Reliable Fault-Tolerant Multi-bus Scheduling Algorithm

Chafik ARAR
Department of Computer Science
University of Batna, Algeria

Hamoudi KALLA
Department of Computer Science
University of Batna, Algeria

Salim KALLA
Department of Computer Science
University of Batna, Algeria

Riadh HOCINE
Department of Computer Science
University of Batna, Algeria

ABSTRACT

In this paper, we propose a fault-tolerant scheduling real-time embedded system. This scheduling algorithm is dedicated to multi-bus heterogeneous architectures, which take as input a given system description and a given fault hypothesis. It is based on a data fragmentation and passive redundancy, which allow fast fault detection/retransmission and efficient use of buses. This scheduling approach consists of a list scheduling heuristic based on a Failure Rate Pressure. In order to maximize the reliability of the system, the scheduling of each fragmented data is depend on Failure Rate Pressure. Data fragmentation allows reliable communication and maximizes the reliability of the system. Finally, simulation results show the performance of our approach when using data fragmentation.

Keywords

Embedded systems, real-time systems, fault tolerance, reliability, passive redundancy, data fragmentation, scheduling algorithm.

1. INTRODUCTION

Heterogeneous systems are being increasingly used in many sectors of human activity, such as transportation, robotics, and telecommunication. These systems are increasingly small and fast, but also more complex and critical, and thus more sensitive to faults. Due to catastrophic consequences (human, ecological, and/or financial disasters) that could result from a fault, these systems must be fault-tolerant.

This is why fault tolerant techniques are necessary to make sure that the system continues to deliver a correct service in spite of faults [1, 2]. A fault can affect either the hardware or the software of the system; we chose to concentrate on hardware faults. More particularly, we consider buses faults. A bus is a multi-point connection characterized by a physical medium that connects all the processors of the architecture. As we are targeting embedded systems with limited resources (for reasons of weight, encumbrance, energy consumption, or price constraints), we investigate only software redundancy solutions (Figure. 1).

In this paper, we present our latest work on dependability of real-time embedded systems [3], [4], [5]. The approach that we propose is a scheduling algorithm based on data fragmentation. The scheduling of data fragment (communication) depends on two cost functions failure rate pressure λ_p and schedule pressure σ.

The rest of this paper is organized as follows. In section 2, we give related work on fault tolerance. In section 3, we give details about our systems and fault model. In section 4, we present our scheduling algorithm. Section 5 and 6 details the performances of our approach. Finally, Section 7 concludes the paper.

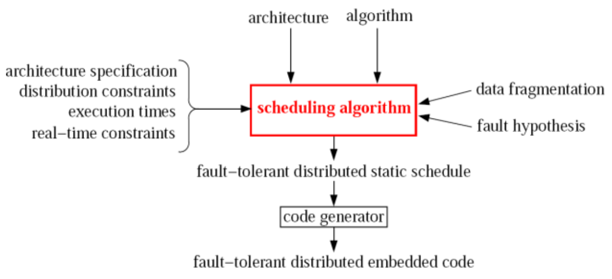

Fig1: The proposed approach.

2. RELATED WORK

In the past decade of fault tolerant systems research, an array of fault tolerant scheduling heuristics has been proposed. However, a few works is done to take into account reliability and real-time criteria at the same time when scheduling tasks when tolerating failures.

Several scheduling heuristics have been proposed to tolerate exclusively processor faults. They are based on active software redundancy or passive software redundancy. In active redundancy, multiple replicas of a task are scheduled on different processors, which are run in parallel to tolerate a fixed number of processor faults. In passive redundancy, also called primary/backup approach, a task is replicated into one primary and several backup replicas, but only the primary replica is executed. If it fails, one of the backup replicas is selected to become the new primary. There are several techniques proposed to tolerate exclusively buses faults are based on proactive [6, 7] or reactive schemes [8]. In [9], a method of identifying bus faults based on support vector machine is proposed. In [2], faults of buses are tolerated using a TDMA (Time Division Multiple Access) communication protocol and an active redundancy approach. The approach proposed in [10] tolerates only a specified set of buses permanent faults. The method proposed in [10] is only suited to one class of algorithms called fan-in algorithms.

