*International Journal of Computer Applications (0975 – 8887)*
*Volume 36– No.5, December 2011*

# Reliability-Driven Fault Tolerant Scheduling Heuristics for Distributed Embedded Real-Time Systems

Salim Kalla
Department of Computer Science, University of Batna
Algeria

Hamoudi Kalla
Department of Computer Science, University of Batna
Algeria

Chafik Arar
Department of Computer Science, University of Batna
Algeria

## ABSTRACT

We present a new scheduling approach to produce automatically a fault tolerant distributed schedule for critical distributed and real-time embedded systems. The approach that we propose take as input a set of operations (tasks), a target distributed architecture, some distribution constraints, some indications on the execution times of the operations on the processors of the target architecture, some indications on the communication times of the data-dependencies on the media communications of the target architecture, and the reliability of processors. IT produces a fault-tolerant distributed and static scheduling of the operations on the architecture, with an indication whether or not the real-time constraints are satisfied. The scheduling approach that we propose for architectures with multiple processors linked by a set of channels (links), consist of a list scheduling heuristic based on active replication strategy. In order to reduce the probability of fault occurrence, the replication process of each operation is based on a Global System Failure Rate (GSFR) objective function. Finally, simulation results show the performance of our approach.

## Keywords

Embedded systems, distributed systems, real-time systems, fault tolerance, transient faults, reliability, scheduling heuristics, active replication.

## 1. INTRODUCTION

Heterogeneous systems are being increasingly used in a major part of critical applications as transportations (aircrafts, automobiles …), nuclear, robotics and telecommunication. In these systems critical real-time constraints must be satisfied [1], where timing constraints which are not met may involve a system failure leading to a human, ecological, and/or financial disaster. One of the major problems of these systems is dependability [2, 3], where the malfunction or the failure of systems components (hardware or software) can lead to a catastrophe. The dependability of such real-time systems can be increased through hardware or software fault tolerance techniques, where a system built with fault tolerance capabilities will manage to keep operating in the presence of failures [4]. Hardware fault tolerance improves dependability of distributed real-time systems by adding extra hardware (processors, communications media, actuators, sensors) into the system [5]. However, hardware fault tolerance techniques are not preferred in most embedded systems due to the limited resources available because of weight, encumbrance, energy consumption (e.g., autonomous vehicles), radiation resistance (e.g., nuclear or space), or price constraints (e.g., consumer electronics). Critical embedded systems are increasingly use software fault tolerance to achieve the required dependability [6].

In this paper, we propose a new scheduling algorithm that generates a fault tolerant distributed static schedule, called Reliability-Driven Fault tolerant Scheduling heuristics (RDFS). This algorithm is different than the ones that we have proposed in [7, 8, 9] in the sense that we use *active replication of operations* and *reliability measures* to improve *both* the system's fault tolerance and the schedule length (and hence the system's run-time). The scheduling approach (Figure 1) that we propose for architectures with multiple processors linked by a set of channels (links), consist of a list scheduling heuristic based on two costs functions: the schedule pressure and the Global System Failure Rate (GSFR).

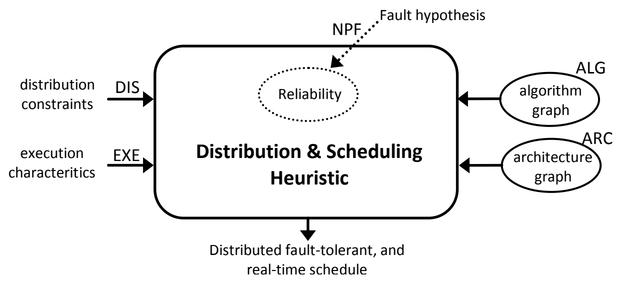

**Fig 1: Our approach**

The paper is structured as follows. Section 2 gives related work on fault tolerance. Section 3 presents the system and the reliability models. Section 4 proposes our RDFS scheduling algorithm; an example is detailed in Section 5. Section 6 gives performances evaluation of our algorithm, and finally the article concludes in Section 7.

## 2. RELATED WORK

In the past decade of fault tolerant systems research, an array of fault tolerant scheduling heuristics has been proposed [10, 11, 12, 13]. 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. In [14], Dogan et al. propose a scheduling heuristic algorithm for heterogeneous architectures that optimizes both the execution time and the reliability of the system. It is called Reliable Dynamic Level

Scheduling (RDLS) which is a variant of the Dynamic Level Scheduling algorithm (DLS) [15]. The authors add a reliability term to the cost function of DLS to take into account the reliability. The main difference between the RDLS heuristic and our algorithm is that the user can tolerate failures.

