BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **18**, No 3
Sofia • 2018 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2018-0038

A Fault Tolerant Scheduling Heuristics for Distributed Real Time Embedded Systems

Bachir Malika[1], *Hamoudi Kalla*[2]

[1]*Computer Science Department, University of Batna 2, 05000 Algeria*
[2]*LaSTIC Laboratory, REDS Tem, University of Batna 2, 05000 Algeria*
E-mails: malika_klm@yahoo.fr hamoudi.kalla@gmail.com

Abstract: *In this paper, fault tolerant task scheduling algorithms are proposed for mapping task graphs to heterogeneous processing nodes. These scheduling heuristics that we propose are redundancy-based software to tolerate hardware faults. We consider only processor permanent failures with a fail-silent behavior. The proposed heuristics generate automatically a real-time fault distributed schedule of dependent and independent tasks into heterogonous multiprocessors architecture. The heuristics are based on active and passive redundancy.*

Keywords: *Real-time embedded systems, scheduling algorithms, fault tolerance, active and passive replication.*

1. Introduction

Embedded real-time systems are now present in all areas of everyday life, from public domain applications (consumer electronics, automotive, …) to critical applications (space, nuclear, …) where a system failure can cause catastrophic consequences (loss of time, money or worse – loss of human life) [2]. Since faults are inevitable and can appear at any time, so systems must be dependable [3]. Dependability is defined as property which allows its users to place confidence in the service that it delivers to them [4]. Several methods are proposed in the literature to guarantee it. Fault tolerance is the most used (or the best) one. Fault tolerance allows the system to continue to deliver the same service expected in the presence of failures [5]. As failures can be caused by both hardware and software, in this paper we concentrate on hardware faults and exactly processor faults.

Redundancy is one of methods used to achieve fault tolerance, it can be active, passive or both. The active replication consists in executing the same task in parallel on several distinct processors [6]. The passive replication consists in replicating each task on n replicas, and only one of the n replicas executes, called primary, and the $n – 1$ other replicas are waiting and are executed only when the primary fails [6]. In real-time systems, fault tolerance is provided by physical, and/or software redundancy. As we are targeting embedded systems, we rely on the distributed architectures which allow redundancy of software components on different processors to meet their specifications.

Faults can be classified according to their duration as: Permanent faults, intermittent faults or transient faults. Permanent faults are persistent, they continue to exist until the faulty component is repaired or replaced. These faults can be caused by catastrophic system failures such as processor failures. Transient faults arise once and then vanish. For example, a network message does not reach its destination but later the message is successfully retransmitted. Intermittent faults are characterized by a fault happening, then vanishing, then happening again, then vanishing again, etc. They are difficult to be defined, but their effects are strongly correlated. A loose connection is an example of this kind of fault [15]. We concentrate on permanent faults of one processor.

In this paper, we investigate the integration of fault-tolerance in real-time distributed embedded systems by starting from a data-flow algorithm and a heterogeneous distributed architecture with multi-point links. Our goal is to produce automatically a fault-tolerant distributed schedule of the algorithm onto the architecture. The faults considered are single processor permanent failures with a fail-silent behavior. To achieve this, we present two new heuristics based on the software redundancy, which generate a static fault tolerant schedule. By taking into account the execution duration of all tasks on all processors, and the communication durations of all data-dependencies on communication links, we are able to compute the total execution time of the obtained schedule, both in the presence and in the absence of failures. The heuristics proposed, which are an extension of the Algorithm Architecture Adequation (AAA) of the SynDEx tool [22, 10], try to find solutions that satisfy real-time, distribution and fault-tolerance constraints.

The work is organized as follows: In Section 2 the related works are briefed, in Section 3 the problem specification of the system are described and the proposed methodologies are discussed in Section 4. Experimental evaluation with respective observation is depicted in Sections 5 and 6, finally conclusion is made in Section 7.

2. Related works

A large number of fault tolerant scheduling algorithms for real-time embedded systems have been proposed in the literature. In [11], authors use software redundancy solutions based on both active and passive backup copies to tolerate only one bus fault in multi-bus heterogeneous architectures. In [12], they overcome the faults in the sensor network and distribute the provided task rationally by using a technique called P/B, it uses passive backup copies overlapping methodology to monitor the mode of backup copies adaptively through scheduling primary copies early and backup copies delayed. In [13], authors presented a scheduling algorithm for tolerating Weibull distributed failures of grid resources in spite of commonly adopted assumption of Poisson failure distribution. The algorithm is applied for dependent and independent tasks, it uses rollback recovery via checkpoint/restart for improving system dependability and reliability. In [14], authors proposed scheduling algorithms for tolerating permanent and transient failures in real-time embedded systems. These algorithms attempt to provide low-cost solutions for fault tolerance, graceful performance degradation, and load shedding in such systems by exploiting trade-offs between space and time redundancy. They place tolerant scheduling algorithms fault in three categories: dynamic scheduling, offline or static planning and programming inaccurate calculations.

