Asian Journal of Information Technology 14 (2): 67-73, 2015
ISSN: 1682-3915

A Reliable Multi-Bus Fault-Tolerant Scheduling Algorithm Based on Variable Data Fragmentation

[1]Chafik Arar and [2]Mohamed Salah Khiereddine
[1]Department of Computer Science,
[2]Department of Electronics, University of Batna, Batna, Algeria

Abstract: In this study, we propose an approach to build fault-tolerant distributed real-time embedded systems. From a given system description and a given fault hypothesis, we generate automatically a fault-tolerant distributed schedule based on GSFR of the source algorithm onto the target architecture which minimizes the system's run-time and tolerates buses communication failures. The scheduling algorithm proposed is dedicated to multi-bus heterogeneous architectures with multiple processors linked by several shared buses. It is based on passive redundancy and variable data fragmentation strategies which allow fast fault detection/retransmission and efficient use of buses, the size of each fragmented data depends on GSFR and the bus failure rates, variable fragment size allows reliable communication and maximize the reliability of the system. As this scheduling problem is NP-hard, we use a heuristic algorithm to obtain an approximate efficiently solution and we are able to show with simulation results that our approach can generally reduce the run-time overhead.

Key words: Distributed real-time systems, heterogeneous systems, software implemented fault-tolerance, transient faults, reliability, scheduling heuristics, passive redundancy, variable data fragmentation

INTRODUCTION

Nowadays, heterogeneous systems are being 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 (Jalote, 1994; Kopetz, 2011; Niino, 2012; Hanmer, 2013).

A fault can affect either the hardware or the software of the system; we chose to concentrate on hardware faults. More particularly, we consider communication faults (Grunsteidl *et al*., 2014) and more specifically bus faults. A bus is a multi-point connection characterized by a physical medium that connects all the processors of the architecture. In the literature, we can identify several fault-buses tolerance approaches for distributed embedded real-time systems which we classify into two categories: proactive or reactive schemes.

In the proactive scheme (Kandasamy *et al*., 2005; Dulman *et al*., 2003), multiple redundant copies of a message are sent along distinct buses. In contrast, in the reactive scheme only one copy of the message called primary is sent; if it fails, another copy of the message called backup will be transmitted. In Dima *et al*. (2001), an original off-line fault tolerant scheduling algorithm which uses the active replication of tasks and communications to tolerate a set of failure patterns is proposed 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 by Pinello *et al*. (2004).

A new solution to tolerate transient faults in distributed heterogeneous architectures with multiple-bus topology is proposed (Girault *et al*., 2006). However, the solution does not take into account hardware reliability. A method of identifying bus faults based on a support vector machine is proposed (Song and Wu, 2010). Faults of buses are tolerated using a TDMA (Time Division Multiple Access) communication protocol and an active redundancy approach (Kopetz, 2011). Researcher propose a fine grained transparent recovery where the property of transparency can be selectively applied to processes and messages (Izosimov *et al*., 2012). Researchers survey the

Corresponding Author: Chafik Arar, Department of Computer Science, University of Batna, Batna, Algeria

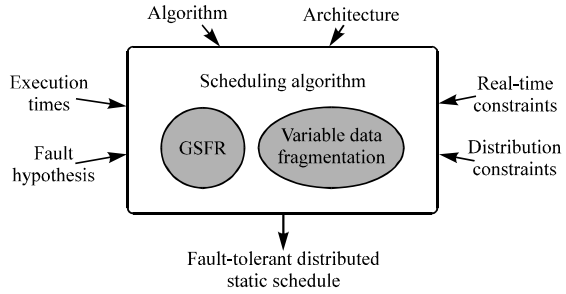

Fig. 1: The proposed approach

problem of how to schedule tasks in such a way that deadlines continue to be met despite processor and communication media (permanent or transient) or software failure (Krishna, 2014).

