A Reliable Fault-Tolerant Scheduling Algorithm for Real Time Embedded Systems

Chafik Arar, Hamoudi Kalla, Salim Kalla, Hocine Riadh

A Reliable Fault-Tolerant Scheduling Algorithm for Real Time Embedded Systems

Chafik Arar, Hamoudi Kalla, Salim Kalla, and Hocine Riadh

Department of Computer Science
University of Batna, Algeria
{chafik.arar,hamoudi.kalla}@gmail.com
{salim.kalla,hocine.riadh}@univ-batna.dz

Abstract. In this paper, we propose a fault-tolerant scheduling for real-time embedded systems. Our 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. Our scheduling approach consist of a list scheduling heuristic based on a Global System Failure Rate (GSFR). In order to maximize the reliability of the system, the size of each fragmented data depends on GSFR and the bus failure rates. variable fragment size allows reliable communication and to maximize the reliability of the system. Finally, simulation results show the performance of our approach when using data fragmentation with a variable fragment size.

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 [5, 8]. A fault can affect either the hardware or the software of the system; we chose to concentrate on hardware faults. More particularly, we consider bus 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). The approach that we propose is a scheduling algorithm based on data fragmentation. The size of each fragment data depends on two cost functions GSFR (Global System Failure Rate) and schedule pressure σ.

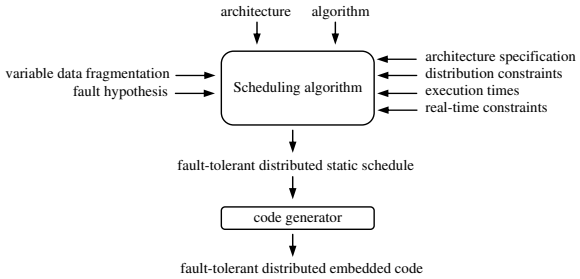

Fig. 1. The proposed approach.

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. Sections 5 and 6 details the performances of our approach. Finally, Section 7 concludes the paper.

2 Related work

In the literature, we can identify several fault tolerance approaches for distributed embedded real-time systems. There are several methods that tolerate processor and bus faults. Here, we present only related work involving scheduling heuristics to tolerate bus faults. Techniques proposed to tolerate exclusively bus faults are based on proactive or reactive schemes. In the proactive scheme [6, 2], multiple redundant copies of a message are sent along distinct buses. In contrast, in the reactive scheme [7], only one copy of the message, called primary, is sent; if it fails, another copy of the message, called backup, will be transmitted. In [12], A method of identifying bus faults based on a support vector machine is proposed. In [8], faults of buses are tolerated using a TDMA (Time Division Multiple Access) communication protocol and an active redundancy approach. The approach proposed in [13] tolerates only a specified set of buses permanent faults. The method proposed in [13] 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 [9], 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 corresponds to 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. [1].

We have proposed in [3] 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 that we propose in this paper is more general since it uses only software redundancy solutions, i.e., no extra hardware is required. Moreover, our approach can tolerate upto a fixed number of arbitrary bus transient faults, and it is based on GSFR and data fragmentation with a variable fragment size to maximize system reliability.

3 Models

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 $\rightarrow$, 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 $v1$, $v2$, $v3$, $v4$, $v5$, $v6$ and $v7$.

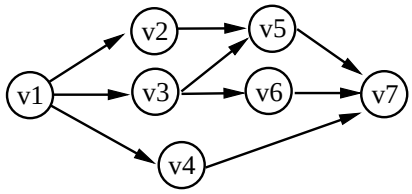

Fig. 2. Algorithm graph.

The architecture is modeled 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.

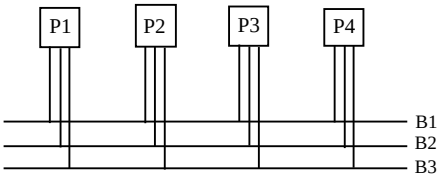

Fig. 3. Architecture graph.

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 $exe(dpd_i, b_j)$.

3.1 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 [10]. Permanent faults are a particular case of transient faults. We assume that at most NBF bus 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 variable 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 variable data fragmentation, the size of each fragmented data depends on GSFR and the bus failure rates λ_B.
- *Passive replication:* to tolerate NBF bus 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 fragments fails, one of the backup fragments replicas is selected to become the new primary.
- *The Global System Failure Rate per time unit (GSFR)*
  The GSFR is the failure rate per time unit of the obtained multiprocessor schedule. Using the GSFR 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. In such cases, applying brutally the

exponential reliability model yields very low reliabilities due to very long execution times (the same remark applies also to very long schedules). Hence, one has to compute beforehand the desired reliability of a single iteration from the global reliability of the system during its full mission; but this computation depends on the total duration of the mission (which is known) and on the duration of one single iteration (which may not be known because it depends on the length of the schedule under construction). In contrast, the GSFR remains constant during the whole system's mission: the GSFR during a single iteration is by construction identical to the GSFR during the whole mission.
Our fault tolerance heuristic is GSFR-based to control precisely the scheduling of each fragmented data from the beginning to the end of the schedule. In [4], we have defined the GSFR of scheduling an operation o_i, noted $\Lambda(S_n)$, by the following equation:

$$\Lambda(S_n) = \frac{-\log R(S_n)}{U(S_n)} \tag{1}$$

where, S_n is the static schedule at step n of the algorithm, and $U(S_n)$ is the total utilization of the hardware resources, defined by:

$$U(S_n) = \sum_{i}^{j} exe(o_i, p_j) + \sum_{k}^{m} exe(dpd_k, b_m) \tag{2}$$

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

$$R(S_n) = \prod_{i} e^{-\lambda_k exe(o_i,p_k)+\sum_k \sum_j \lambda_c exe(dpd_j^k,b_c)} \tag{3}$$

