TRANSIENT PROCESSOR/BUS FAULT TOLERANCE FOR EMBEDDED SYSTEMS

With hybrid redundancy and data fragmentation

Alain Girault[1], Hamoudi Kalla[2], and Yves Sorel[3]

[1]*INRIA Rhône-Alpes, 655 avenue de l'Europe, 38334 Saint-Ismier cedex, FRANCE*
Alain.Girault@inrialpes.fr

[2]*IRISA, Campus Universitaire de Beaulieu, 35042 Rennes Cedex France Cedex, FRANCE*
Hamoudi.Kalla@irisa.fr

[3]*INRIA Rocquencourt, B.P.105 – 78153 Le Chesnay Cedex, FRANCE*
Yves.Sorel@inria.fr

Abstract We propose an approach to build fault-tolerant distributed real-time embedded systems. From a given system description (application algorithm and architecture) and a given fault hypothesis (type and number of faults to be tolerated), we generate automatically a static fault-tolerant multiprocessor schedule of the algorithm components on the target architecture, which minimizes the schedule length, and tolerates transient faults of both processors and communication media. Our approach is dedicated to heterogeneous architectures with multiple processors linked by several shared buses. It is based on hybrid redundancy and data fragmentation strategies, which allow fast fault detection and handling. This scheduling problem is NP-hard and we rely on a heuristic algorithm to obtain efficiently an approximate solution. Our simulation results show that our approach generally reduces the schedule length overhead.

Keywords: real-time embedded systems, safety-critical systems, transient faults, scheduling heuristics, hybrid redundancy, data fragmentation, heterogeneous architectures.

1. Introduction

Today, embedded real-time systems invade many sectors of human activity, such as transportation, robotics, and telecommunication. The progresses achieved in electronics and data processing improve the performances of these systems. As a result, the new 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]. A fault can affect either the hardware or the software of the system. Thanks to formal validation techniques, such as model-checking and theorem proving, a lot of software faults can be prevented. Although software faults are still an important issue, we chose to concentrate on hardware faults. More particularly, we consider processor and bus faults. A bus is a multipoint 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, volume, energy consumption, or price constraints), we investigate only software redundancy solutions based on scheduling algorithms.

The paper is organized as follows. Sections 2 and 3 describe respectively related work and system models. Section 4 states the faults assumptions and our fault-tolerance problem. Section 5 presents our approach for providing fault-tolerance, and Section 6 details the performances of our approach. Finally, Section 7 concludes the paper and proposes future research directions.

2. Related work

The literature about fault tolerance of distributed embedded real-time systems is very abundant. Yet, there are very few methods that manage to tolerate both processor and bus faults. Here, we present related work involving scheduling heuristics to tolerate processor faults, bus faults, or both.

Processor faults. Several scheduling heuristics have been proposed to tolerate exclusively processor faults. They are based on active software redundancy [2, 3] or passive software redundancy [4–6]. 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. [2] presents an off-line scheduling algorithm that tolerates a single processor faults in multiprocessor systems, while [3] tolerates multiple 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. For instance, [5] presents a scheduling algorithm that tolerates one processor fault.

Bus faults. Techniques proposed to tolerate exclusively buses faults are based on proactive or reactive schemes. In the proactive scheme [7, 8], multiple redundant copies of a message are sent along distinct buses. In contrast, in the reactive scheme [9], only one copy of the message, called primary, is sent; if it fails, another copy of the message, called backup, will be transmitted.

Processor and bus faults. Few techniques have been proposed to tolerate both processor and bus faults [10–12]. In [10], faults of buses are tolerated using a TDMA (Time Division Multiple Access) communication protocol and an active redundancy approach, while faults of processors are tolerated using a

hardware redundancy approach. The approach proposed in [11] tolerates only a specified set of processor and bus permanent faults. The method proposed in [12] is only suited to one class of algorithms called fan-in algorithms. Our approach is more general since it uses *only software redundancy solutions*, i.e., no extra hardware is required, because hardware resources in embedded systems are limited. Moreover, our approach can tolerate up to a fixed number of *arbitrary processor and bus transient faults*. This is important since transient faults [13] are increasingly the majority of faults in logic circuits, due to radiation, energetic particles, and so on.