In [11], failures are tolerated using the fault recovery scheme and a primary/backups strategy. In [12], Dima et al. propose an original off-line fault tolerant scheduling algorithm which uses the active replication of tasks and communications to tolerate a set of failure patterns; each failure pattern is a set of processor and/or communications media that can fail simultaneously, and each failure pattern correspond a reduced architecture. The proposed algorithm starts by building a basic schedule for each reduced architecture plus the nominal architecture, and then merges these basic schedules to obtain a distributed fault tolerant schedule. It has been implemented very recently by Pinello et al. [13].

We have proposed in [14] a solution to tolerate transient faults in distributed heterogeneous architectures with multiple-bus topology. However, the solution does not take into account hardware reliability. The approach presented in this paper can tolerate upto a fixed number of arbitrary bus transient faults.

3. MODELS

3.1 Algorithm Model

The algorithm is modeled as a data-flow graph, called algorithm graph and noted ALG. Each vertex of ALG is an operation (task) and each edge is a data-dependence. A data-dependence, noted by →, corresponds to a data transfer between a producer operation and a consumer operation. $o_1 \rightarrow o_2$ means that o_1 is a predecessor of o_2, and o_2 is a successor of o_1. Operations with no predecessor (resp. no successor) are the input interfaces (resp. output), handling the events produced by the sensors (resp. actuators). Figure. 2 presents an example of an algorithm graph, with seven operations v_1, v_2, v_3, v_4, v_5, v_6 and v_7.

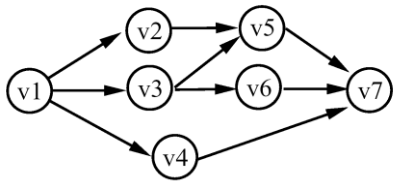

Fig 2: Algorithm graph.

3.2 Architecture Model

The architecture is modelled by a non-directed graph, noted ARC, where each node is a processor, and each edge is a bus. Classically, a processor is made of one computation unit, one local memory, and one or more communication units, each connected to one communication link. Communication units execute data transfers. We assume that the architecture is heterogeneous and fully connected. Figure 3 is an example of ARC, with four processors P1, P2, P3 and P4, and three buses B1, B2 and B3.

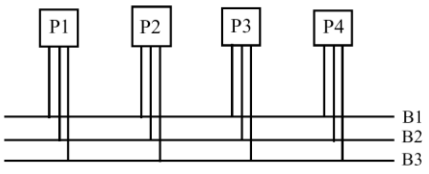

Fig 3: Architecture graph.

Our real-time system is based on cyclic executive; this means that a fixed schedule of the operations of ALG is executed cyclically on ARC at a fixed rate. This schedule must satisfy one real-time constraint which is the length of the schedule. As we target heterogeneous architecture, we associate to each operation o_i a worst case execution time (WCET) on each processor p_j of ARC, noted $exe(o_i; p_j)$. Also, we associate to each data dependency dpd_i a worst case transmission time (WCTT) on each bus b_j of the architecture, noted $exe(dpd_i; b_j)$.

3.3 Fault Model

We assume only hardware components (buses) failures and we assume that the algorithm is correct w.r.t. its specification, i.e., it has been formally validated, for instance with model checking and/or theorem proving tools. We consider only transient bus faults. Transient faults, which persist for a short duration, are significantly more frequent than other faults in systems [15]. Permanent faults are a particular case of transient faults. We assume that at most NBF buses faults can arise in the system, and that the architecture includes more than NBF buses.

4. THE PROPOSED SOLUTION

In this section, we present our reliable fault-tolerant scheduling algorithm. The algorithm we propose is a greedy list scheduling heuristic, called A Reliable Bus Fault-tolerant algorithm (RBF). It is based on a data fragmentation. The objective of the algorithm is to maximize the reliability of the system, and at the same time attempts to minimize the length of the whole generated schedule in both presence and absence of failures. In our approach, we achieve high reliability and fault tolerance in three ways:

—*Data fragmentation:* the communication of each data dependency is fragmented into NBF+1 fragments, sent by each operation source of the data-dependency via NBF+1 distinct buses to each of operation destination. As our approach uses data fragmentation, the scheduling of each fragmented data is depend on the bus failure rates λ_B.

—*Passive replication:* to tolerate NBF buses faults, each data dependency is replicated on NBF+1 replicas. Each replica is fragmented on NBF+1 fragments scheduled on NBF+1 distinct Buses. We called primary replica the replica with the earliest ending time and the other ones are the backup replicas. Only the primary replica (its NBF+1 fragments) is executed. If one fragment fails, one of the backup fragments replicas is selected to become the new primary.