As we are targeting embedded systems with limited resources (for reasons of weight, encumbrance, energy consumption or price constraints), the approach that we propose in this study (Fig. 1) is more general since it uses only software redundancy solutions, i.e., no extra hardware is required. Moreover, our approach can tolerate up to a fixed number of arbitrary bus transient faults and is scheduling algorithm based on GSFR and data fragmentation with a variable fragment size to maximize system reliability.

MATERIALS AND METHODS

System description: Distributed real-time embedded systems are composed of two principal parts which are the algorithm (software part) and the distributed architecture (hardware part). The specification of these systems involve describing the algorithm (algorithm model), the architecture (architecture model) and the execution characteristics of the algorithm onto the architecture (execution model).

The algorithm is modeled as a data-flow graph 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. $t_1 \rightarrow t_2$ means that t_1 is a predecessor of t_2 and t_2 is a successor of t_1. Operations with no predecessor (resp. no successor) are the input interfaces (resp. output), handling the events produced by the sensors (resp. actuators).

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

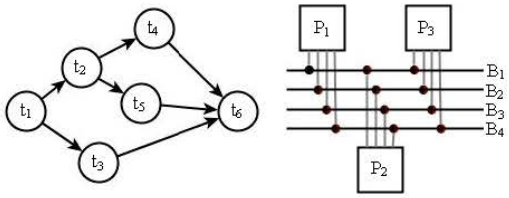

Fig. 2: ALG and ARC graphs

transfers. We assume that the architecture is heterogeneous and fully connected. Figure 2 presents an example of ALG with seven operations t_1-t_7 and ARC with three processors P_1-P_3 and four buses B_1-B_4.

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 t_i a Worst Case Execution Time (WCET) on each processor P_j of ARC, noted exe (t_i, P_j). Also, we associate to each data dependency $data_i$ a Worst Case Transmission Time (WCTT) on each bus B_j of the architecture, noted exe ($data_i$, B_j).

Fault model: We assume only hardware components (processors and 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 (Pizza *et al*., 1998). Permanent faults are a particular case of transient faults. We assume that at most N bus faults can arise in the system and that the architecture includes more than N buses.

Fault-tolerant scheduling algorithm based on variable data fragmentation: In this study, we first discuss the basic principles used in our solution based on scheduling algorithms. Then, we describe in details our scheduling algorithm. The aims of this algorithm are twofold, first, maximize the reliability of the system, secondly, 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:

Passive replication: To tolerate N bus faults, each data dependency is replicated on N+1 replicas. Each replica is fragmented on N+1 fragments scheduled on N+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 (it's N+1 fragments) is executed. If one fragments fails, one of the backup fragments replicas is selected to become the new primary.

Variable data fragmentation: In order to use efficiently the bus redundancy of the architecture, we propose to use a mechanism of communication based on data fragmentation. Variable data fragmentation allows the fast recovering from partial/complete buses errors and it may also reduce the error detection latency (the time it takes to detect the error). The communication of each data dependency $t_i \rightarrow t_j$ is fragmented into N+1 fragments data = $data_1 \bullet \ldots \bullet data_{N+1}$, sent by each operation source of the data-dependency via N+1 distinct buses to each of operation destination (Fig. 3). The operation (•) is used to concatenate two data packets; it is associative. As our approach uses variable data fragmentation, the size of each fragmented data depends on GSFR and the bus failure rates λ_B.

Variable data fragmentation operates in three phases: first in order to tolerate at most N communication bus errors each data dependency is fragmented into N+1 fragments of equal size. The initial size of each fragment is calculated by:

$$\text{Size}(\text{data}_i) = \frac{\text{Size}(\text{data})}{N+1}$$

The main problem with the equal size data fragmentation comes from the difference between ending time of different fragments (Fig. 4) because the destination operation must wait to getting all the fragments of the data dependency to start execution.