3. System description

In this section, we present the system models (algorithm and architecture), and define the execution characteristics of the algorithm on the architecture.

Algorithm model. The algorithm is modeled by a data-flow graph, called algorithm graph and noted $\mathcal{A}lg$. Each vertex of $\mathcal{A}lg$ is an operation and each edge is a data-dependency. A data-dependency $(o_1 \triangleright o_2)$ corresponds to a data transfer from a producer operation o_1 to a consumer operation o_2, defining a partial order on the execution of operations. We say that o_2 is a successor of o_1, and that o_1 is a predecessor of o_2. An operation of $\mathcal{A}lg$ can be either an external input/output operation or a computation operation. Operations with no predecessor (resp. no successor) are the input interfaces (resp. output), handling the events produced by the sensors (resp. actuators). The inputs of a computation operation must precede its outputs. Moreover, computation operations are side-effect free, i.e., the output values depend only of the input values.

Figure 1(left) is an example of $\mathcal{A}lg$, with seven operations: In_1 and In_2 (resp. Out_1) are input (resp. output) operations, while A, B, C and D are computation operations. The data-dependencies between operations are depicted by arrows. For instance the data-dependency $(A \triangleright D)$ can correspond to the sending of some arithmetic result computed by A and needed by D.

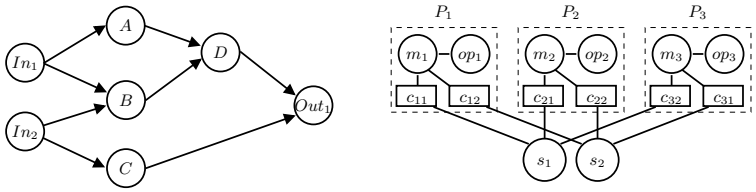


Figure 1. Example of an algorithm graph (left) and an architecture graph (right).

Architecture model. The architecture is composed of two principal components: a processor and a bus. A processor P_i consists of an operator op_i, a memory resource m_i of type RAM (Random Access Memory), and several communicators c_{ij}. A bus B_i consists of one communicator for each existing

processor and one memory resource s_i of type SAM (Sequential Access Memory). Each operator executes sequentially a set of operations of $\mathcal{A}lg$, and reads and writes data from and into its local memory. Each communicator of each processor cooperates with each other in order to execute sequentially transfers of data stored in the memory between processors through a SAM.

The architecture is modeled by a non-directed graph, called architecture graph and noted $\mathcal{A}rc$. Vertices of $\mathcal{A}rc$ are: operators, communicators, and memory resources. Edges of $\mathcal{A}rc$ are connections between these components. Figure 1(right) gives an example of $\mathcal{A}rc$, with three processors P_1, P_2, and P_3, and two buses B_1=$\{s_1, c_{11}, c_{21}, c_{31}\}$ and B_2=$\{s_2, c_{12}, c_{22}, c_{32}\}$, where each processor P_i is made of one operator op_i, one local memory m_i, and two communicators c_{i1} and c_{i2}.

Execution characteristics. We target systems based on a cyclic execution model; this means that a fixed schedule of the operations of $\mathcal{A}lg$ is executed cyclically on $\mathcal{A}rc$ at a fixed rate. This schedule must satisfy one real-time constraint $\mathcal{R}tc$ and a set of distribution constraints $\mathcal{D}is$. In our execution model $\mathcal{E}xe$, we associate to each operator op a list of pairs $\langle o, d/op\rangle$, where d is the worst case execution time (WCET) of the operation o on op. Also, we associate to each communicator c a list of pairs $\langle dpd, d/c\rangle$, where d is the worst case transmission time (WCTT) of the data-dependency dpd on c. Since we target heterogeneous architecture, WCET (resp. WCTT) for a given operation (resp. data-dependency) can be distinct on each operator (resp. communicator). Specifying the distribution constraints $\mathcal{D}is$ amounts to associating the value "∞" to some pairs of $\mathcal{E}xe$: $\langle o, \infty/op\rangle$ meaning that o cannot be executed on op. Finally, since we produce static schedules, we can compute their length and compare it to the real-time constraint $\mathcal{R}tc$.

