DE GRUYTER OPEN

BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **16**, No 2
Sofia • 2016 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.1515/cait-2016-0021

An Efficient Fault-Tolerant Multi-Bus Data Scheduling Algorithm Based on Replication and Deallocation

Chafik Arar[1], *Mohamed Salah Khireddine*[2]

[1]*Computer Science department, Batna University, Batna, 05000 Algeria*
[2]*Department of Electronics, Batna University, Batna, 05000 Algeria*
Emails: chafik.arar@gmail.com mkhireddine@yahoo.fr

Abstract: *The paper proposes a new reliable fault-tolerant scheduling algorithm for real-time embedded systems. The proposed scheduling algorithm takes into consideration only one bus fault in multi-bus heterogeneous architectures, caused by hardware faults and compensated by software redundancy solutions. The proposed algorithm is based on both active and passive backup copies, to minimize the scheduling length of data on buses. In the experiments, this paper evaluates the proposed methods in terms of data scheduling length for a set of DAG benchmarks. The experimental results show the effectiveness of our technique.*

Keywords: *Fault-tolerance, scheduling, real time systems, active and passive redundancy, replication, deallocation.*

1. Introduction

Nowadays, our society becomes increasingly dependent on heterogeneous, distributed, embedded and real-time systems, which take over even exceptionally critical decisions. These systems are increasingly becoming more complex and more sensitive to faults, due to potentially catastrophic consequences that could result from a malfunction of these systems, fault tolerant techniques are required to ensure that these systems continue to provide a correct service in spite of faults [1-4]. The hardware thus as the software of a system, can be the target of a variety of faults with different causes; we concentrate on hardware faults and especially communication faults.

We can define fault tolerance as a system's ability to continue operating as planned, despite the presence of faults. There are different ways to achieve fault tolerance. A common one is that some redundancy (such as re-execution and

N-version programming) or a kind of recovery actions is built into the system. However, fault tolerance increases complexity and may lead to performance degradation if applied in an artless way.

As we target embedded systems, due to their limited resources (due to space, weight and cost considerations); it is impossible to provide space redundancy. This is why we study only time redundancy solutions. Several fault-communication tolerance approaches for distributed embedded real-time systems have been proposed. These techniques are based on active or passive backup methods.

In the active backup scheme, different copies of the message are sent along distinct buses. In [5] authors develop a fault-tolerant allocation and scheduling method, which maps messages onto a low-cost multiple-bus system to ensure predictable inter-processor communication. In [6], the reliability of the system can be increased by providing several paths from source to destination and sending the same packet through each of them (the algorithm is known as multipath routing), authors use this idea to propose a new mechanism that enables the trade-off between the amount of traffic and the reliability.

On the other hand, in the passive backup scheme only the primary copy of the message is sent; if it fails, another copy (backup) of the message, will be transmitted. In [7], authors provide a generic algorithm, based on replication of operations and data communications, which solves the problem of off-line fault tolerant scheduling of an algorithm onto a multiprocessor architecture. They take into account two kinds of failures: fail-silent and omission. In [8], the authors propose a synthesis-based design methodology, which incorporates formal validation techniques, and relieves the designers from the burden of specifying detailed mechanisms for addressing platform faults, while involving them in the definition of the overall fault-tolerance strategy. In [9], authors survey the 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. In [10], authors propose a new method to identifying bus faults based on support vector machine. The proposed method operates on two stages, first, the bus fault state is simulated using PSCAD/EMTDC, then a support vector machine model is established for carrying out data pre-treatment. In [2], both, active redundancy and a TDMA (Time Division Multiple Access) communication protocol is used to tolerate faults of buses. In [11], authors propose a fine grained transparent recovery, where the property of transparency can be selectively applied to processes and messages. In [12] authors propose a QoS-aware dynamic fault-tolerant scheduling algorithm called QAFT that can tolerate a node's permanent failures at one time instant for real-time tasks.

In this paper, we are interested in approaches based on scheduling algorithms, more specifically those based on static scheduling that allows for the inclusion of the dependencies and execution cost of tasks and data dependencies in its scheduling decisions, and the schedule is already computed at compile-time.

The main objective is to minimise the scheduling length of data on buses, which is the total sending time of data, under the assumption that at most only one bus may fail. The basic idea of our work, which is the combination of active and

passive redundancy in the same scheme, was originally proposed by [4], for processors fault tolerance, what we propose is its adaptation for communication fault tolerance. For that, many transformation and redevelopment are needed.