— *The Failure Rate Pressure λ_p (FRP)*

The FRP is the failure rate of the obtained multiprocessor schedule. Using the FRP is very satisfactory in the area of periodically executed schedules. This is the case in most real-time embedded systems, which are periodically sampled systems. Our fault tolerance heuristic is FRP-based to control precisely the scheduling of each fragmented data from the beginning to the end of the schedule. We define the FRP of scheduling an operation oi, noted $\lambda p(S_n)$, by the following equation:

$$\lambda_p(o_i) = \lambda_j * exe(o_i, p_j) + \sum_k (\lambda_l * exe(dpd_k, b_1)) \qquad (1)$$

Where, b_l is chosen from the b_l best buses failure rate.

The reliability $R(S_n)$ is computed, for each operation o_i and each processor p_j , by the following equation:

$$R(S_n) = \prod_i e^{-\lambda_k exe(o_i, p_k) + \sum_i \sum_j \lambda_c exe(dpd_j^k, b_c)} \qquad (2)$$

Where, S_n is the schedule generated at step n of the scheduling algorithm.

4.1 Scheduling algorithm

Our scheduling algorithm is a greedy list scheduling heuristic, which schedules one operation at each step n. It generates a distributed static schedule of a given algorithm ALG onto a given architecture ARC, which minimizes the system's run-time, and tolerates upto NBF buses faults. Our algorithm is based on Failure Rate Pressure FRP and a schedule pressure. Failure Rate Pressure is used to select the reliable bus for each data dependency. The schedule pressure, noted by $\sigma^{(n)}_{(oi;\,pj)}$ is used in our algorithm as a cost function to select the best operation which minimize the length of the critical path taking into account data fragmentation. It is computed for each operation as follows:

$$\sigma^{(n)}(o_i, p_j) = S^{(n)}_{o_i,p_j} + \bar{S}^{(n)}_{o_i} - R^{n-1}$$

where, R^{n-1} is the critical path length of the partial schedule composed of the already scheduled operations, $S^{(n)}_{oi;pj}$ is the earliest time at which the operation o_i can start its execution on the processor p_j , and $S^{(n)}_{oi}$ is the latest start time from the end of o_i, defined to be the length of the longest path from o_i to ALG's output operations. The schedule pressure measures how much the scheduling of the operation lengthens the critical path of the algorithm. Therefore it introduces a priority between the operations to be scheduled. Note that, since all candidates operations at step n have the same value $R^{(n-1)}$, it is not necessary to compute $R^{(n-1)}$.
The RBF scheduling algorithm is shown in Figure 4. The set of candidate operations O_{cand} is initialized as the operations without predecessor. The set of scheduled operations O_{sched} is initially empty. In the selection step, a processor is selected among all the processor of ARC to schedule each operation and its communication data. The selection rule is based $\sigma_{(n)}$ and FRP.
The scheduled operation obest is removed from O_{cand}, and the operations of ALG which have all their predecessors in the new set of scheduled operations are added to this set.

5. AN EXAMPLE

We have applied the RBF heuristic to an example of an algorithm graph and an architecture graph composed of four processors and four buses. The algorithm graph is presented in Figure 5. The failure rates of all the processors are all equal to 10^{-5}, and the failure rate of the Buses SAM_MP2, SAM_MP1, SAM_MP3 and SAM_MP4 are respectively 10^{-6}, 10^{-6}, 10^{-5} and 10^{-4}.
Figure 6 shows the non-reliable schedule produced for our example with a basic scheduling heuristic. The schedule length generated by this heuristic is 30.2.

```
Algorithm RBF:
input: ALG, ARC, NBF;
output: a reliable fault-tolerant schedule;
Initialize the lists of candidate and scheduled operations:
n := 0;
$O^{(0)}_{cand} := \{o \in O \mid pred(o) = \emptyset\}$;
$O^{(0)}_{sched} := \emptyset$;
While $O^{(n)}_{cand} \neq \emptyset$ do
  ① For each candidate operation $o_{cand}$, compute $\sigma^{(n)}$ and the
     failure rate pressure $\lambda_p$ on each processor $p_k$.
  ② For each candidate operation $o_{cand}$, select the best processor
     $p^{o_{cand}}_{best}$ which minimizes $\sigma^{(n)}$ and FRP.
  ③ Select the most urgent candidate operation $o_{urgent}$ between
     all $o^i_{cand}$ of $O^{(n)}_{cand}$.
  ④ fragment each data communication of $o_{urgent}$ on NBF frag-
     ments
  ⑤ Schedule $o_{urgent}$ and its fragmented data;
  ⑥ Update the lists of candidate and scheduled operations:
     $O^{(n)}_{sched} := O^{(n-1)}_{sched} \cup \{o_{urgent}\}$;
     $O^{(n+1)}_{cand} := O^{(n)}_{cand} - \{o_{urgent}\} \cup \{o' \in succ(o_{urgent}) \mid$
     $pred(o') \subseteq O^{(n)}_{sched}\}$;
  ⑦ n := n + 1;
end while
end
```

Fig 4: The RBF scheduling algorithm.

