC overloading. The targeted processor load equals 0.5 and 1.0, respectively, and the chosen processor allocation policy is the first found solution search.

First of all, we note that the results of the rejection rate for the PB approach with BC overloading are better than the ones for the PB approach alone (up to 13%) since the implemented technique allows backup copies to overload each other, unless their primary copies are on the same processor, which saves up free slots that can be used for new arriving tasks. Regarding the processor load, both approaches reach similar values.

Then, it can be seen in Fig. 2a that the rejection rate decreases with the increasing number of processors. We remind the reader that the targeted processor load is set as a constant. Thus, according to the definition of the Poisson distribution parameter λ, when the number of processors increases, the parameter λ decreases, which implies that tasks have shorter interarrival time and arrive more often. Actually, the addition of processors brings more possibilities to find a suitable slot so that the processor load eventually decreases. Besides, the higher the targeted processor load, the higher the rejection rate because the system becomes more charged and consequently rejects more tasks.

Finally, the processor load is shown in Fig. 2b. The continuous lines account for the workload of the whole system, i.e. when the primary and backup copies are taken into account. The processor load increases when the number of processors rises. It can be noticed that, since the BC deallocation is not used, the system performs the same computation twice even though only one execution is necessary in the fault-free environment. The dashed lines in Fig. 2b stand for the processor load when backup copies are not considered. Actually, it is the effective processor load from the user's point of view. Even when the BC overloading is put into practice, the effective processor load is about 50%. Consequently, to improve the system performances, the technique of the BC deallocation should be introduced and analysed, which is the aim of the next section.



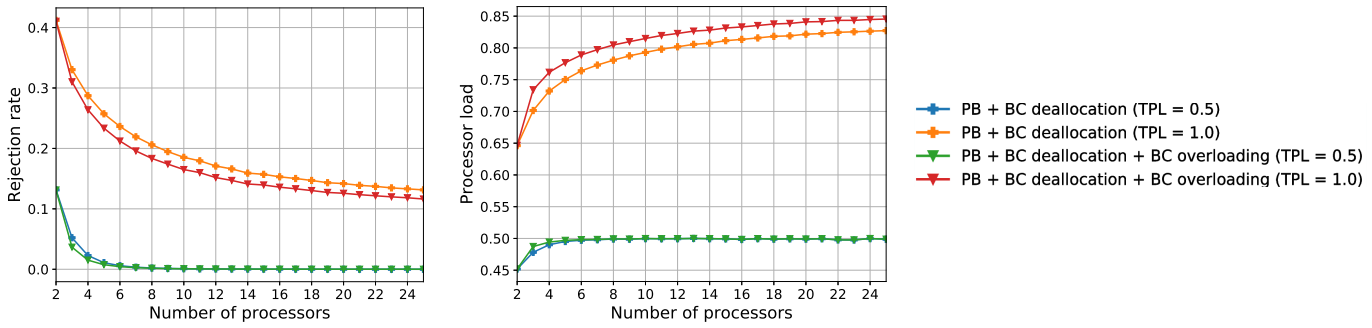
(a) Rejection rate (b) Processor load

Figure 2: Rejection rate and processor load for the passive PB approach alone and for the one with BC overloading as a function of the number of processors and TPL (first found solution search)

B. Merit of Backup Deallocation

To obtain comparable results to the previous ones, the algorithm makes use of the first found solution search and the TPL is fixed at 0.5 and 1.0, respectively. Fig. 3 represent the studied metrics as a function of the number of processors for the passive PB approach with BC deallocation only and for the passive PB approach with BC deallocation and BC overloading.

Foremost, it can be noticed that the PB approach with BC deallocation and BC overloading achieves better results than the PB approach with BC deallocation only, which means that the BC deallocation and the BC overloading can be used fruitfully together.