Several scheduling heuristics have been proposed to tolerate exclusively processor faults. They are based on active software redundancy [16, 9] or passive software redundancy [17, 18]. 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. [16] presents an off-line scheduling algorithm that tolerates a single processor faults in multiprocessor systems, while [9] 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, [18] presents a scheduling algorithm that tolerates one processor fault.

In [19], failures are tolerated using the fault recovery scheme and a primary/backups strategy. In [20], 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 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. [21].

The heuristics presented in this paper is our most recent work for integrating fault tolerance in the SynDEx tool (http://www-rocq.inria.fr/syndex), a system level CAD software tool for optimizing the implementation of real-time embedded applications on multi-components architectures. Prior work has been published in [22, 23, 24, 9].

## 3. MODELS

### 3.1 Algorithm

The algorithm is modeled by 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 corresponds to a data transfer from a producer operation to a consumer operation, defining a partial order on the execution of operations. This partial order relation is denoted by "$\rightarrow$". Based on this partial order, we say that o2 is a successor of o1 if (o1$\rightarrow$o2), and symmetrically that o1 is a predecessor of o2. We also say that o1 is the source of the data dependence (o1$\rightarrow$o2) and that o2 is its destination. An operation of ALG 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. The algorithm graph is executed repeatedly at each input event from the sensors in order to compute the output events for the actuators; each execution is called an iteration. Such a model is commonly used to specify periodic sampled systems, which constitute most of embedded control systems. For instance, it is used in many automotive and civil avionics applications.
Figure 2 is an example of an algorithm graph, with six operations v1, v2, v3, v4, v5 and v6. The data-dependencies between operations are depicted by arrows. For instance, the data-dependency v1$\rightarrow$ v4 corresponds to the sending of some arithmetic result computed by v1 and needed by v4.

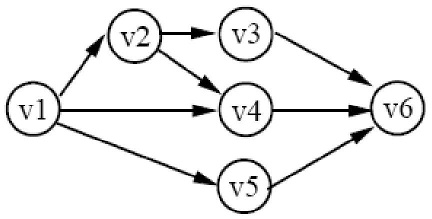

**Fig 2: Example of an algorithm graph ALG**

### 3.2 Architecture

The architecture is modeled by a graph, where each node is a processor, and each edge is a communication link. 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 six communications links L12, L23, L34, L14, L13 and L24.

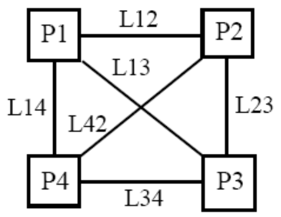

**Fig 3: Example of an architecture graph ARC.**

### 3.3 Execution

One of the characteristics of our real-time systems is that they are 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 RTC (the length of the schedule must be less than RTC), and a set of distribution constraints DIS (each one specifying that one operation of ALG cannot be executed on one processor of ARC). In our execution model EXE, we associate to each processor p a list of pairs (o, exe), where d is the worst case execution time (WCET) of the operation o on the processor p. Also, we associate to each communication link l a list of pairs (dpd, c), where c is the worst case transmission time (WCTT) of the data-dependency dpd on the communication link l. Since the target architecture is heterogeneous, the WCET (resp. WCTT) for a given operation (resp. data-dependency) can be distinct on each processor (resp. link). Finally, specifying the distribution constraints DIS amounts to associating the value "∞" to some pairs of p, (o,∞) meaning that o cannot be executed on p.

For instance, EXE for ALG and ARC of Figure 2 and Figure 3 is given in Figure 4 and Figure 5.

| | | operation | | | | | |
|---|---|---|---|---|---|---|---|
| | time | v1 | v2 | v3 | v4 | v5 | v6 |
| proc. | P1, P2 | 8 | 4 | 3 | 2 | 2 | 6 |
| | P3, P4 | 10 | 6 | 5 | 4 | 4 | 8 |

**Fig 4: Execution times.**

| | | data-dependency | | | |
|---|---|---|---|---|---|
| | time | v1 ▷ v2 | v1 ▷ v4<br>v1 ▷ v5 | v2 ▷ v3<br>v2 ▷ v4 | v3 ▷ v6<br>v4 ▷ v6<br>v5 ▷ v6 |
| link | L12, L14, L13 | 2 | 3 | 4 | 5 |
| | L23, L24, L34 | 4 | 6 | 8 | 10 |