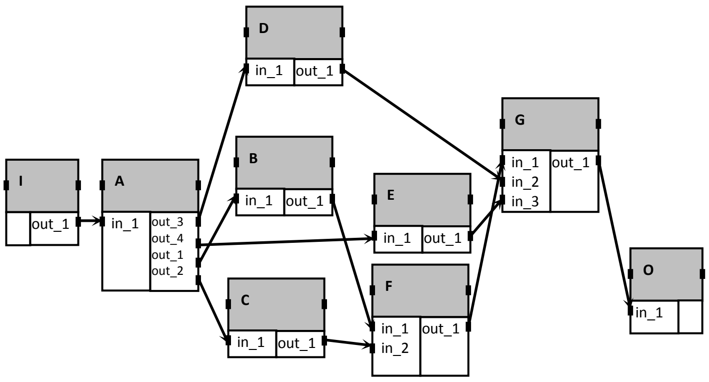

Fig 5: Algorithm graph.

We apply our heuristic to the example of Figure 5. The user requires the system to tolerate one bus failure, i.e., NBF = 1. Figure 7 shows the scheduled generated by our heuristic.

The schedule length generated by our heuristic is 19.8.

Figure 8 shows the scheduled generated by our heuristic for NBF=2.

The schedule length generated by our heuristic is 21.8.

The important thing to note in Figures 7 and 8 is that data fragmentations reduces the schedule length and improve significantly the reliability of the system.

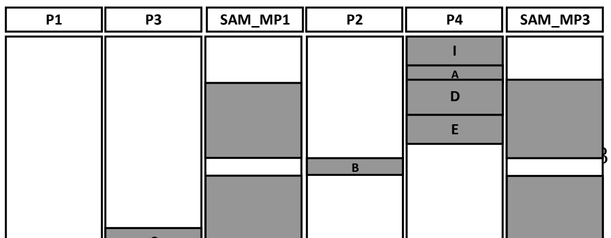

Fig 6: Final schedule generated.

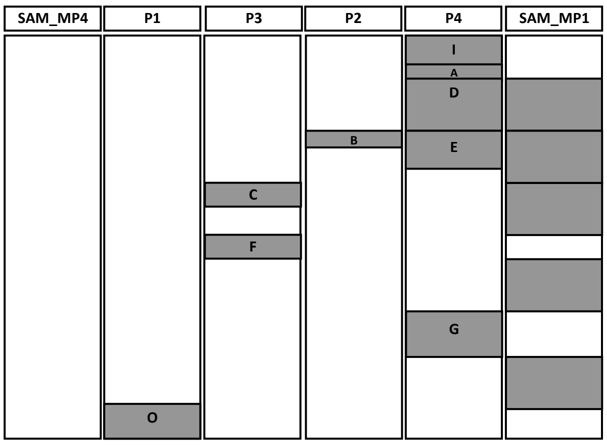

Fig 7: fault tolerant schedule for NBF=1.

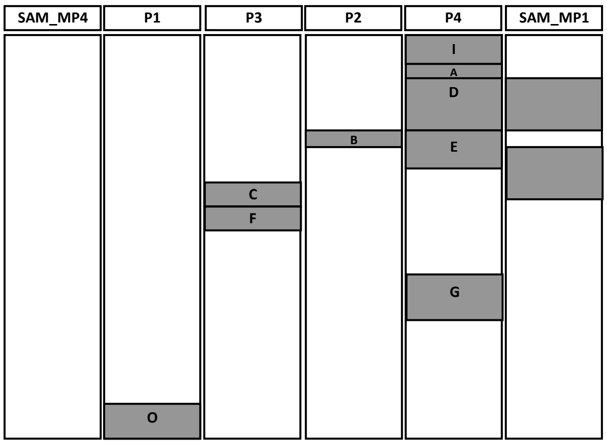

Fig 8: fault tolerant schedule for NBF=2.