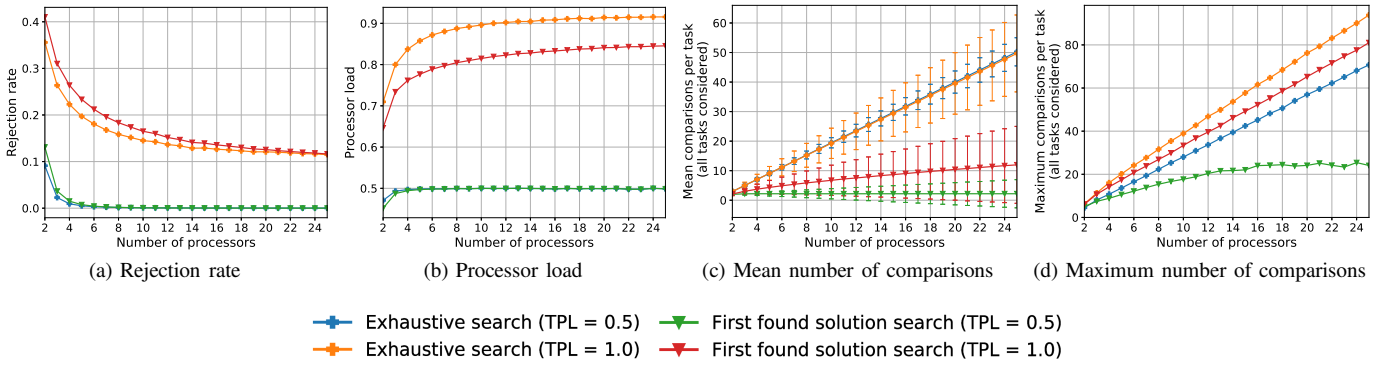
(a) Rejection rate (b) Processor load

Figure 3: Rejection rate and processor load for the passive PB approach with BC deallocation and for the one with BC deallocation and BC overloading as a function of the number of processors and TPL (first found solution search)

Fig. 3a depicts that the rejection rate is significantly improved compared to Fig. 2 thanks to the BC deallocation. For example for 20-processor system and $TPL = 1.0$, the gain is about 75% no matter whether the technique of BC overloading is implemented or not.

The curves of processor load depicted in Fig. 3b account for the workload of the whole system and the workload of the primary copies only as well because all backup copies are deallocated and no fault occurs. It means that when the BC deallocation is put into practice, the system can accept twice as more tasks compared to the system without this technique. Moreover, when a system has higher number of processors, curves tend to the value of the targeted processor load (0.5 or 1.0) showing the effectiveness of the BC deallocation technique.

C. Comparison of Two Processor Allocation Policies



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

Figure 4: Experimental results for the passive PB approach with BC deallocation and BC overloading comparing two processor allocation policies as a function of the number of processors and TPL

In this section, we compare two processor allocation policies described in Section III-A: the *exhaustive search* (ES) and the *first found solution search* (FFSS). The TPL is still respectively set at 0.5 and 1.0 and the passive PB approach with BC deallocation and BC overloading is considered. Fig. 4 respectively depict the rejection rate, the processor load and the mean and maximum number of comparisons as a function of the number of processors.

Fig. 4a illustrates that the results of the rejection rate for two processor allocation policies are almost the same when $TPL = 0.5$ and that the ES achieves slightly better results than the FFSS when $TPL = 1.0$. Nevertheless, the difference between two techniques diminishes with the increasing number of processors and becomes negligible for 20-processor system.

The processor load represented in Fig. 4b shows that, when $TPL = 1.0$, the FFSS reaches lower values than the ES (approximatively about 10%). Taking into account that, the FFSS and the ES have similar rejection rate, the FFSS manages better the system workload, which can put off the ageing of processors.

The results of the mean number of comparisons are significantly improved and the ones of the maximum number of comparisons are also ameliorated. For instance for a 20-processor system, the mean number of comparisons drops by 94% and 74% when $TPL = 0.5$ and $TPL = 1.0$, respectively, and the maximum number of comparisons decreases by 58% and 14% when $TPL = 0.5$ and $TPL = 1.0$, respectively. We notice that the reduction in number of comparisons is more notable for the

mean value than for the maximum one. Actually, when a solution using the FFSS is found, a search terminates. Nonetheless, in the worst case, i.e. when a solution is not found, all processors are tested.