Another mechanism of fault tolerance is presented by K. Hashimoto, T. Tsuchiya and T. Kikuno [20]. This mechanism makes it possible to tolerate the faults of a single processor by using the active duplication of software components. Their algorithm, called HBP partitions all tasks as a first step into groups according to their size and in a second time, it calls a basic algorithm that allows to order and duplicate each task of each group on condition that each task is scheduled after its predecessors in the same processor and its replica in a different processor. Y. Oh and S. H. Son [21] proposed a fault tolerance scheduling heuristic called 1TFT for independent and non-preemptive tasks. Heuristics are an off-line scheduling that assures the deadlines of each task, even in the presence of arbitrary failures of a single processor based on passive redundancy. H. Kalla [1] presents a heuristic called AAA-TB, based on Hybrid redundancy to tolerate arbitrary faults in processors and communication buses in a reactive system. Hybrid redundancy consists, in this case, on the one hand replicate the operations of the software architecture into multiple actively placed copies on multiple separate processors, and on the other hand to replicate data dependencies in multiple replicas of which only one executes and the others remain inactive until the appearance of bus failures or processor implanting the primary copy.

3. Specifications of the problem

A hard-real-time system is modeled by software architecture (the set of tasks and messages), a hardware architecture (physical runtime support) and a faults' models.

3.1. Algorithmic specification

Our algorithm is modelled by a data flow graph [7], which is an oriented and valued hyper graph called algorithm graph. The set of nodes $T= \{t_i, t_j, \ldots, t_n\}$ corresponds to the tasks components of the system, and the set of edges $C \subseteq T \times T$ represents the data-dependences between tasks (messages exchanged by the tasks by intra-processor or inter-processor communication). A task is distinguished in the algorithm graph by two types: the computation task and the input/output task [1].

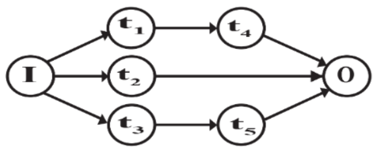


Fig. 1. Example of an algorithm graph

Our algorithm also can represented by a hyper graph without edges called algorithm graph without precedence constraints (data dependencies).

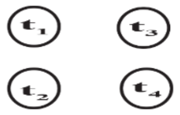


Fig. 2. Algorithm graph without data dependencies

3.2. Hardware specification

The hardware specification is a heterogeneous multiprocessor architecture (processors have different characteristics). It is presented by an undirected graph whose nodes denote processors and edges denote physical links between processors, we considered a multipoint network (bus).

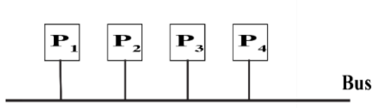


Fig. 3. A multipoint connexion

3.3. Faults models

The failure constraints give assumptions about the maximum number of processor faults that the system can tolerate. In this paper, we assume only permanent faults of only one processor with a fail-silent behaviour, i.e., either it works and gives a result, or it does not work so does not give any result. We assume also that the communication bus is fault-free, i.e., reliable.

3.4. Distribution, execution time and real-time constraints

The distribution constraints [8] are the hardware preferences. They define the exclusion relationship between some hardware and some software components. As the architecture is heterogeneous, the execution time of each task can vary from one processor to another. The temporal constraints [8] define the worst execution times and the worst communication times respectively of the tasks and data communication on the architectures' components (processors and communication bus).

4. Proposed methodologies

4.1. Methodology AAA-FAULT^{IDT}

The objective of this methodology is to optimize the automatic generation of distribution and scheduling in real-time of independent tasks and also to tolerate permanent faults of a single processor. Fault tolerance is achieved in this case into two phases using hybrid of active and passive redundancy. The first step is to transform the algorithm graph ALG of a set of independent tasks [9] on a new graph ALGN in which each task is replicated into two copies (primary and backup), linked by a conditional dependency [16, 17]. The second phase consists on placing this new redundant graph on the hardware architecture where every replicated task is actively executed (run both) on two different processors. At the end of execution of the primary copy, this last one sends a message to her replica to ignore her execution in order to reduce the overhead on the global time of system execution. This heuristic benefits the both advantages of passive and active replication [18]. This methodology consists of three phases: transformation phase, matching phase and the fault tolerance phase.