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 bus faults. Our algorithm is based on two cost functions GSFR and a schedule pressure. GSFR is used to select the reliable bus for each data dependency and to choose the best size for each fragment. The schedule pressure, noted by $\sigma^{(n)}(o_i, p_j)$ 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:

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

where, R^{n-1} is the critical path length of the partial schedule composed of the already scheduled operations, $S^{(n)}_{o_i,p_j}$ is the earliest time at which the operation

o_i can start its execution on the processor p_j, and $\overline{S}_{o_i}^{(n)}$ 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.

Algorithm RBF**:**
input: ALG, ARC, NBF;
output: a reliable fault-tolerant schedule;
Initialize the lists of candidate and scheduled operations:
n := 0;
$O_{cand}^{(0)} := \{o \in O \mid pred(o) = \emptyset\}$;
$O_{sched}^{(0)} := \emptyset$;
While $O_{cand}^{(n)} \neq \emptyset$ do

① For each candidate operation o_{cand}, compute $\sigma^{(n)}$ and GSFR on each processor p_k.

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

$$\Lambda(S_n) = \frac{-\log R(S_n)}{U(S_n)}$$

② For each candidate operation o_{cand}, select the best processor $p_{best}^{o_{cand}}$ which minimizes $\sigma^{(n)}$ and GSFR.

③ Select the most urgent candidate operation o_{urgent} between all o_{cand}^{i} of $O_{cand}^{(n)}$.

④ fragment each data communication of o_{urgent} on NBF fragments

⑤ Schedule o_{urgent} and its fragmented data;

⑥ Update the lists of candidate and scheduled operations:
$O_{sched}^{(n)} := O_{sched}^{(n-1)} \cup \{o_{urgent}\}$;
$O_{cand}^{(n+1)} := O_{cand}^{(n)} - \{o_{urgent}\} \cup \{o' \in succ(o_{urgent}) \mid pred(o') \subseteq O_{sched}^{(n)}\}$;

⑦ n := n + 1;

end while
end

Fig. 4. The RBF scheduling algorithm.

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 GSFR. The scheduled operation o_{best} 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}.

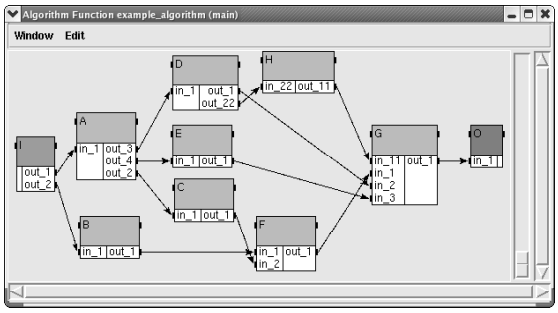

Fig. 5. Algorithm graph.

Figure 6 shows the non-reliable schedule produced for our example with a basic scheduling heuristic (for instance the one of SynDEx). SynDEx is a tool for optimizing the implementation of real-time embedded applications on multicomponent architecture. The schedule length generated by this heuristic is 22.2. The GSFR of the non-reliable schedule Λ_{syndex} is equal to 0.0000246.

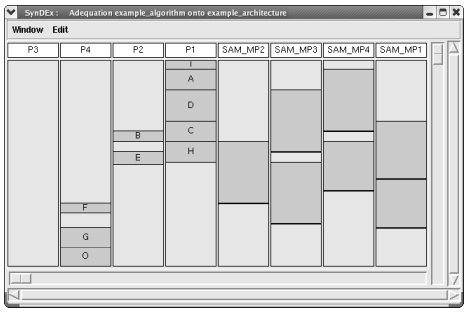

Fig. 6. Schedule generated by SynDEx.

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 17.2. The GSFR of the non-reliable schedule $\Lambda_{nbf} = 1$ is equal to 0.00000597.

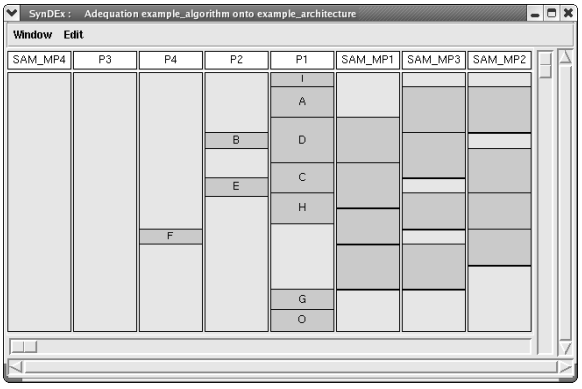

Fig. 7. fault tolerant schedule for NBF=1.