Fig. 4c also depicts standard deviations for each value of the number of comparisons. It can be seen that the more processors in the system and thus the higher value of the mean number of comparisons, the larger the standard deviation. Furthermore, when the value of TPL increases, the standard deviation becomes larger because the processor load is higher and there are more comparisons to carry out and therefore higher chance to have larger standard deviation.

Similar results were obtained for the PB approach with BC deallocation only (and without BC overloading).

To sum up, we can state that, even though the FFSS can miss the best solution, i.e. the one where the PC is scheduled as soon as possible and the BC as late as possible, the FFSS achieves similar performances as the ES, especially when the number of processors is high. Moreover, the FFSS has lower number of comparisons, which makes this technique more suitable for real-time embedded systems and can reduce the energy consumption.

D. Evaluation of the Active Primary/Backup Approach

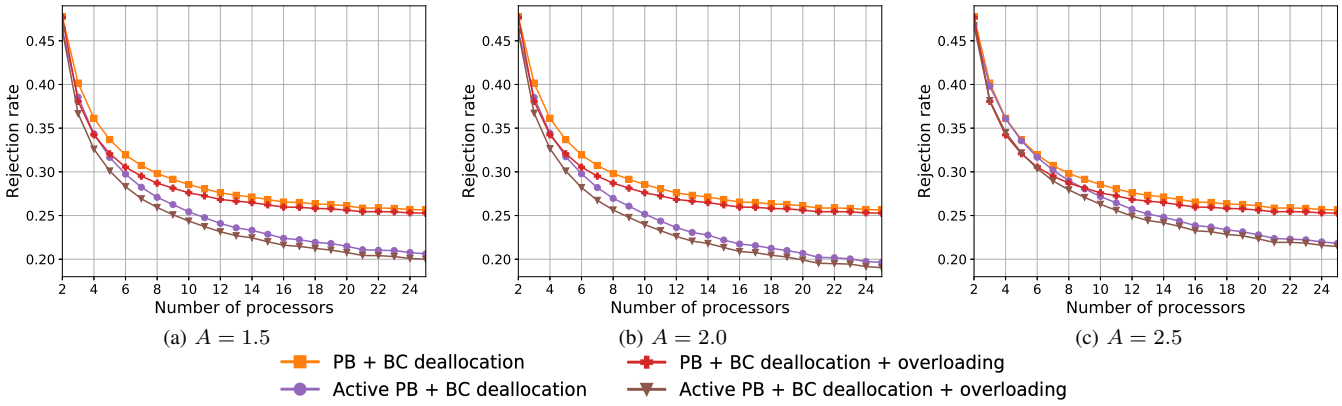
To evaluate the merit of the active PB approach, we make use of the standard simulation parameters as summarised in Table I but instead of the task window between $2c$ and $5c$ we consider its size between c and $5c$. Regarding that the passive PB approach requires the size of at least $2c$, this scenario allows us to assess the active approach.

The algorithm is based on the first found solution search and the TPL is fixed at 1.0. Fig. 5 depicts the rejection rate as a function of the number of processors for the PB approach with BC deallocation only and for the PB approach with BC deallocation and BC overloading and for their respective versions using the active approach.

As it was mentioned in Section III-B, the active PB approach induces system overheads. Consequently, when we employ this approach, we limit its application for tasks with tight deadline. We therefore introduce a threshold A, which is defined as $tw = d - a < A \cdot c$ and which decides whether the active approach is used or not. The value of A in Fig. 5 respectively equals to 1.5, 2.0 and 2.5.

Since the simulation scenario remains the same and only the value of A changes, the results of passive PB approaches are identical. In general, it can be observed that the active PB approach helps to reduce the rejection rate regardless the value of A. The lowest rejection rate is obtained when $A = 2.0$ because this value is located at the transition from the passive PB approach to the active one. For example, the active approach for a 20-processor system reduces the rejection rate by 21% for the PB approach with BC deallocation and by 22% for the PB approach with BC deallocation and BC overloading.