4. Fault model and scheduling problem definition

In our fault hypothesis, we assume only hardware faults and a fault-free software. We consider only transient processor and bus faults. Transient faults, which persist for a "short" duration, are significantly more frequent than other faults in systems [13]. Permanent faults are a particular case of transient faults. We assume at most $\mathcal{N}pf$ processor faults and $\mathcal{N}bf$ buses faults can occur in the system, and that the architecture includes at least $\mathcal{N}pf$+1 processors and $\mathcal{N}bf$+1 buses. Our problem is therefore formally stated as:

PROBLEM 1 *Given:*

- *a distributed heterogeneous architecture $\mathcal{A}rc$ composed of a set $\mathcal{P}$ of processors and a set $\mathcal{B}$ of buses: $\mathcal{P} = \{\ldots, P_i, \ldots\}$, $\mathcal{B} = \{\ldots, B_j, \ldots\}$*
- *an algorithm $\mathcal{A}lg$ composed of a set $\mathcal{O}$ of operations and a set $\mathcal{E}$ of data-dependencies: $\mathcal{O} = \{\ldots, o_i, \ldots, o_j, \ldots\}$, $\mathcal{E} = \{\ldots, (o_i \triangleright o_j), \ldots\}$*
- *all the execution characteristics $\mathcal{E}xe$ of the algorithm components of $\mathcal{A}lg$ on the architecture components of $\mathcal{A}rc$,*

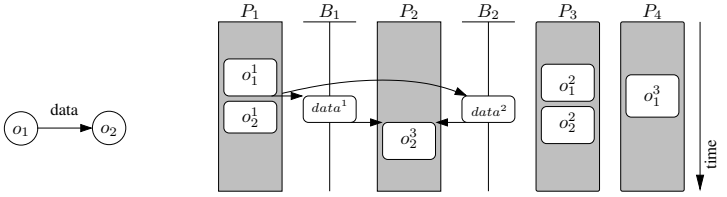
- *a real-time constraint $\mathcal{R}tc$ (schedule length), and several distribution constraints $\mathcal{D}is$,*
- *a number $\mathcal{N}pf < |\mathcal{P}|$ of processor faults that may affect the system,*
- *a number $\mathcal{N}bf < |\mathcal{B}|$ of bus faults that may affect the system,*

find a multiprocessor static schedule of $\mathcal{A}lg$ on $\mathcal{A}rc$, which minimizes the schedule length, and tolerates up to $\mathcal{N}pf$ processor and $\mathcal{N}bf$ bus faults with respect to $\mathcal{R}tc$, $\mathcal{E}xe$, and $\mathcal{D}is$.

5. The proposed approach

Our solution is based on hybrid redundancy and data fragmentation techniques. In the aim to minimize communication overhead, we use active redundancy to tolerate processor faults, and passive redundancy to tolerate bus faults. The reason why to use data fragmentation is to minimize the fault detection latency, i.e, the time it takes to detect a fault.

Hybrid redundancy and data fragmentation. In order to tolerate $\mathcal{N}pf$ processor and $\mathcal{N}bf$ bus faults, each operation is replicated in $\mathcal{N}pf$+1 replicas scheduled on $\mathcal{N}pf$+1 distinct processors. The replica with the earliest ending time is the primary replica, while the other ones are the backup replicas. The earliest ending time is the sum of the earliest starting time (computed in absence of faults) plus the operation's WCET. The data of each data dependency is fragmented into $\mathcal{N}bf$+1 packets, sent by the primary replica of the data-dependency source via $\mathcal{N}bf$+1 distinct buses to each of the $\mathcal{N}pf$+1 replicas of the data-dependency destination. For example, in the schedule of Figure 2b, operations o_1 and o_2 of Figure 2a are replicated into three replicas to tolerate two processors faults ($\mathcal{N}pf$=2), and the data of the data-dependency $(o_1 \triangleright o_2)$ are fragmented into two packets to tolerate one bus fault ($\mathcal{N}bf$=1).