- The transformation phase: In this phase we transform the non-redundant graph algorithm ALG into a new redundant graph ALGN, where:
  - Each task is replicated into two copies, primary T_i^{p} and secondary T_i^{b};
  - A new dependency is added to the new ALGN between each task and its backup ($T_i^{\mathrm{p}} \rightarrow T_i^{\mathrm{b}}$), there time execution is null;
- The matching phase: The proposed heuristic maps the new graph ALGN and the hardware architecture ARC to achieve an optimal schedule.
- The fault tolerance phase: consists in detecting the processor failure and then updating the scheduling of its tasks.

In active redundancy all redundant tasks are in operation and are sharing the load with the main task. Upon failure of one task, the surviving tasks carry the load. The redundant or back-up tasks in passive or standby, systems start operating only when one or more fail, the back-up tasks remain dormant until needed.

Our methodology is based on the combination of the two redundancies. For each new system execution, the active replication is applied. After a time *T*, each replica receives a message of its primary copy indicating its successful execution; in this case, execution of these replicas is stopped until next cycle or a random stop of their primary tasks, which is the mechanism of passive replication.

In this methodology, error detection is not necessary. In case of failure of a processor, replicas of tasks located there will not receive a message from their primary copies, while pursuing their implementation and cover this failure.

The principle of this methodology is presented as follows:

```
Begin
  Active execution of all tasks (primary and secondary) by order of execution
  For each primary task & after time $t$
          $T_i^{\mathrm{p}}$ send message to its reply $T_i^{\mathrm{b}}$
          $T_i^{\mathrm{b}}$ ignore its execution and locks
  If $T_i^{\mathrm{b}}$ did not receive this message then
          $T_i^{\mathrm{b}}$ continue execution and mask the failure of $T_i^{\mathrm{p}}$
end
```

This heuristic profit the both advantages of passive and active replication. In fact, it has small recovery delay after failures and small execution delay without failures. The following example illustrates the effect of this methodology:

Taking software architecture ALG (Fig. 4) which consists of three independent tasks: t_1, t_2 and t_3, and hardware architecture ARC (Fig. 5) composed of three heterogeneous processors connected via a bus. Runtimes of tasks on these processors are respectively: (1, 3), (2, 1) and (2, 4). ALG is transformed into ALGN in which each node is replicated in two copies with its execution times on different processors, the communication duration between nodes is null.

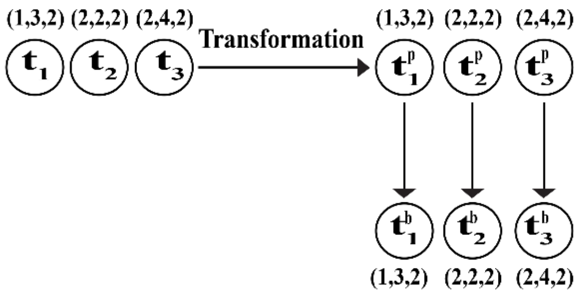


Fig. 4. Software architecture

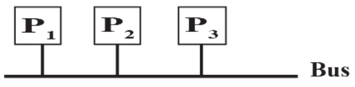


Fig. 5. Hardware architecture

After applying the AAA-FAULT^{IDT} methodology on this specification, before and after the occurrence of a failure of one processor, we obtained the results summarized in Figs 6 and 7.

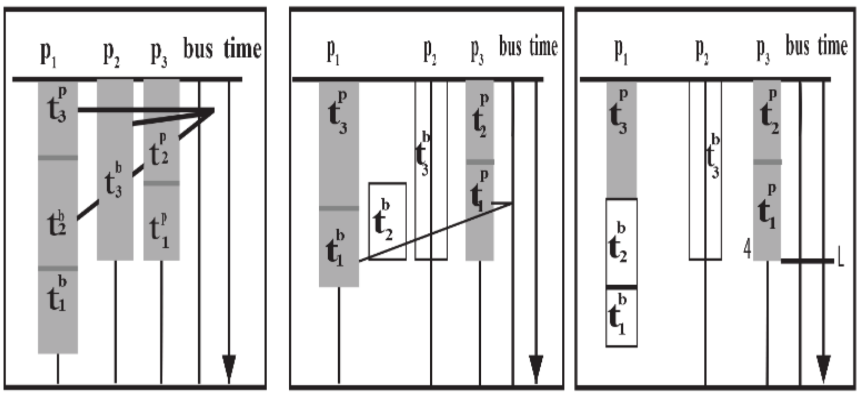


Fig. 6. Distributed schedule with absence of failure

Upon failure, $p_1$for example, t_3^b does not receive a message from t_3^p , then it continues to execute and mask its failure. Thus distributed schedule is shown in Fig. 7.