Second, the goal of passing from equal size data fragmentation to variable data fragmentation (Fig. 4) is to minimize the difference between ending time ET of different fragments (Fig. 5).

$$ET_1 \leq ET_2 \leq \ldots \leq ET_{N+1}$$

$$\forall_{i \in \{1, \ldots, N+1\}}, \text{Minimize}\{ET_{i+1} - ET_i\}$$

With variable data fragmentation based on minimizing the difference between ending time, another problem can occur and grows extremely the execution time. The bus over which accumulates data may also fail, therefore, the quantity of data to be retransmitted is more important.

Third, the definition of a compromise between the load of each communication bus and the maximum data to be transmitted on this bus as illustrated in Fig. 6. Variable data fragmentation must not exceed this value when defining the new fragments size. The improvement in time of the scheduling is shown in Fig. 7. The algorithm that enable variable data fragmentation is show as:

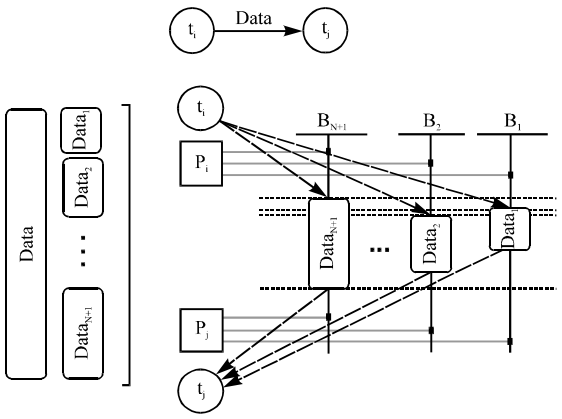

Fig. 3: Variable data fragmentation

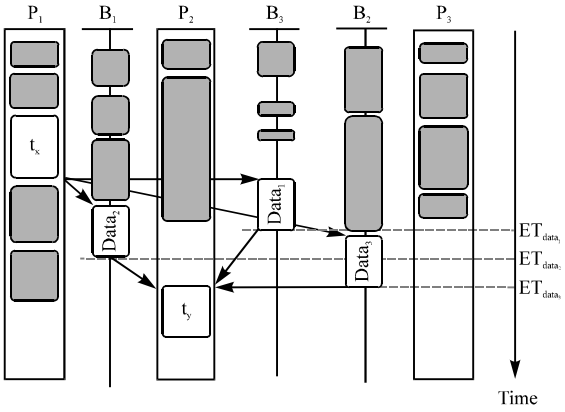

Fig. 4: Ending time in equal size data fragmentation

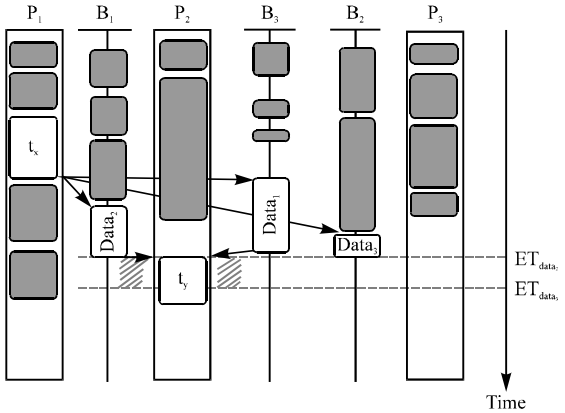

Fig. 5: Minimize difference between ending time ET

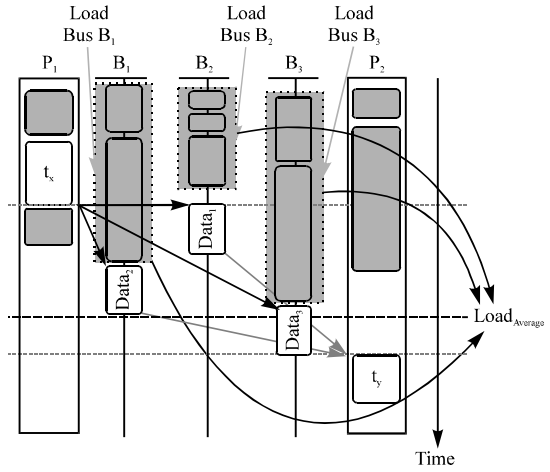

Fig. 6: The average load $Load_{average}$

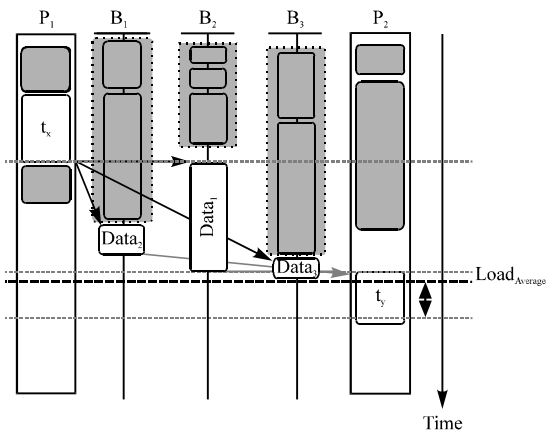

Fig. 7: The improvement in time of the scheduling

Algorithm (The VDF algorithm):

input: data-dependence (data = $t_i \neg t_j$), N

output: the set of N+1 affection ($data_i(B_x)$).;

1. Each data dependency (data = $t_i \neg t_j$) is fragmented into N+1 fragments of equal size:

$$Size(data_1) = ..Size(data_{N+1}) = \frac{Size(data)}{N+1}$$

2. Compute the loading sill of buses:

$$Load_{Average} = \frac{\Sigma \lambda_{B_i} \times Load(B_i)}{N+1}$$

3. Schedule the N+1 fragments of data-dependence on N+1 bus