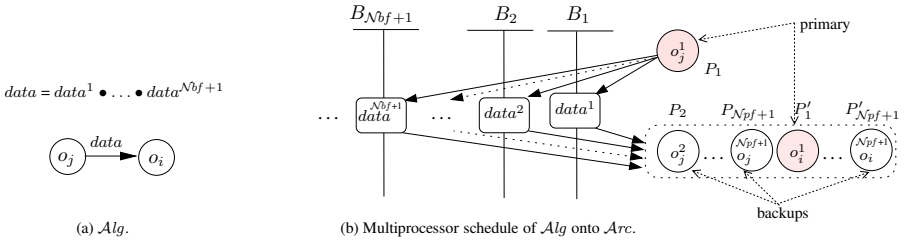
(a) $\mathcal{A}lg$. (b) Multiprocessor schedule of $\mathcal{A}lg$ onto $\mathcal{A}rc$.

Figure 2. Tolerating two processors and one bus faults.

Figure 3 illustrates these principles in the general case where $\mathcal{N}pf \geq 1$ and $\mathcal{N}bf \geq 1$. Only the primary replica of each operation o_j sends all the fragmented data "$data^m$", of each of its data outputs, in parallel via $\mathcal{N}bf$+1 buses to all the replicas of all its successor operations in $\mathcal{A}lg$.

Communication mechanism. Each operation receives each of its data inputs via $\mathcal{N}bf$+1 buses; when it has received all the packets of each data input, it defragments these packets and starts its execution. In some cases, the

replica of an operation will only receive some of its inputs once, through an intra-processor communication; this will occur whenever one of its predecessor operations has one of its replicas scheduled on the *same* processor.



(a) $\mathcal{A}lg$. (b) Multiprocessor schedule of $\mathcal{A}lg$ onto $\mathcal{A}rc$.

Figure 3. Tolerating $\mathcal{N}pf$ processors and $\mathcal{N}bf$ buses faults.

Transient fault recovery and handling. In Figure 3, three cases can occur:

1. All the packets $data^m$ sent by o_j^1 are received: in this case, each replica of o_i defragments these packets and starts its execution. Also, each replica of o_j receives a copy of these packets, which it ignores.

2. None of the packets $data^m$ sent by o_j^1 are received: this concerns $\mathcal{N}bf$+1 packets, and as no more than $\mathcal{N}bf$ buses faults may occur in the system (by hypothesis), this means the failure of the processor P_1 executing the replica o_j^1. To deal with this failure, one backup replica among the $\mathcal{N}pf$ other replicas of o_j is selected to re-send all the packets $data^m$ via the same buses. Since the fault of processor P_1 can be transient, it is not marked as faulty by the other processors. This scheme can be improved by deciding that, if a processor remains faulty during some number of consecutive executions of the schedule (e.g., 5), then its fault is permanent and this processor is permanently removed from the schedule.

3. Some packets $\{data^m, \ldots, data^k\}$ sent by o_j^1 are not received: let $data^-$ be this set of missing packets, and $\mathcal{B}^-=\{B^m, \ldots, B^k\}$ be the set of the buses that were supposed to transmit them. Since other packets have been received, it means that P_1, the processor executing o_j^1 is not faulty, and hence that the buses of $\mathcal{B}^-$ are faulty. Therefore, the same replica o_j^1 re-sends the packets $data^-$ via other buses chosen among the set $\mathcal{B} \setminus \mathcal{B}^-$. Since the fault of the buses of $\mathcal{B}^-$ can be transient, they are not marked as being faulty. This scheme can be improved with a similar approach as in step **2**.

In summary, this communication mechanism yields three advantages: ① fast fault detection; ② fast distinction between processor and bus faults; and ③ fast fault recovery.