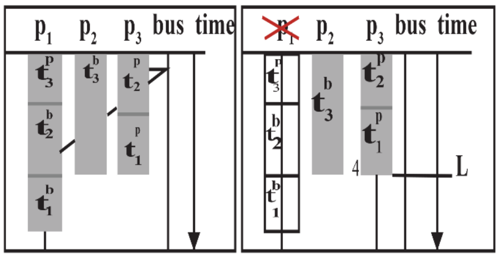


Fig. 7. Distributed schedule with failure of p_1

4.2. Methodology AAA-FAULT^{DT}

In this methodology, fault tolerance is based on the passive redundancy which requires a mechanism for detecting errors [19]. Therefore, we insert in ALG a new task called watchdog and denoted *w*. this methodology consists of three phases: transformation phase, matching phase and the fault tolerance phase.

- Transformation phase: In this phase we transform the non-redundant graph algorithm ALG into a new redundant graph ALGN, where:
  - Each task is replicated into two copies, primary t_i^{p} and secondary t_i^{b};
  - A new task w_i is added between each primary task t_i^{p} and its replica t_i^{b};
  - Three new dependencies are added to the new ALGN: ($t_i^{\mathrm{p}} \rightarrow w_i$), ($w_i \rightarrow t_i^{\mathrm{b}}$) and ($t_i^{\mathrm{b}} \rightarrow t_j$); where t_j is the successor of t_i.

The new dependencies are of two types: two data dependencies and a conditional dependence, one of the data dependencies transfers a signal from the primary task t_i^{p}indicating its execution to the task w_i and the other one transfers the result of the secondary task t_i^{b} to its successor t_j. The conditional dependence which transfers awakening message since the task *w* towards the replica of each task, it is called conditional dependence as its execution depends on the state of the processor (faulty or not).

- The matching phase: the proposed heuristic maps the new graph ALGN and the hardware architecture ARC to achieve an optimal schedule.
- The fault tolerance phase: consists in detecting the processor failure and then updating the scheduling of its tasks.

Passive redundancy requires an error detection mechanism; we propose a new task to be created called watchdog and denoted *w* to be added as a successor to each task in the architecture graph. The role of the task *w* is to receive a signal indicating the execution of its predecessor (task t_i). So, after some time Δ if no signal is detected then the processor p_i that executes the task t_i is down, therefore it wakes up the secondary copy of the task t_i located on another processor by sending a wake-up message through the bus or via an intra-processor communication.

The body of the task *w* has the following form:

```
If  ($w_i$ does not receive a signal of $t_i^{p}$ after time $\Delta$) then
    Send a wake-up message to the replica $t_i^{b}$ to update scheduling. This message
    also applies to other replicas located on $p_i$ .
End if
```

The example below shows a distribution/scheduling algorithm of an algorithm graph on an architecture graph in which tasks (respectively communications) are represented by boxes whose height is proportional to their execution time (resppectively of communications).

Let be the following heterogeneous material architecture: $P = \{P_1, P_2, P_3\}$, L = {Bus}, and the algorithm graph architecture ALG transformed into ALGN in which each node is associated with its execution times on different processors, and each arc is associated with the communication duration between nodes that it links (Fig. 8).

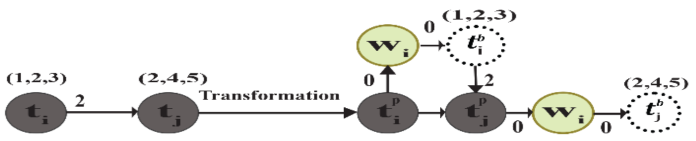


Fig. 8. Transformation scheme of ALG

The fault-tolerant distribution/scheduling is outlined as follows, where p_i^{v} (virtual processor) which indicate the passive placement of the replicas of the tasks awaiting activation on processor p_i, and L is the length of distribution/scheduling with absence of failures (Fig. 9).

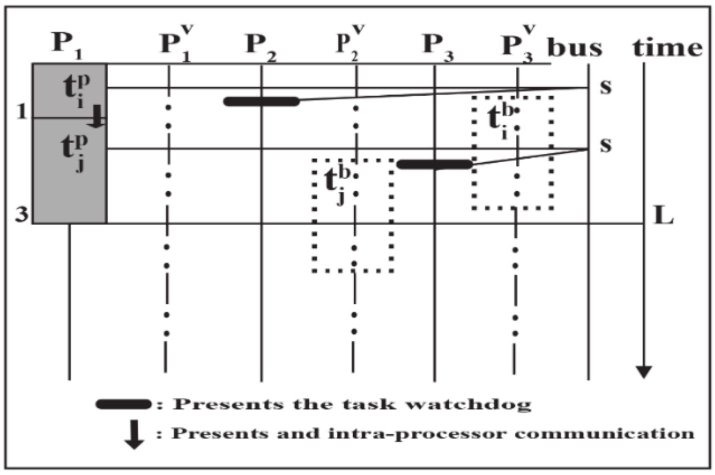
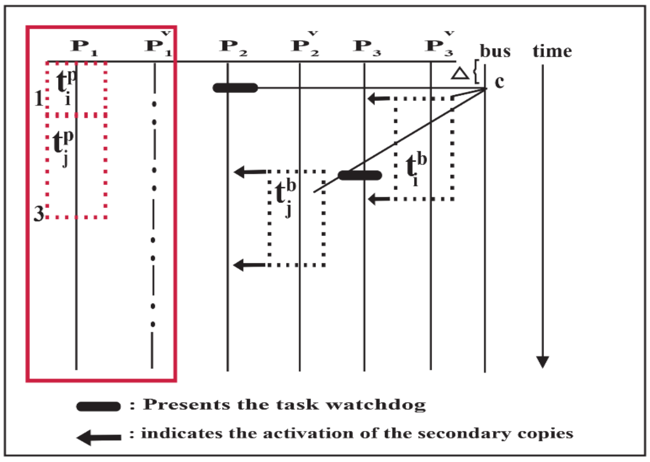