4. Order the data fragments according to thier ending time:

$$ET_1 \leq ET_2 \leq ... \leq ET_{N+1}$$

5. Compute the sum of the shift of ending time:

$$Sum^{new}_{shift\text{-}time} := 0; Sum_{shift\text{-}time} = \Sigma ET_{i+1} - ET_i$$

6. While $\left(Sum^{new}_{shift\text{-}time} \leq Sum_{shift\text{-}time}\right)$ do

a) $Sum_{shift\text{-}time} := Sum^{new}_{shift\text{-}time}$

b) Fragment the data fragment with the last end time on two fragment ($data(ET_{N+1}) = data_A \bullet data_B$), respecting the following three conditions:

(i) $Size(data_A) \geq Size_{min}(data_{ET_i})$

(ii) $Size(data_{ET_i}) + Size(data_B) \leq Load_{Average}$

(iii) $ET_1 + Size(data_B) B_{data1} \leq ET_{N+1}$

(c) Order the data fragments according to theIr new ending time ET_i

(d) Compute the new value of $Sum^{new}_{shift\text{-}time}$:

$$Sum^{new}_{shift\text{-}time} = \Sigma ET_{i+1} - ET_i ;$$

End While

End

GSFR: The Global System Failure Rate (GSFR) per time unit 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. Girault and Kalla (2009) has defined the GSFR of scheduling an operation t_i, noted $\Delta(S_n)$ by equation:

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

Where:

S_n = The static schedule at step n of the algorithm

$U(S_n)$ = The total utilization of the hardware resources, defined by:

$$U(S_n) = \sum_i^j exe(t_i, p_j) + \sum_k^m exe(data_k, b_m)$$

The reliability $R(S_n)$ is computed for each operation t_i and each processor P_j by equation:

$$R(S_n) = \prod_i e^{-\lambda_k exe(t_i, p_k)} + \sum i \sum j^{\lambda_c exe(data_j^k, b_c)}$$

Scheduling algorithm: These tree principles are implemented by a scheduling algorithm called algorithm. It is a greedy list scheduling heuristic, 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 N buses faults with respect to the real-time constraint and the distribution constraints. 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.