We have implemented these principles in a greedy list scheduling heuristic, called FT-AAA (Fault-Tolerant Adequation Algorithm Architecture). In the following algorithm of FT-AAA, the superscript numbers in parentheses refer to the steps of the heuristic, e.g., $O_{sched}^{(n)}$:

ALGORITHM FT-AAA

- ***Inputs*** *= $\mathcal{A}lg$, $\mathcal{A}rc$, $\mathcal{N}pf$, $\mathcal{N}bf$, $\mathcal{E}xe$, $\mathcal{R}tc$, and $\mathcal{D}is$;*
- ***Output*** *= a fault-tolerant multiprocessor static schedule;*

INITIALIZATION

Initialize the sets of candidate operations O_{cand} and scheduled operations O_{sched}:

$O^{(1)}_{cand} := \{$*operations of $\mathcal{A}lg$ without predecessors*$\}$*;*

$O^{(1)}_{sched} := \emptyset$*;*

While $O^{(n)}_{cand} \neq \emptyset$ ***do***

SELECTION

- *Select for each candidate operation o_{cand} of $O^{(n)}_{cand}$ a set $\mathcal{P}_{best}$ of $\mathcal{N}pf$ +1 processors that minimizes the dependable schedule pressure (Equation (1));*
- *Select for each candidate operation o_{cand} of $O^{(n)}_{cand}$, among the processors $\mathcal{P}_{best}(o_{cand})$, the best processor P_{best} that maximizes the dependable schedule pressure;*
- *Select, among all the pairs (o_{cand}, P_{best}), the best pair (o_{best}, P_{best}) that maximizes the dependable schedule pressure;*

DISTRIBUTION AND SCHEDULING

- *Let $\mathcal{P}_{best}(o_{best})$ be a best set of $\mathcal{N}pf$ +1 processors of o_{best} computed at the "Selection" step;*
- *For each o_j, predecessor of o_{best}, fragment the $data$ of the data-dependency $(o^1_j \triangleright o_{best})$ into $\mathcal{N}bf$ +1 packets $data^m$;*
- *Schedule the packets $data^m$ of each data-dependency on $\mathcal{N}bf$ +1 distinct buses;*
- *Add $\mathcal{N}pf$ replicas of o_{best} into $\mathcal{A}lg$;*
- *Schedule each replica o^k_{best} on the processor P^k_{best} of $\mathcal{P}_{best}(o_{best})$.*

UPDATE SETS

- *Update the sets of candidate and scheduled operations for the next step $(n+1)$:*

$O^{(n+1)}_{sched} := O^{(n)}_{sched} \cup \{o_{best}\}$*;*

$O^{(n+1)}_{cand} := O^{(n)}_{cand} - \{o_{best}\} \cup$

$\left\{o_{new} \in \{\text{successors of } o_{best}\} \mid \{\text{predecessors of } o_{new}\} \subseteq O^{(n+1)}_{sched}\right\}$*;*

end While

END OF THE ALGORITHM

The algorithm of FT-AAA is divided in four main steps:

Initialization step. The set of candidate operations $O^{(1)}_{cand}$ is initialized as the operations without predecessor. Later, an operation is said to be a candidate if all its predecessors are already scheduled. The set of scheduled operations $O^{(1)}_{sched}$ is initially empty.

Selection step. For each candidate operation $o_{cand} \in O^{(n)}_{cand}$, a set $\mathcal{P}_{best}$ of $\mathcal{N}pf$+1 processors is selected among all the processors of $\mathcal{P}$ to schedule $\mathcal{N}pf$+1 replicas of o_{cand}. The selection rule is based on the dependable schedule pressure function, noted $\sigma^{(n)}$. It is computed, for each operation $o_i \in O^{(n)}_{cand}$ and each processor $P_j \subset \mathcal{P}$, as follows:

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

where $S^{(n)}_{o_i,P_j}$ is the earliest time at which operation o_i can start its execution on processor P_j, $\overline{S}^{(n)}_{o_i}$ is the latest start time from end of o_i (defined to be the length of the longest path from the output operations to o_i), and $R^{(n-1)}$ is the schedule length at step $(n-1)$. The set $\mathcal{P}_{best}$ of each $o_{cand} \in O^{(n)}_{cand}$ is composed of the $\mathcal{N}pf$+1 processors that minimize $\sigma^{(n)}$. Then, among all $O^{(n)}_{cand}$, the most urgent candidate o_{best}, with a processor $P_{best} \in \mathcal{P}_{best}(o_{best})$ that maximizes this function, is selected to be replicated and scheduled.