**Fig 5: transmission times.**

## 3.4 Fault Model and Problem Definition

We assume only hardware components (processors and communication links) 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 assume that at most NPF (Number of Processor Failures) processor faults can arise in the system. The failure of a processor has an exponential distribution [25], i.e., it follows a Poisson law with a constant failure rate λ. Furthermore, components failures are assumed to be independent. For instance, Figure 6 gives the failure rates of the processors and communication links of the architecture of Figure 3.

| | processors | | communication links | |
|---|---|---|---|---|
| | P1, P2 | P3, P4 | L12, L14, L13 | L23, L24, L34 |
| $\lambda$ | $10^{-5}$ | $10^{-6}$ | $10^{-3}$ | $10^{-4}$ |

**Fig 6: Failure rates for system components.**

Our goal is to produce automatically a fault-tolerant distributed static schedule of a given algorithm ALG onto a given distributed architecture ARC, which must satisfy a set of constraints. Actually, we want to solve the following problem:

Given:

- a distributed heterogeneous architecture ARC composed of a set P of processors and a set L of links: P ={…, pi,…}, L ={…, Lj,…}
- an algorithm ALG composed of a set O of operations and a set E of data-dependencies: O = {…, oi, …, o_j,…}, E = { …,(oi→ oj), …}
- all the execution characteristics EXE of the software components of ALG onto the hardware components of ARC,
- several distribution constraints DIS (if any),
- a real-time constraint RTC,
- a number NPF of processor faults that may affect the system, with NPF<|P|,

the problem is to find a distributed static schedule of ALG onto ARC, which minimizes the system's run-time (makespan), maximizes system reliability and tolerates NPF processors with respect to RTC, EXE, and DIS.

The makespan, noted by $RT_{sched}$, is the end execution time of the operation that is completed last among all operations. It is computed as follows:

$$\mathcal{R}t_{sched} = \max_{p_j}\left\{\max_{o_i\ on\ p_j} end(o_i, p_j)\right\}$$

where end($o_i$, $p_j$) is the time at which operation $o_i$ terminates its execution on processor $p_j$.

## 4. THE PROPOSED ALGORITHM

Our solution is based on active redundancy and on a global system failure rate (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 the NPF+1 replicas of each operation from the beginning to the end of the schedule.

The GSFR of scheduling NPF+1 replicas of an operation oi, noted $\Lambda(S_{oi})$, is computed by the following equation:

$$\Lambda(S_{o_i}) = \frac{-\log R(S_{o_i})}{U(S_{o_i})} \qquad (1)$$

The reliability $R(S_{oi})$ is computed, for each operation oi and each processor p, by the following equation:

$$R(S_{o_i}) = \prod_{k=1}^{\mathcal{N}pf+1} e^{-\lambda_{p_k} exe(o_i^k, p_k)} \qquad (2)$$

The total utilization of the hardware resources $U(S_{oi})$ is computed, for each operation oi and each processor p, by the following equation:

$$U(S_{o_i}) = \sum_{k=1}^{\mathcal{N}pf+1} exe(o_i^k, p_k) \quad (3)$$

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 NPF processors, with respect to the real-time constraint RTC and the distribution constraints DIS. At each step of the greedy list scheduling heuristic, GSFR and schedule pressure are used as a cost function to select the best operation to be scheduled.

The schedule pressure function, noted $\sigma^{(n)}$, is computed, for each operation oi and each processor pj, as follows:

$$\sigma^{(n)}(o_i, p_j) := S^{(n)}_{o_i,p_j} + \overline{S}^{(n)}_{o_i} - R^{(n-1)} \quad (4)$$

where $S^{(n)}_{oi,pj}$ is the earliest time at which operation $o_i$ can start its execution on processor $p_j$ (computed in absence of faults), $\overline{S}^{(n)}_{oi}$ is the latest start time from end of oi (defined to be the length of the longest path from the outputs operations to oi, and $R^{(n-1)}$ is the critical path length at step (n-1) (that is, before scheduling oi. The schedule pressure measures how much the scheduling of an operation lengthens the critical path of the algorithm in the absence of processors failures. For more details, please refer to [23].