At each step of our greedy list scheduling heuristic, the schedule pressure noted by $\sigma(n)(t_i, P_j)$ is used as a cost function to select the best operation which minimize the length of the critical path taking into account variable data fragmentation, it is computed for each operation as follows:

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

Where:

R^{n-1} = The critical path length of the partial schedule composed of the already scheduled operations

$S^{(n)}_{oi, pj}$ = The earliest time at which the operation t_i can start its execution on the processor P_j

$\overline{S}^{(n)}_{t_i}$ = The latest start time from the end of t_i, defined to be the length of the longest path from oi 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. The RMFS-VDF scheduling algorithm is shown as:

Algorithm (The RMFS-VDF scheduling algorithm):

input: ALG, ARC, N;
output: a reliable fault-tolerant schedule;

Initialize the lists of candidate and scheduled operations;
n : = 0;
$T^{(0)}_{cand}$: = {t∈T|pred(t) = ϕ};
$T^{(0)}_{sched}$: = ϕ;
While ($T^{(0)}_{cand} \neq \phi$) do

1) For each candidate operation t_{cand}, compute $\sigma^{(n)}$ and GSFR on each processor P_k
2) For each candidate operation t_{cand}, select the best processor $p^{t_{cand}}_{best}$ which minimizes $\sigma^{(n)}$ and GSFR
3) Select the most urgent candidate operation t_{urgent} between all t_{cand} of $T^{(n)}_{cand}$
4) For each data dependencies whose t_{urgent} is the producer operation: fragment the data comminication on Nbf fragment using the variable data fragmentation algorithm;
5) Scheduel t_{urgent} and its fragmented data;
6) Update the lists of candidate and scheduled operations:

$T^{(n)}_{sched} := T^{(n-1)}_{sched} \cup \{t_{urgent}\}$;

$T^{(n+1)}_{cand} := T^{(n)}_{cand} - \{t_{urgent}\} \cup \left\{t' \in succ(t_{urgent}) \middle| pred(t') \subseteq T^{(n)}_{sched}\right\}$;

7) n : = n+1;

End while
End

RESULTS AND DISCUSSION

We have applied the RMFS-VDF heuristic to an example of the algorithm graph presented in Fig. 8 and an architecture graph composed of four processors and four buses. The failure rates of all the processors are all equal to 10^{-5} and the failure rate of the Buses SAMMP2, SAMMP1, SAMMP3 and SAMMP4 are respectively 10^{-6}, 10^{-6}, 10^{-5} and 10^{-4}.

Figure 9 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 multi-component architecture. The schedule length generated by this heuristic is 22.2. The GSFR of the non-reliable schedule $\lambda_{syndex} = 0.0000246$.

We apply our heuristic to the example of Fig. 9. The user requires the system to tolerate one bus failure, i.e., N = 1. Figure 10a shows the scheduled generated by our heuristic. The schedule length generated by our heuristic is 17.2. The GSFR of the non-reliable schedule $\lambda_{Nbf=1} = 0.00000597$.

Figure 10b shows the scheduled generated by our heuristic for N = 2. The schedule length generated by our heuristic is 15.2. The GSFR of the non-reliable