Fig. 9. Distributed schedule with absence of failure

In case of a failure, p_1 for example, w_i does not receive data of t_i^{p}, then after the time out Δ, w_i wake the backup t_i^{b} located on p_3 by running the conditional dependence c, the wake-up message is received too by the task t_j^{b}. So, the failure of p_1 is masked as it is shown in Fig. 10.


P1
P1v
P2
P2v
P3
P3v
bus
time
c
1
3
t_i^p
t_j^p
t_i^b
t_j^b
: Presents the task watchdog
: indicates the activation of the secondary copies
Fig. 10. Distributed schedule with presence of failure

Then t_i^{b} will be activated and executed on p_3, it sends the result x_{b} to t_j^{b} by inter-processor communication to be performed and provide the results. The processor $p_1$will be excluded from the material architecture. Finally, the execution of the system in subsequent cycles follows this new distribution/scheduling (Fig. 11).

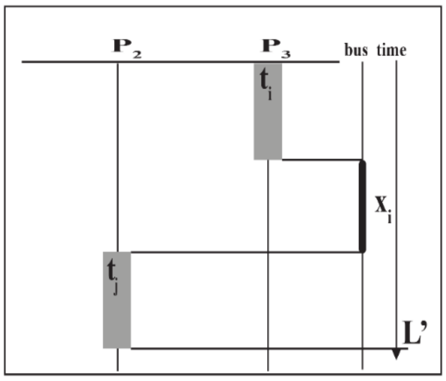


Fig. 11. New distributed schedule after failure

5. Evaluation of the methodology AAA-FAULT$^{\mathrm{DT}}$

We have based on the methodology AAA (Adequation Algorithm Architecture) to schedule tasks on processors, it is based on a cost function called scheduling pressure, denoted σ_{T_i,P_j}^{n} [1]. $\sigma_{T_i,P_j}^{(n)} = \mathrm{St}_{T_i,P_j}^{(n)} + \overline{\mathrm{st}}_{T_i}^{(n)} - R^{(n-1)}$ where:

$\mathrm{St}_{T_i,P_j}^{(n)}$ represents the earliest start date of T_i on P_j, from the beginning [15];

$\overline{\mathrm{st}}_{T_i}^{(n)}$ represents the latest start date of T_i, since the end [15];

$R^{(n)}$ is distribution/scheduling length of an algorithm graph on an architecture graph.

5.1. Evaluation parameters

We have applied AAA-FAULT^{DT} heuristic to a set of random algorithm graphs with a set of parameters that affect our results. Processors number *P*, tasks number *N* and the factor CCR (ratio of the average communication time and the average execution time) are the parameters that we have modified to test the effectiveness of our methodologies in the absence/presence of failures.

Following Kalla's Cheme [1], a random algorithm graph is generated (Fig. 12) by several levels (≥ 2). Each level *i* is composed of several nodes (tasks), and each node of level *i* has at least one predecessor lower level *j* such that $i > j$. Fig. 12 shows the process of generating random graphs ALG. This generator is based on three parameters:

- *N* – the graph size *n*; the number of nodes in the graph (each node corresponds to the task).
- *H* – the height of the graph; the maximum number of levels in the graph;
- *L* – the width of the graph; the maximum number of independent nodes in a level graph.

The node generation phase itself is performed in two steps. First, we draw randomly *n* nodes in a matrix of dimensions ($L \times H$), *L* and *H* affect the shape of the graph. Next, we construct the *N* levels of the graph; each level consists of *k* nodes on the same horizontal line in the matrix. In the generation phase of arcs, we generate the arcs in two steps. First, we choose randomly for each node of level *i* one or more nodes as predecessors of level *j*, with $j = i - 1$. Next, we choose randomly for each node of level *i* zero or more nodes as predecessors of level *j*, with $j < i - 1$.