The algorithm is divided into four steps: initialization step, selection step, distribution and scheduling step, and finally an update step. The superscript number in parentheses refers to the steps of the heuristic, e.g., $O^{(n)}_{sched}$.

**The RDFS Algorithm**

```
------------------------------------------------------------------------------
Inputs = ALG, ARC, NPF, EXE, RTC, Rel and DIS;
Output = a fault-tolerant multiprocessor static schedule;
Initialization Step
-   Compute the set $\Psi$ of all arbitrary combinaison of NPF+1
    processors;
-   Initialize the sets of candidate operations $O_{cand}$ and
    scheduled operations $O_{sched}$
      $O^{(1)}_{cand}$ := {operations of ALG without predecessors};
      $O^{(1)}_{sched} := \phi$;
Distribution and Scheduling Loop
While $O^{(n)}_{cand} \neq \phi$ do
  Selection
   - select for each candidate operation $o_{cand}$ of $O^{(n)}_{cand}$ a set
     $P_{best} \subset \Psi$ of NPF+1 processors that minimizes the schedule
     pressure function such that $\Lambda(S_{oi}) < Rel$;
   - select for each candidate operation $o_{cand}$ of $O^{(n)}_{cand}$, among
     the processors $P_{best}(o_{cand})$, the best processor $P_{best}$ that
     maximizes the schedule pressure function;
   - select, among all the pairs ($o_{cand}$, $P_{best}$), the best pair
     ($o_{best}$,$P_{best}$) that maximizes the schedule pressure function;
  Distribution and Scheduling
   - Let $P_{best}(o_{best})$ be a best set of NPF+1 processors of $o_{best}$
     computed at the selection step;
   - Schedule each replica $o^k_{best}$ on the processor $P^k_{best}$ of
     $P_{best}(o_{best})$.
  Update Sets
   - Update the sets of candidate and scheduled operations for
     the next step (n+1):
      $O^{(n+1)}_{sched} := O^{(n)}_{sched} \cup \{o_{best}\}$;

      $O^{(n+1)}_{cand} := O^{(n)}_{cand} - \{o_{best}\} \cup \{o_{new}$ successors of $o_{best}$
              $| \{$predecessors of $o_{new}\} \subset O^{(n+1)}_{sched}\}$;
end While
------------------------------------------------------------------------------
```

**End of the algorithm**

The initialization step involves initializing the list of candidate operations $O^{(1)}_{cand}$ with the operations without predecessor. Later, an operation is said to be a candidate if all its predecessors are already scheduled. Moreover, the list of scheduled operations $O^{(1)}_{sched}$ is initially empty.

In the selection step, for each candidate operation $o_{cand}$ of $O^{(n)}_{cand}$, a set $P_{best}$ of NPF+1 processors is selected among all the processors of P to schedule NPF+1 replicas of $o_{cand}$. The rule of selection is based on the schedule pressure function and the GSFR function. 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 set $P_{best}$ of each candidate operation $o_{cand}$ is composed from NPF+1 processors that minimize the schedule pressure, i.e., processors that minimize the length of the schedule. Then, among all these candidates operations, the most urgent candidate $o_{best}$, with a processor $p_{best}$ of $P_{best}(o_{best})$ that maximizes this function, is selected to be placed and scheduled.

The distribution and scheduling step involves first replicating the best candidate $o_{best}$ into NPF replicas, and second scheduling each replica $o^k_{best}$ of $o_{best}$ respectively onto the processor $p^k_{best}$ of $P_{best}$.

The updating step involves updating the list of scheduled operations and then the list of candidates operations. The scheduled operation $o_{best}$ is removed from $O^{(n)}_{cand}$, while the new operations added to $O^{(n)}_{cand}$ are the operations of ALG which have all their predecessors in the new list of scheduled operations.

## 5. AN EXAMPLE

We have implemented our RDFS heuristic within the SynDEx tool [26], a system level CAD software tool for optimizing the implementation of real-time embedded applications on multi-component architectures. To illustrate the principles of our heuristic, we apply it to the example of Figure 2 for ALG and Figure 3 for ARC. The execution characteristics are specified by Figure 4 and Figure 5. The user requires the system to tolerate one processor failure, i.e., NPF=1.

We obtain the fault tolerant schedule presented in Figure 7 (a screen capture from SynDEx). Each operation of the algorithm graph is replicated twice and these replicas are assigned to different processors; furthermore, each replica receives its inputs twice and from disjoint paths.