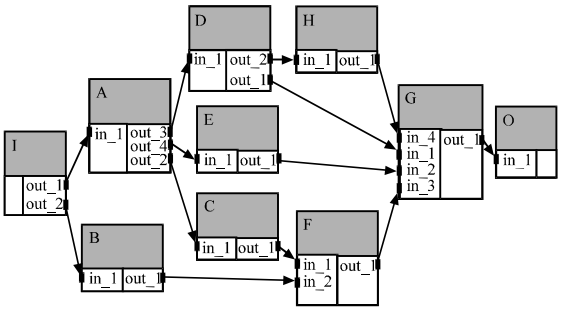

Fig. 8: Algorithm graph of example

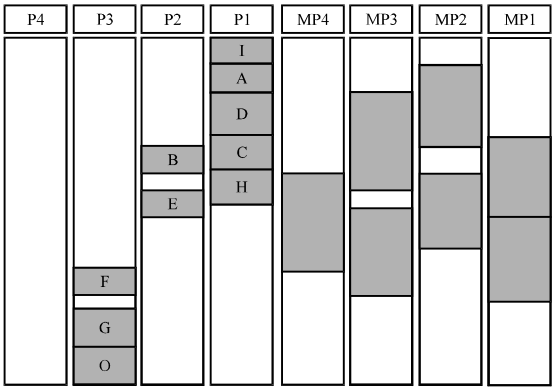

Fig. 9: Schedule generated by SynDex

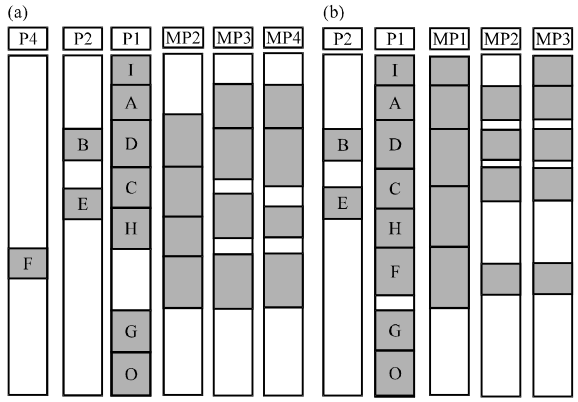

Fig. 10: Fault tolerant schedule; a) N = 1 and b) N = 2

schedule $\lambda_{N=2} = 0.00000613$. The important thing to note in Fig. 9 and 10 is that data fragmentation reduce the schedule length and improve significantly the reliability of the system.

Simulations: To evaluate our heuristic, we have applied the RMFS-VDF heuristic to a random algorithm graphs and a heterogeneous and completely connected architecture graph composed of 4 processors. Figure 11 and 12 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 RMFS-VDF.

Figure 11 shows the impact on schedule length obtained by RMFS-VDF for P = 4 and N = 1 (respectively, N = 2). As we can see, the schedule length grows almost linearly when CCR increases from 1-20.

Figure 12 shows the impact on the GSFR obtained by RMFS-VDF for P = 4 and N = 1 (respectively N = 2). As we can see, the GSFR decreases when CCR increases from 1-20.

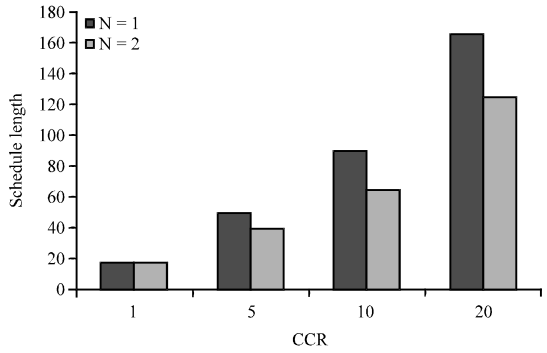

Fig. 11: Impact on schedule length of the CCR

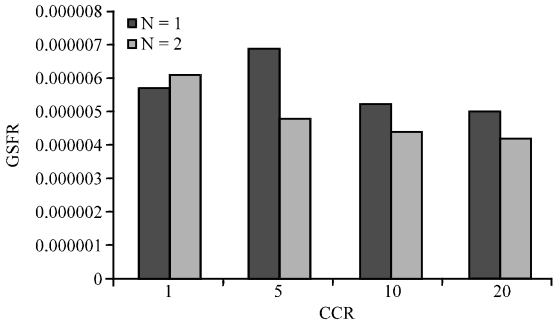

Fig. 12: Impact on GSFR of the CCR

CONCUSION

We have proposed in this study, a solution to tolerate communication media faults in distributed heterogeneous architectures with multiple-bus topology. The proposed solution, based on passive redundancy and variable data fragmentation strategies is a list scheduling heuristic called RMFS-VDF. It generates automatically, a distributed static schedule of a given algorithm onto a given multi-buses architecture which minimizes the system's run-time and tolerates upto N communication media faults with respect to real-time and distribution constraints.