Figure 8 shows the scheduled generated by our heuristic for NBF=2. The schedule length generated by our heuristic is 15.2. The GSFR of the non-reliable schedule $\Lambda_{nbf} = 2$ is equal to 0.00000613.

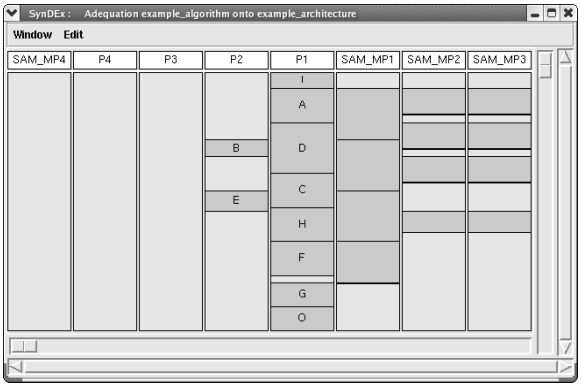

Fig. 8. fault tolerant schedule for NBF=2.

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

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.

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.

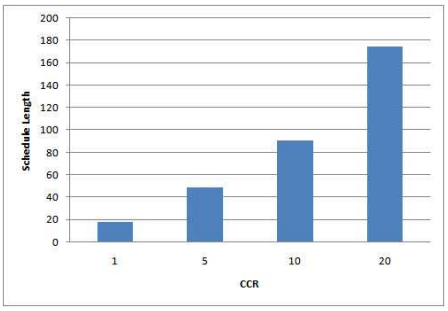

Fig. 9. Impact on schedule length of the CCR for NBF=1.

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

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 based on GSFR. We have proposed a new scheduling

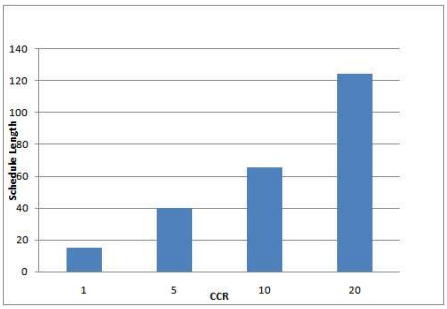

Fig. 10. Impact on schedule length of the CCR for NBF=2.

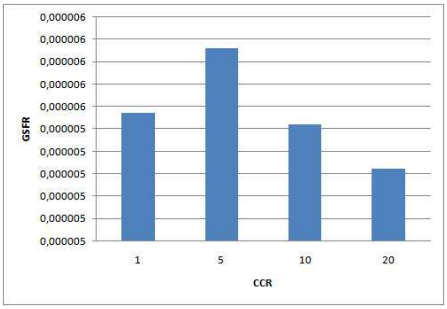

Fig. 11. Impact on GSFR of the CCR for NBF=1.

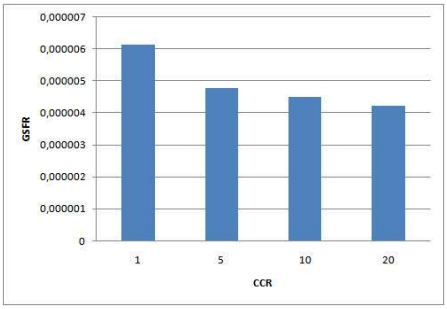

Fig. 12. Impact on GSFR of the CCR for NBF=2.

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.

References

1. L. Carloni C. Pinello and A. Sangiovanni Vincentelli. Fault-tolerant deployment of embedded software for cost-sensitive real-time feedback-control applications design. In *Automation and Test in Europe , DATE'04, IEEE*, 2004.
2. 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.
3. 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.
4. Alain Girault and Hamoudi Kalla. A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate. *IEEE Transactions on Dependable and Secure Computing*, 6(4):241–254, 2009.
5. P. Jalote. *Fault-Tolerance in Distributed Systems.* Prentice Hall, Englewood Cliffs, New Jersey, 1994.
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. B. Kao, H. Garcia-Molina, and D. Barbara. Aggressive transmissions of short messages over redundant paths. *TPDS*, 5(1):102–109, January 1994.
8. H. Kopetz and G. Bauer. The time-triggered architecture. *PIEEE*, 91(1):112–126, October 2003.
9. C. Dima; A. Girault; C. Lavarenne; and Y. Sorel. Off-line real-time fault-tolerant scheduling. In *9th Euromicro Workshop on Parallel and Distributed Processing*, pages 410–417, 2001.
10. 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.
11. X. Qin, H. Jiang, and D. R. Swanson. An efficient primary-segmented backup scheme for dependable real-time communication in multihop networks. In *IEEE/ACM trans. on Networking*, 2003.
12. 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.
13. R. Vaidyanathan and S. Nadella. Fault-tolerant multiple bus networks for fan-in algorithms. In *International Parallel Processing Symposium*, pages 674–681, April 1996.