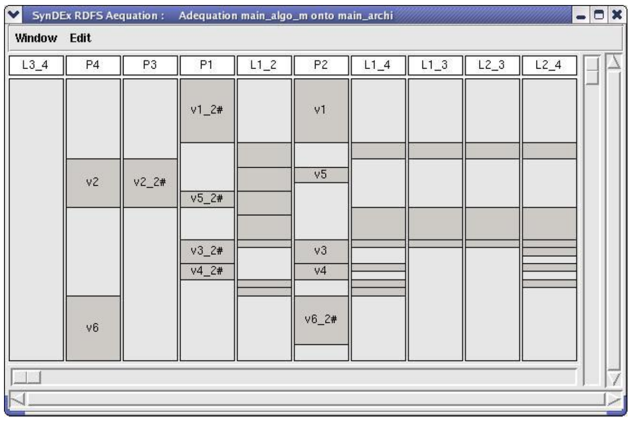

**Fig 7: A final fault tolerant schedule of ALG on ARC.**

## 6. SIMULATIONS

To evaluate our heuristic, we have applied the RDFS heuristic to a set of random algorithm graphs and a heterogeneous and completely connected architecture graph composed of 4, 5 and 6 processors. The following figures have been obtained by generating randomly 50 ALG graphs of 10,20,…,100 operations each, with a CCR set to 0.1, 1, 10. 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 advantage of this method is that the randomly generated algorithm graphs are as close as possible to real applications: in particular, the nodes are not generated randomly but within a square matrix whose height and width depend on the total number of nodes N.

The general objective of our simulations is to study the impact of the number of processors P, number of operations N, and CCR on the schedule length and reliability introduced by RDFS and the fault tolerance heuristic AAA-TP presented in [9].

Figure 8 shows the impact on schedule length obtained by AAA-TP and RDFS with respect to GSFR, averaged over 50 ALG graphs, for each N, P=4, NPF=2, and CCR=1. As we can see, the schedule length grows almost linearly when N increases from 10 to 100.

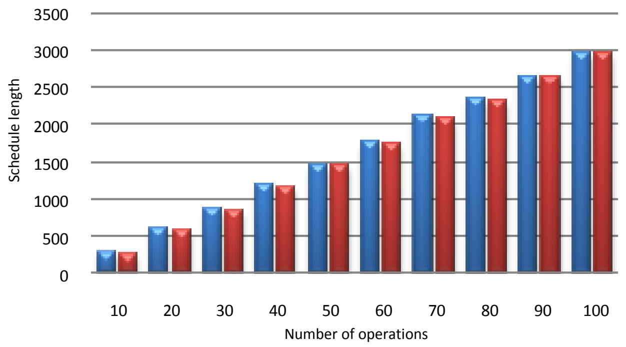

**Fig 8: Impact on schedule length of the number of operations for CCR=1 and P=4.**

Figure 9 shows the impact on reliability obtained by RDFS and AAA-TP with respect to GSFR, averaged over 50 ALG graphs, for each N, P=4, NPF=2, and CCR=1. As we can see, RDFS performs better than AAA-TP.

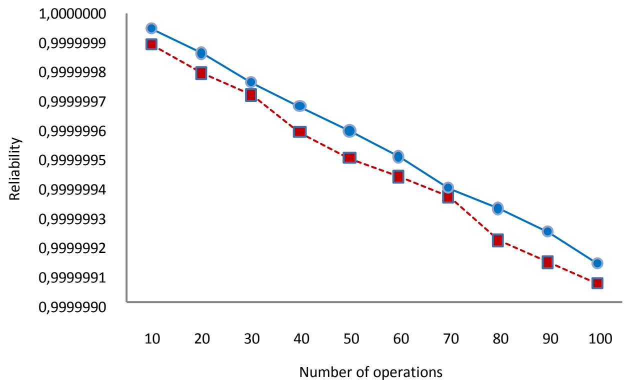

**Fig 9: Impact on reliability of the number of operations for CCR=1 and P=4.**

Figure 10 shows the impact on schedule length obtained by RDFS and AAA-TP with respect to GSFR, averaged over 50 ALG graphs, for each N=50, NPF=2, and CCR=1.