The communication mechanism, based on variable data fragmentation, allows the fast detection and handling of errors. Simulations show that our approach can generally reduce the run-time overhead. Currently, we are working on a new solution for extending communication mechanism developed to sensor networks.

REFERENCES

Dima, C., A. Girault, C. Lavarenne and Y. Sorel, 2001. Off-line real-time fault-tolerant scheduling. Proceedings of the 9th Euromicro Workshop on Parallel and Distributed Processing, February 7-9, 2001, Mantova, pp: 410-417.

Dulman, S., T. Nieberg, J. Wu and P. Havinga, 2003. Trade-off between traffic overhead and reliability in multipath routing for wireless sensor networks. IEEE Wireless Commun. Networking, 3: 1918-1922.

Girault, A. and H. Kalla, 2009. A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate. IEEE Trans. Dependable Secure Comput., 6: 241-254.

Girault, A., H. Kalla and Y. Sorel, 2006. Transient processor/bus fault tolerance for embedded systems. Proceedings of the Working Conference on Distributed and Parallel Embedded Systems, October 11-13, 2006, Braga, Portugal, pp: 135-144.

Grunsteidl, G., H. Kantz and H. Kopetz, 2014. Communication Reliability in Distributed Real-Time Systems. In: Distributed Computer Control Systems 1991: Towards Distributed Real-Time Systems with Predictable Timing Properties, Kopetz, H. and M.G. Rodd (Eds.). Elsevier, London, UK., ISBN-13: 9781483299464.

Hanmer, R., 2013. Patterns for Fault Tolerant Software. John Wiley and Sons, New York, USA.

Izosimov, V., P. Pop, P. Eles and Z. Peng, 2012. Scheduling and optimization of fault-tolerant embedded systems with transparency/performance trade-offs. ACM Trans. Embedded Comput. Syst., Vol. 11, No. 3. 10.1145/2345770.2345773.

Jalote, P., 1994. Fault Tolerance in Distributed Systems. PTR Prentice Hall, Englewood Cliffs., ISBN-13: 9780133013672, Pages: 432.

Kandasamy, N., J.P. Hayes and B.T. Murray, 2005. Dependable communication synthesis for distributed embedded systems. Reliability Eng. Syst. Saf., 89: 81-92.

Kopetz, H., 2011. Real-Time Systems: Design Principles for Distributed Embedded Applications. 2nd Edn., Springer, New York, USA., ISBN-13: 9781441982377, Pages: 396.

Krishna, C.M., 2014. Fault-tolerant scheduling in homogeneous real-time systems. ACM Comput. Surv., Vol. 46, No. 4. 10.1145/2534028.

Pinello, C., L.P. Carloni and A.L. Sangiovanni-Vincentelli, 2004. Fault-tolerant deployment of embedded software for cost-sensitive real-time feedback-control applications. Proceedings of the Conference on Design, Automation and Test in Europe, Volume 2, February 16-20, 2004, Paris, France.

Pizza, M., L. Strigini, A. Bondavalli and F. di Giandomenico, 1998. Optimal discrimination between transient and permanent faults. Proceedings of the 3rd IEEE International High-Assurance Systems Engineering Symposium, November 13-14, 1998, Washington, DC., pp: 214-223.

Song, H. and H. Wu, 2010. The applied research of support vector machine in bus fault identification. Proceedings of the 6th International Conference on Natural Computation, Volum 3, August 10-12, 2010, Yantai, Shandong, pp: 1326-1329.