In addition, if A is less than 2.0, some tasks are automatically rejected due to the tight deadline, which is the reason why the rejection rate for $A = 1.5$ (Fig. 5a) is higher than the one for $A = 2.0$ (Fig. 5b). If A is larger than 2.0, no task is automatically rejected but the merit of active approach diminishes. The results (not presented in this paper) show that the higher the value of A, the higher the rejection rate of active approach and therefore the smaller the difference between the passive and active approaches.



(a) $A = 1.5$ (b) $A = 2.0$ (c) $A = 2.5$

Figure 5: Rejection rate of the active PB approach as a function of the number of processors for different values of A ($TPL = 1.0$)

VI. Conclusion

This paper analyses a method using two task copies for fault-tolerant on-line scheduling on multiprocessor embedded systems. It first presents the results of the primary/backup approach alone and the ones with the backup overloading. It can be seen that the technique of backup overloading helps to reduce the rejection rate by up to 13%. When the technique of the backup deallocation is then added, the improvement is even more noteworthy. For instance for 20-processor system and $TPL = 1.0$,

the gain is about 75% compared to the baseline PB approach and regardless whether the technique of backup overloading is implemented or not.

Afterwards, two processor allocation policies are studied: the *exhaustive search* and the *first found solution search*. It was observed that for a multiprocessor system the both techniques achieve similar rejection rate. Even though the exhaustive search has slightly lower rejection rate, the first found solution search surpasses it in the processor load (about 10%) and especially in the number of comparisons (up to 74% for the mean value of this number and 14% for the maximum one). Such a significant gain can be very useful when conceiving fault-tolerant embedded systems subject to energy constraints.

Finally, the active approach is evaluated. It was demonstrated that it helps systems dealing with tasks with tight deadline and it therefore reduces the rejection rate. For 20-processor system, there is a drop of 21% for the PB approach with BC deallocation and of 22% for the PB approach with BC deallocation and BC overloading.

There exists also another interesting metric to evaluate system performances. It is the *Time To Next Fault* (TTNF) [1] 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 (Section III). The lower the TTNF value, the better. Our results were not presented in this paper for lack of space but it was found that all obtained values are of the same magnitude and that the exhaustive search achieves slightly better results than the first found solution search.

Our future work deals with an enhancement for the primary/backup approach, which further decreases the number of comparisons compared to the first found solution search but without worsening system performances, especially the rejection rate. The aim is to improve the approach so that it can be efficiently implemented onto embedded systems with restricted power consumption budget. Moreover, we are about to carry out simulations with fault injection to asses performances of different approaches in a faulty environment not satisfying the TTNF.

REFERENCES

[1] 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, 1997, pp. 272–284.

[2] 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.

[3] 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.

[4] 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.

[5] L. Tang, Z. Huang, Y. Hou, F. Fang, and H. Zhang, "A Fault Detection Approach for MPSoC," in *International Conference on Computer Sciences and Applications*, 2013, pp. 418–422.

[6] G. A. Reis, J. Chang, and D. I. August, "Automatic Instruction-Level Software-Only Recovery," in *IEEE Micro*, vol. 27, no. 1, 2007, pp. 36–47.

[7] O. Goloubeva, M. Rebaudengo, M. S. Reorda, and M. Violante, "Soft-error Detection Using Control Flow Assertions," in *Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT03)*, 2003.

[8] P. Bernardi, L. M. V. Bolzani, M. Rebaudengo, M. S. Reorda, F. L. Vargas, and M. Violante, "A New Hybrid Fault Detection Technique for Systems-on-a-Chip," in *IEEE Transactions on Computers*, vol. 55, no. 2, 2006, pp. 185–198.

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

[10] 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.

[11] 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.

[12] 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.

[13] A. K. Samal, A. K. Dash, P. C. Jena, S. K. Pani, and S. Sha, "Bio-Inspired Approach to Fault-Tolerant Scheduling of Real-Time Tasks on Multiprocessor - A Study," in *IEEE Power, Communication and Information Technology Conference (PCITC)*, 2015, pp. 905–911.

[14] 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.