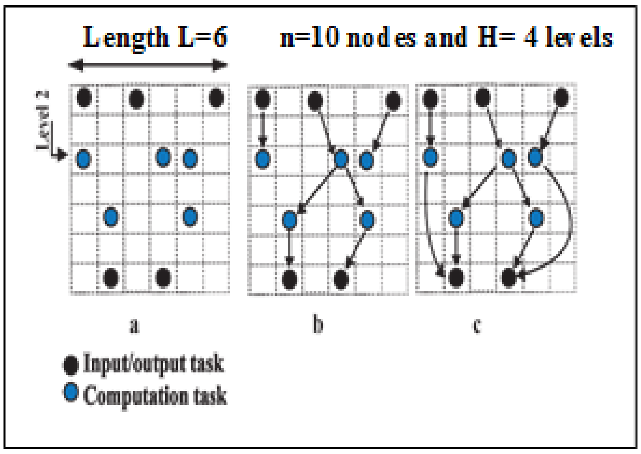


Fig. 12. Stages of random graph generation

5.2. Results

We have plotted in Figs 13, 14 and 15 the schedule length *L* as a function of *N*, *P* and CCR. The orthogonal axis in all figures shows the schedule length obtained by the execution of our methodologies on a desktop computer with moderate capacities, also the values of the execution costs of the tasks on the processors and the communication durations are taken randomly from an interval of integer values from 1 up to 10 then they do not reflect the actual values of the microprocessor execution time nor the communication time between them. In reality, they are incomparable; of course, very much better (it's a simulation).

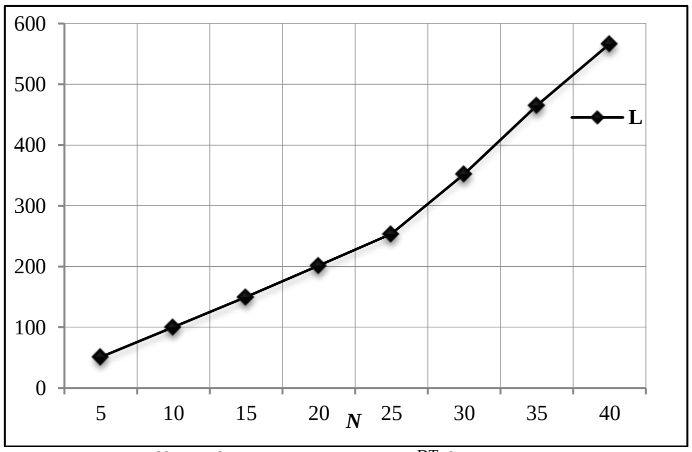


Fig. 13. Effect of *N* on AAA-FAULT^{DT} for *p*=5 and CCR=2

Fig. 13 shows that the schedule length increases with *N* that they are more replicated.

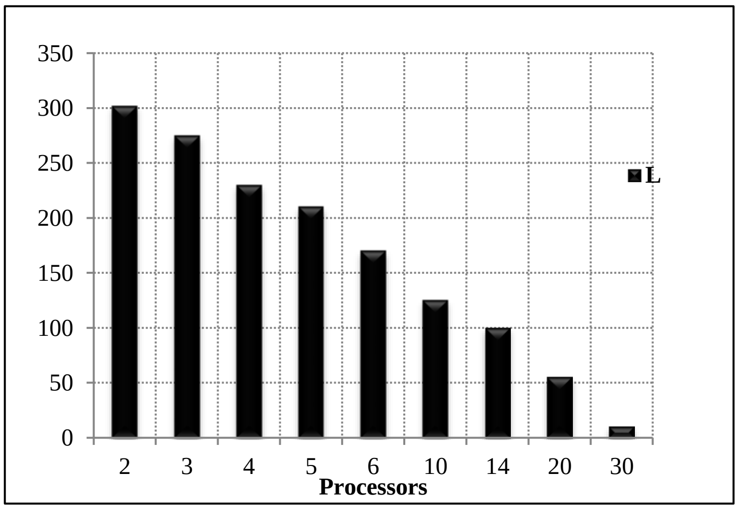


Fig. 14. Effect of number of processors on AAA-FAULT^{DT} for *N*= 40 and CCR=1

Fig. 14 shows that the schedule length decreases with *P*. This is due to the number of processor available to tolerate failures.

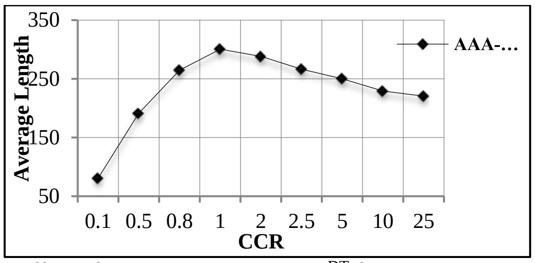


Fig. 15. Effect of CCR on AAA-FAULT^{DT} for P = 6 and N =50

Fig. 15 shows that the schedule length increases for CCR<1 and decreases for CCR>1. This is due to the performance of heuristics when the communication cost is important.