6. SIMULATION

To evaluate our heuristic, we have applied the RBF heuristic to a random algorithm graphs and a heterogeneous and completely connected architecture graph composed of 4 processors. The following figures have been obtained with a CCR set to 1, 5, 10 and 20. CCR (Communication-to-Computation Ratio) is the ratio between the average communication cost (over all the data dependencies) and the average computation cost (over all the operations). A random algorithm graph is generated as follows: given the number of operations N, we randomly generate a set of levels with a random number of operations. Then, operations at a given level are randomly connected to operations at a higher level.

The execution times of each operation are randomly selected from a uniform distribution with the mean equal to the chosen average execution time. Similarly, the communication times of each data dependency are randomly selected from a uniform distribution with the mean equal to the chosen average communication time.

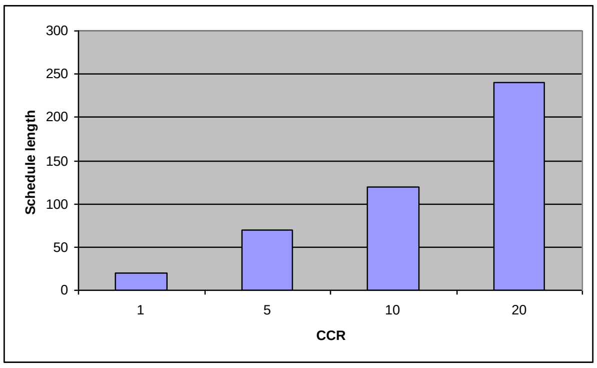

Fig 9: Impact on schedule length of the CCR for NBF=1.

The general objective of our simulations is to study the impact of the data fragmentation and CCR on the schedule length and reliability introduced by RBF. Figure 9 (Respectively Figure 10) shows the impact on schedule length obtained by RBF, for P=4 and NBF=1 (Respectively NBF=2). As we can see, the schedule length grows almost linearly when CCR increases from 1 to 20.

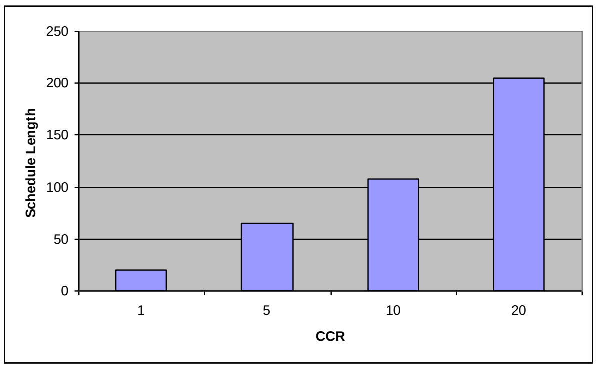

Fig 10: Impact on schedule length of the CCR for NBF=2.

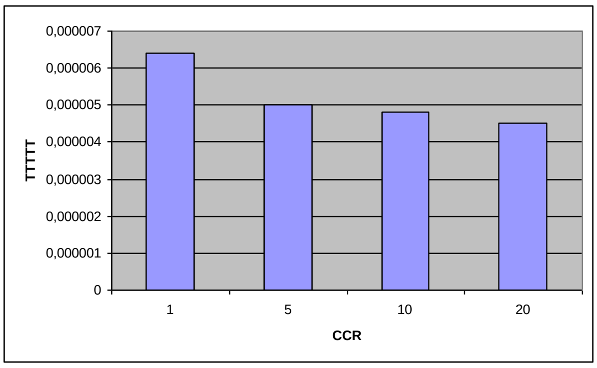

Fig 11: Impact on FRP of the CCR for NBF=1.

Figure 11 (Respectively Figure 12) shows the impact on the FRP obtained by RBF, for P=4 and NBF=1 (Respectively NBF=2). As we can see, the FRP decreases when CCR increases from 1 to 20.

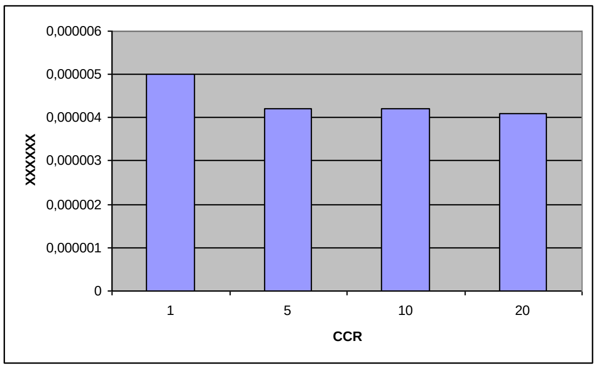

Fig 12: Impact on FRP of the CCR for NBF=2.