Distribution and scheduling step. This step involves first replicating the best candidate o_{best} into $\mathcal{N}pf + 1$ replicas, and second scheduling each replica o^k_{best} of o_{best} respectively on the processor P^k_{best} of $\mathcal{P}_{best}$. Before scheduling each of these replicas, the data of each data-dependency are fragmented into $\mathcal{N}bf$+1 packets that are scheduled on $\mathcal{N}bf$+1 distinct buses.

Updating step. The scheduled operation o_{best} is removed from $O^{(n)}_{cand}$, and the operations of $\mathcal{A}lg$ which have all their predecessors in the new set of scheduled operations are added to this set.

6. Simulations

To evaluate FT-AAA, we have implement it in SYNDEX, a CAD tool for optimizing and implementing real-time embedded systems (`http://www.syndex.org`). Then, we have applied the FT-AAA heuristic to a set of randomly generated algorithm graphs and an architecture graph composed of five processors ($|\mathcal{P}| = 5$) and four buses ($|\mathcal{B}| = 4$). In our simulations, we study the impact of $\mathcal{N}pf$, $\mathcal{N}bf$, the number of operations N, and CCR (Communication to Computation Ratio) on the schedule length overhead introduced by FT-AAA, computed by Equation (2):

$$\text{overhead} = \frac{\text{length(FT-AAA}(\mathcal{N}pf,\mathcal{N}bf)) - \text{length(AAA)}}{\text{length(AAA)}} \tag{2}$$

where FT-AAA takes as parameter the numbers of processor and bus faults $(\mathcal{N}pf, \mathcal{N}bf)$, AAA is exactly FT-AAA$(0,0)$, and "length" is a function that computes the schedule's length.

Impact of $\mathcal{N}bf$ and N. We have plotted in Figure 4 the average overheads on the schedule length of 100 random algorithm graphs for each N, $\mathcal{N}pf$=0, CCR=1, and $\mathcal{N}bf$=1, 2, 3. This figure shows that the average overhead is very low (between 6% and 18%) and increases slightly with N. This is due first to $\mathcal{N}pf$=0, i.e., operations of $\mathcal{A}lg$ are not replicated, and second to the use of passive redundancy of communication. Also, for the three values of $\mathcal{N}bf$, the heuristics FT-AAA(0,1), FT-AAA(0,2) and FT-AAA(0,3) bear almost similar results with no significant advantage between the three variants.

Impact of $\mathcal{N}pf$ and N. We have plotted in Figure 5 the average overheads on the schedule length of 100 random $\mathcal{A}lg$ for each N, $\mathcal{N}bf$=0, CCR=1,

and $\mathcal{N}pf$=1, 2. This figure shows that the average overhead when $\mathcal{N}pf$=1 is 45%, while for $\mathcal{N}pf$=2 it is 75%. These figures are much lower than the expected 100% when all computations are scheduled twice, and 200% when all computations are scheduled thrice. It also shows that the performances of FT-AAA decrease when $\mathcal{N}pf$ increases. This is due to the fact that FT-AAA uses the active redundancy of operations. However, for the two values of $\mathcal{N}pf$, FT-AAA($\mathcal{N}pf$,0) produces almost no significant difference between the overheads obtained for the different values of N.

Impact of CCR. We have plotted in Figure 6 the average overheads on the schedule length of 100 random $\mathcal{A}lg$ for N=40, $\mathcal{N}pf$=1, $\mathcal{N}bf$=1,2,3, and each CCR. Thanks to the data fragmentation, this figure shows that, when the communications are less expensive than the computations (CCR <1), the performances are almost identical for $\mathcal{N}bf$=1 to 3. In contrast, when the communications are more expensive (CCR >1), the performances decrease when $\mathcal{N}bf$ increases. Also, for $\mathcal{N}bf \leq 2$, CCR has no significant impact on the performances of FT-AAA; again this is due to the data fragmentation. It is not true anymore when $\mathcal{N}bf \geq 3$, because the number of buses, 4, becomes limitative.