Regardless of tasks number, execution duration and communication duration which increase the schedule length, results in obtained proof of the effectiveness of our methodology in the case of processor failure. In fact even the scheduling distribution length increases, it does not exceed the deadline. So, the expected service of the system is provided on time in all cases.

6. Evaluation of the methodology AAA-FAULT^{IDT}

Due the principle of this methodology which combines the advantages of passive and active redundancy, results obtained compared to the previous ones are better. We have applied it to a set of random algorithm graphs with a set of parameters. We denote by:

L is the length of distribution and scheduling real-time of dependent tasks (AAA-FAULT^{DT}).

L' is the length of distribution and scheduling real-time of independent tasks (AAA-FAULT^{IDT}).

* denotes DT or IDT

By varying the number of tasks on a random algorithms architecture and hardware architecture, we obtain the following results (Fig. 16).

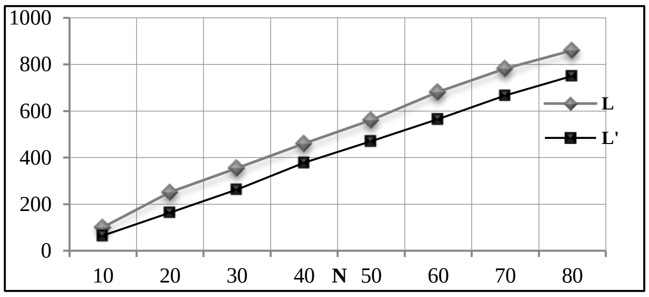


Fig. 16. Effect of N on AAA-FAULT* for p=5 and CCR=2

By varying the number of processors on random algorithm architecture of 50 tasks, we obtained the following results in Fig. 17.

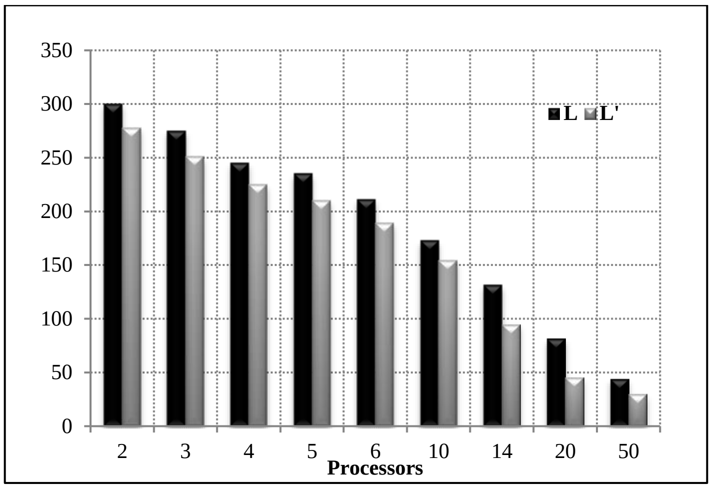


Fig. 17. Effect of number of processors on AAA-FAULT* for *N*= 40 and CCR=1

Results of this methodology show that it is better than the previous. In fact, as we have already said, it has small recovery delay after failures and small execution delay without failures.

7. Conclusion

Dependability of critical real-time system is a crucial property which must be taken into account in conception process. Our work is based on the processors fault tolerance techniques in a multi-component heterogeneous non-preemptive system connected by a bus. Our solutions can minimize the schedule length on the absence/presence of faults in the case of dependent or independent tasks. Simulations results show the performance of our proposed approaches. Currently, we are performing extensive benchmark testing of our heuristics on heterogeneous architectures with several failures. The first results show that the overheads increase with the number of failures.

References

1. K a l l a, H. Génération Automatique de Distributions/Ordonnancements Temps Réel Fiables et Tolérants aux Fautes. PhD Thesis, Mathematics, Sciences and Technologies of the Information, L'INPG, 2004. 179 p.
2. Z h a n g, Y., K. C h a k r a b a r t y. Energy-Aware Adaptive Checkpointing in Embedded Real-Time Systems. – IEEE Computer Society Washington, DC, USA, Vol. **1**, 2003, p. 10918.
3. L i u, X., H. D i n g, K. L e e, L. S h a, M. C a c c a m o. Feedback Fault Tolerance of Real-Time Embedded Systems – Issues and Possible Solutions, ACM New York, NY, USA, Vol. **3**, April 2006, No 2, pp. 23-28.
4. Dependability: Basic Concepts and Terminology. – In: J. C. Laprie, Ed. Dependability: Basic Concepts and Terminology. Dependable Computing and Fault-Tolerant Systems, Springer, Vienna, Vol. **5**, 1992, pp. 3-245.
5. J a l o t e, P. Fault-Tolerance in Distributed Systems. Prentice-Hall, Inc., 1994.