7. CONCLUSION

In this paper, we have studied the problem of fault-tolerance in embedded real-time systems and proposed a software implemented fault-tolerance solution for multi-buses architectures.

We have proposed a new scheduling heuristic, called RBF, which produces automatically a static distributed fault-tolerant schedule of a given algorithm ALG on a given multi-buses architecture ARC. Our solution is based on the software redundancy of communications. Only the fragments of the primary copy of the message, called primary, is sent; if one fragment fails, another fragments of the message, called backup, will be transmitted.

The implementation uses a scheduling heuristic for optimizing the critical path and maximizing the reliability of the distributed algorithm obtained. Finally, we plan to experiment our method on an electric autonomous vehicle, with a 5-processor multi-buses architecture.

8. REFERENCES

[1] P. Jalote. Fault-Tolerance in Distributed Systems. Prentice Hall, Englewood Cliffs, New Jersey, 1994.

[2] H. Kopetz and G. Bauer. The time-triggered architecture. PIEEE, 91(1):112–126, October 2003.

[3] Salim Kalla, Hamoudi Kalla, and Chafik Arar. Article: Reliability-driven fault tolerant scheduling heuristics for distributed embedded real-time systems. International Journal of Computer Applications, 36(5):5–11, December 2011. Published by Foundation of Computer Science, New York, USA.

[4] Ismail Assayad, Alain Girault, and Hamoudi Kalla. Tradeoff exploration between reliability, power consumption, and execution time for embedded systems. International Journal on Software Tools for Technology Transfer, pages 1–17, 2012.

[5] Ismail Assayad, Alain Girault, and Hamoudi Kalla. Scheduling of real-time embedded systems under reliability and power constraints. In Complex Systems (ICCS), 2012 International Conference on, pages 1–6. IEEE, 2012.

[6] N. Kandasamy, J.P. Hayes, and B.T. Murray. Dependable communication synthesis for distributed embedded systems. In International Conference on Computer Safety, Reliability and Security, SAFECOMP'03, Edinburgh, UK, September 2003.

[7] S. Dulman, T. Nieberg, J. Wu, and P. Havinga. Trade-off between traffic overhead and reliability in multipath routing for wireless sensor networks. In Wireless Communications and Networking Conference, 2003.

[8] B. Kao, H. Garcia-Molina, and D. Barbara. Aggressive transmissions of short messages over redundant paths. TPDS, 5(1):102–109, January 1994.

[9] Hong Song and Hao Wu. The applied research of support vector machine in bus fault identification. In Natural Computation (ICNC), 2010 Sixth International Conference on, volume 3, pages 1326–1329, Aug.

[10] R. Vaidyanathan and S. Nadella. Fault-tolerant multiple bus networks for fan-in algorithms. In International Parallel Processing Symposium, pages 674–681, April 1996.

[11] X. Qin, H. Jiang, and D. R. Swanson. An efficient primarysegmented backup scheme for dependable real-time communication in multihop networks. In IEEE/ACM trans. on Networking, 2003.

[12] C. Dima; A. Girault; C. Lavarenne; and Y. Sorel. Off-line realtime fault-tolerant scheduling. In 9th Euromicro Workshop on Parallel and Distributed Processing, pages 410–417, 2001.

[13] L. Carloni C. Pinello and A. Sangiovanni Vincentelli. Faulttolerant deployment of embedded software for cost-sensitive real-time feedback-control applications design. In Automation and Test in Europe , DATE'04, IEEE, 2004.

[14] A. Girault, H. Kalla, and Y. Sorel. Transient processor/bus fault tolerance for embedded systems. In IFIP Working Conference on Distributed and Parallel Embedded Systems, DIPES'06, pages 135–144, Braga, Portugal, October 2006. Springer.

[15] M. Pizza, L. Strigini, A. Bondavalli, and F. Di Giandomenico. Optimal discrimination between transient and permanent faults. In 3rd IEEE High Assurance System Engineering Symposium, pages 214–223, Bethesda, MD, USA, 1998.

[16] Alain Girault and Hamoudi Kalla. A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate. IEEE TDSC, 6(4):241–254, 2009.