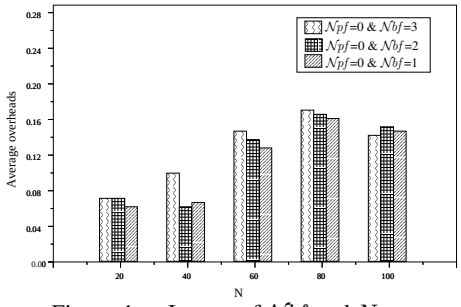


Figure 4. Impact of $\mathcal{N}bf$ and N

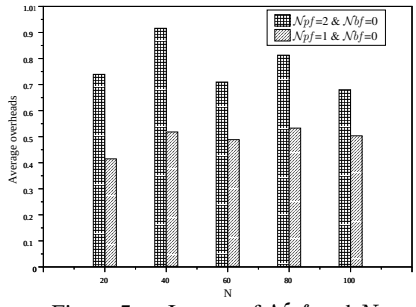


Figure 5. Impact of $\mathcal{N}pf$ and N

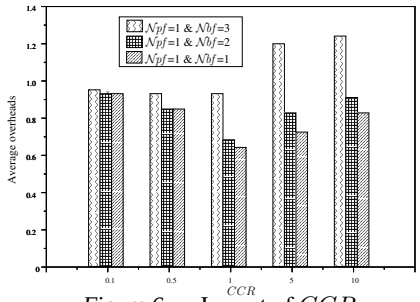


Figure 6. Impact of CCR

7. Conclusion

We have proposed in this paper a solution to tolerate transient faults of both processors and communication media in distributed heterogeneous architectures with multiple-bus topology. Our solution, based on hybrid redundancy and data-fragmentation strategies, is a list scheduling heuristic, called

FT-AAA. It generates automatically a multiprocessor static schedule of a given algorithm on a given architecture, which minimizes the schedule length, and tolerates up to $\mathcal{N}pf$ processors and up to $\mathcal{N}bf$ buses faults, with respect to real-time and distribution constraints. The communication mechanism, based on data-fragmentation, allows the fast distinction between processor and bus faults, the fast detection of faults, and the fast handling of faults. Simulations show that our approach can generally reduce the schedule length overhead. Currently, we are working on an improved solution to take sensors/actuators faults into account.

References

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

[2] K. Hashimoto, T. Tsuchiya, and T. Kikuno. Effective scheduling of duplicated tasks for fault-tolerance in multiprocessor systems. *IEICE Trans. on Information and Systems*, E85-D(3):525–534, March 2002.

[3] 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, DSN'03*, San-Francisco, USA, June 2003. IEEE.

[4] K. Ahn, J. Kim, and S. Hong. Fault-tolerant real-time scheduling using passive replicas. In *Pacific Rim International Symposium on Fault-Tolerant Systems*, Taipei, Taiwan, December 1997.

[5] 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 *International Conference on Parallel Processing*, pages 360–386, Vancouver, Canada, August 2002.

[6] Y. Oh and S. H. Son. Scheduling real-time tasks for dependability. *Journal of Operational Research Society*, 48(6):629–639, June 1997.

[7] 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.

[8] 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.

[9] B. Kao, H. Garcia-Molina, and D. Barbara. Aggressive transmissions of short messages over redundant paths. *IEEE Trans. on Parallel and Distributed Systems*, 5(1):102–109, January 1994.

[10] H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 91(1):112–126, October 2003.

[11] C. Dima, A. Girault, and Y. Sorel. Static fault-tolerant scheduling with "pseudo-topological" orders. In *Joint Conference FORMATS-FTRTFT'04*, volume 3253 of *LNCS*, Grenoble, France, September 2004. Springer-Verlag.

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

[13] 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.