6. K r a k o w i a k, S. Tolérance aux fautes – 1 Introduction, techniques de base. Joseph Fourier University Sardes Project (INRIA and IMAG-LSR), 2004.
7. L o p e z, P. Approche par contraintes des problèmes d'ordonnancement et d'affectation: structures temporelles et mécanismes de propagation. Paris, France, Ellipses Marketing Edition S. A., 2004.
8. M a r o u f, M. Ordonnancement temps réel dur multiprocesseur tolérant aux fautes appliqué à la robotique mobile. PhD Thesis, Real Time Computing, Robotics and Automation, Mines Paris Tech, 2012. 212 p.
9. de R a u g l a u d r e, D. Vérification formelle de conditions d'ordonnançabilité de tâches temps réel périodiques strictes. JFLA – Francophone Days of Business Languages, 2012, February 2012, Carnac, France, 2012.
10. G i r a u l t, A., H. K a l l a. A Novel Bicriteria Scheduling Heuristics Providing a Guaranteed Global System Failure Rate. – IEEE Trans. Dependable Sec. Comput., Vol. **6**, 2009, No 4, pp. 241-254.
11. A r a r, C., M. S. K h i r e d d i n e. An Algorithm Based on Replication and Deallocation Efficient Fault-Tolerant Multi-Bus Data Scheduling Algorithm Based on Replication and Deallocation. – Cybernetics and Information Technologies, Vol. **16**, 2016, No 2, pp. 69-84.
12. P r i y a n k a, M., S. A n i s h a, R. S a k t h i P r a b h a. VLSI Design for a PSO-Optimized Real-Time Fault-Tolerant Task Allocation Algorithm in Wireless Sensor Network. – ARPN Journal of Engineering and Applied Sciences, Vol. **11**, July 2016, No 13, pp. 8226-8230.
13. G a r g, R., S. A. K u m a r. Fault Tolerant Task Scheduling on Computational Grid Using Checkpointing under Transient Faults. – Arabian Journal for Science and Engineering, Vol. **39**, 2014, No 12, pp. 8775-8791.
14. N a g a r a j a n, K., J. P. H a y e s, B. T. M u r r a y. Task Scheduling Algorithms for Fault Tolerance in Real-Time Embedded Systems. – In: D. R. Avresky, Ed. Dependable Network Computing. The Springer International Series in Engineering and Computer Science. Vol. **538**. Boston, MA, Springer, 2000, pp. 395-412.
15. W i l w e r t, C. Influence des Fautes Transitoires et des Performances Temps Réel sur la Sûreté des Systèmes X-by-Wire. PhD Thesis, Networks and Telecommunications [cs. NI]. National Polytechnic Institute of Lorraine – INPL, France 2005. 131 p.
16. D o n a d e o, R. Tivoli Workload Scheduler for z/OS Conditional and Step Dependencies: When and How to Use Them to Shorten Workload Execution Time or to Automatically Handle Branching Scenarios. – IBM Rome Tivoli Lab Via Sciangai, 53 00144, Rome, Italy 2011, p. 21.
17. F a n, W., F. G e e r t s, X. J i a. Conditional Dependencies: A Principled Approach to Improving Data Quality. – In: A. P. Sexton, Ed. Dataspace: The Final Frontier. BNCOD 2009. Lecture Notes in Computer Science. Vol. **5588**. Berlin, Heidelberg, Springer, 2009, pp. 8-20.
18. M o n t r e s o r, A. Distributed Algorithms Consistency & Replication. University of Trento, Attribution-ShareAlike 4.0 International License, Italy 2016.
19. H o l r o y d, C. B., N. Y e u n g, M. G. H. C o l e s, J. D. C o h e n. A Mechanism for Error Detection in Speeded Response Time Tasks. – Journal of Experimental Psychology: General, Vol. **134**, 2005, No 2, pp.163-191.
20. H a s h i m o t o, K., T. T s u c h i y a, T. K i k u n o. Effective Scheduling of Duplicated Tasks for Fault Tolerance in Multiprocessor Systems. – IEICE Transactions on Information and Systems, Vol. **E85-D**, 2002, No 3, pp. 525-534.
21. O h, Y., S. H. S o n. Scheduling Real-Time Tasks for Dependability. – Journal of Operational Research Society, Vol. **48**, 1997, pp. 629-639.
22. **http://www.syndex.org**

Received 31.01.2018; Second Version 21.06.2018; Accepted 28.06.2018