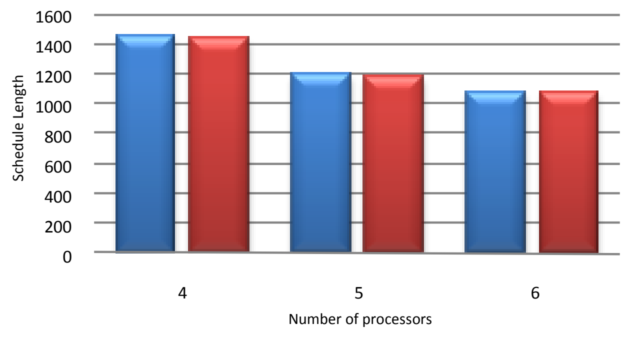

**Fig 10: Impact on schedule length of number of processors for: CCR=1 and N=50.**

Figure 11 shows the impact on schedule length obtained by RDFS and AAA-TP with respect to GSFR, averaged over 50 ALG graphs, for N=50, P=4, and NPF=2.

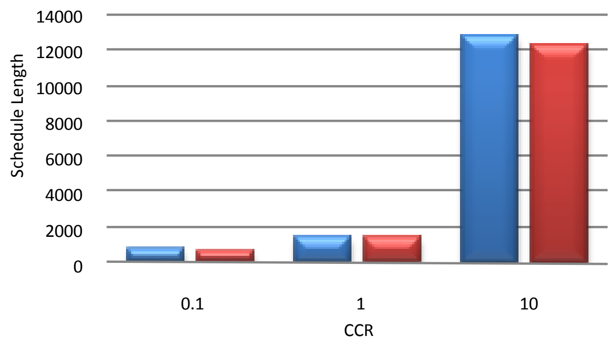

**Fig 11: Impact on schedule length of CCR for: P=4 and N=50.**

Figure 11 shows that, when the average communication time is strictly greater than the average execution time, the schedule length increase. This is due to the communication overhead.

In this simulation, RDFS outperforms AAA-TP, and AAA-TP is less efficient for the reliability.

## 7. CONCLUSION

The literature about fault-tolerance of distributed and/or embedded real-time systems is very abundant. Yet, there are few attempts to combine fault-tolerance and automatic generation of distributed code for embedded systems. In this paper, we have studied this problem and proposed a software implemented fault-tolerance solution based on GSFR.

We have proposed a new scheduling heuristic, called RDFS, which produces automatically a static distributed fault-tolerant schedule of a given algorithm ALG on a given distributed architecture ARC. Our solution is based on the software redundancy of both the computation operations and the communications. All replicated operations send their results but only the one which is received first by the destination processor is used; the other results are discarded. The implementation uses a scheduling heuristic for optimizing the critical path of the distributed algorithm obtained. SynDEx is able to generate automatically executable distributed code, by first producing a static distributed schedule of a given algorithm on a given distributed architecture, and then by generating a real-time distributed executive implementing this schedule. Experimental results show that RDFS performs better than AAA-TP. Finally, we plan to experiment our method on an electric autonomous vehicle, with a 5-processor distributed architecture.

## 8. REFERENCES

[1] Rushby, J. Critical System Properties: Survey and Taxonomy Reliability Engineering and Systems Safety, 1994, 43, 189-219

[2] Suri, N. & Ramamritham, K. Editorial: Special Section on Dependable Real-Time Systems IEEE Trans. on Parallel and Distributed Systems, 1999, 10, 529-531

[3] Avizienis, A.; Laprie, J. & Randell, B. Fundamental Concepts in Dependability 3rd IEEE Information Survivability Workshop, ISW'00, 2000, 7-12

[4] Jalote, P. Fault-Tolerance in Distributed Systems Prentice-Hall, 1994

[5] Kopetz, H. & Bauer, G. The Time-Triggered Architecture Proceedings of the IEEE, 2003, 91, 112-126

[6] Torres-Pomales, W. Software Fault Tolerance: a Tutorial 2000

[7] Girault, A. & Kalla, H. A Novel Bicriteria Scheduling Heuristics Providing a Guaranteed Global System Failure Rate IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, 2009, 6, 241-254

[8] Assayad, I.; Girault, A. & Kalla, H. A Bi-Criteria Scheduling Heuristics for Distributed Embedded Systems Under Reliability and Real-Time Constraints International Conference on Dependable Systems and Networks, DSN'04, IEEE, 2004, 347-356

[9] Girault, A.; Kalla, H.; Sighireanu, M. & Sorel, Y. An Algorithm for Automatically Obtaining Distributed and Fault-Tolerant Static Schedules International Conference on Dependable Systems and Networks, DSN'03, IEEE, 2003

[10] Chen, H.; Luo, W.; Wang, W. & Xiang;, J. A novel real-time fault-tolerant scheduling algorithm based on distributed control systems International Conference on Computer Science and Service System, 2011

[11] Chen, J.; Yang, C.; Kuo, T. & Tseng;, S. Real-Time Task Replication for Fault Tolerance in Identical Multiprocessor Systems 13th IEEE Symposium on Real Time and Embedded Technology and Applications, 2007

[12] Gan, J.; Gruian, F.; Pop, P. & Madsen, J. Energy/reliability trade-offs in fault-tolerant event-triggered distributed embedded systems 16th Asia and South Pacific on Design Automation Conference, 2011

[13] Jinyong, Y.; Hanguang, S.; Li, Y. & Qiangqiang, C. A Real-time Fault-tolerant Scheduling Algorithm for Software/Hardware Hybrid Tasks International Conference

on Mechatronic Science, Electric Engineering and Computer, 2011

[14] Dogan, A. & Özgüner, F. Matching and Scheduling Algorithms for Minimizing Execution Time and Failure Probability of Applications in Heterogeneous Computing IEEE Trans. on Parallel and Distributed Systems, 2002, 13, 308-323

[15] Sih, G. & Lee, E. A Compile-Time Scheduling Heuristic for Interconnection Constraint Heterogeneous Processor Architectures IEEE Trans. on Parallel and Distributed Systems, 1993, 4, 175-187

[16] Hashimoto, K.; Tsuchiya, T. & Kikuno, T. Effective Scheduling of Duplicated Tasks for Fault-Tolerance in Multiprocessor Systems IEICE Trans. on Information and Systems, 2002, E85-D, 525-534

[17] Ahn, K.; Kim, J. & Hong, S. Fault-Tolerant Real-Time Scheduling using Passive Replicas Pacific Rim International Symposium on Fault-Tolerant Systems, PRFTS'97, 1997

[18] Qin, X.; Jiang, H. & Swanson, D. An Efficient Fault-tolerant Scheduling Algorithm for Real-time Tasks with Precedence Constraints in Heterogeneous Systems International Conference on Parallel Processing, ICPP'02, 2002, 360-386

[19] Gummadi, K.; Pradeep, M. & Murthy, C. R. An Efficient Primary-Segmented Backup Scheme for Dependable Real-Time Communication in Multihop Networks IEEE/ACM Trans. on Networking, 2003, 11

[20] Dima, C.; Girault, A.; Lavarenne, C. & Sorel, Y. Off-Line Real-Time Fault-Tolerant Scheduling 9th Euromicro Workshop on Parallel and Distributed Processing, PDP'01, 2001, 410-417

[21] Pinello, C.; Carloni, L. & Sangiovanni-Vincentelli, A. Fault-Tolerant Deployment of Embedded Software for Cost-Sensitive Real-Time Feedback-Control Applications Design, Automation and Test in Europe, DATE'04, IEEE, 2004

[22] Girault, A.; Kalla, H. & Sorel, Y. Transient Processor/Bus Fault Tolerance for Embedded Systems IFIP Working Conference on Distributed and Parallel Embedded Systems, DIPES'06, Springer, 2006, 135-144

[23] Girault, A.; Kalla, H. & Sorel, Y. A Scheduling Heuristics for Distributed Real-Time Embedded Systems Tolerant to Processor and Communication Media Failures International Journal of Production Research, 2004, 42, 2877-2898

[24] Girault, A.; Kalla, H. & Sorel, Y. An Active Replication Scheme that Tolerates Failures in Distributed Embedded Real-Time Systems IFIP Working Conference on Distributed and Parallel Embedded Systems, DIPES'04, Kluwer Academic Publishers, 2004

[25] Shatz, S.; Wang, J. & Goto, M. Task Allocation for Maximizing Reliability of Distributed Computer Systems IEEE Trans. on Computers, 1992, 41, 1156-1168

[26] Grandpierre, T. & Sorel, Y. From Algorithm and Architecture Specifications to Automatic Generation of Distributed Real-Time Executives: A Seamless Flow of Graphs Transformations International Conference on Formal Methods and Models for Codesign, MEMOCODE'03, IEEE, 2003.