First, we start by outlining the definition of the scheduling problem as an optimization problem. We use the linear programming to formulate the optimization problem of the fault-tolerant scheduling data with two types of backup copies, to minimise the scheduling length. It provides the best results, but since the problem is NP-hard, this solution generally takes a long time to obtain an optimal solution, and for some cases we cannot find a feasible solution in an acceptable time.

To overcome this problem, we propose our solution, based on a heuristic algorithm called: Fault-Tolerant multi-bus data scheduling Algorithm based Replication and Deallocation (FTA-RD). 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 faults. We are able to show with simulation results that our approach can generally reduce the run-time overhead.

The remainder of this paper is structured as follows: in Section 2, we give detailed description of our system models and backup copies types. In Section 3, we introduce and discuss our approach with a motivational example, which shows how our approach can minimize the length of the whole generated schedule. Section 4 present our solution and give a detailed description of our scheduling algorithm. In Section 5, we present the experiments. We finally conclude this work in Section 6.

2. Problem definition

2.1. System models description

In this section, we first give some definitions that describe our system and then we define the problem for fault-tolerant scheduling formally. The specification of this system involves the description of tasks and data models, architecture model and fault model.

2.1.1. Task model

The task model is defined by a Directed Acyclic Graph (DAG) noted $G_{\text{task}} = (T, E, \text{Exe}_{\text{task}})$, where: $T = \{t_1, t_2, \cdots, t_n\}$ represents a set of *n* tasks; *E* is a set of directed edges represents the task dependencies, where an edge from a task t_i to a task t_j noted by $t_i \rightarrow t_j$ means that task t_j depends on the output of task t_i; $\text{Exe}_{\text{task}}(t_i)$ is a function that calculates the execution cost of task $t_i \in T$. Fig. 2 represents an example of a task model.

2.1.2. Architecture model

The architecture is modelled by a non-directed graph, noted $G_{\text{arch}} = (P, B)$, where each node is a processor, and each edge is a media communication (bus). We

assume that the architecture is heterogeneous and fully connected. Fig. 1 shows an example of an architecture model.

2.1.3. Data model

The data model is modelled by another DAG noted $G_{\text{data}} = (M, P, \text{Exe}_{\text{data}})$. The graph G_{data} is generated from the graph G_{task} with a transformation that respects the data precedence, $M = \{m_1, m_2, \cdots, m_l\}$ represents a set of all data transferred between tasks, the cardinality of *M* is equal to that of *E*, *P* is a set of directed edges and represents the precedence relationships, where an edge from a data m_i to a data m_j noted by $m_i \rightarrow m_j$ means that data m_j require m_i to be calculated, $\text{Exe}_{\text{data}}(m_i)$ is a function, represents the transfer cost of data $m_i \in M$ and the time required to run a failure-detection routine that determines whether the data was received successfully or not.

2.1.4. Failure model

We assume only buses faults. Each bus may fail due to hardware fault. The faults can be transient or permanent and are independent. It is assumed that at most one bus will fail to execute data transfer, in our proposed algorithms. We call it one-bus failure model. There exists a fault-detection mechanism such as fail-signal and acceptance test to detect the bus failure.

2.2. Backups copies

2.2.1. Replicated backup copy

A Replicated backup copy m_i^{Rep} of data m_i is an active backup copy, which is sent independently, no matter whether the Primary copy m_i^{Pr} was received successfully or not. In the case that the primary copy fails to reach its destination properly, the replicated copy can be used instead of the primary. For example, in Fig. 5, m_1^{Rep} is a replicated backup copy of m_1 scheduled on bus B_2 .

2.2.2. Deallocated backup copy

A Deallocated backup copy m_i^{Del} of data m_i is a passive backup copy, which is sent only if the primary copy m_i^{Pr} fails. The deallocated copy cannot be scheduled to start until the primary copy was completely sent and the activation message was received. An activation message A_{message} is a message originating from a primary copy to its deallocated backup copies, which indicates whether m_i was sent successfully or not. We use $\| A_{\text{message}} \|$ to denote the time cost of the message.

The fact of scheduler several deallocated backups at the same time on the same bus, is called backup overlapping. The implementation of backup overlapping is under the assumption that at most, there is one bus failure to be tolerated so that no more than one backup will run at any time. For example, in Fig. 5, m_2^{Del} is a deallocated backup copy of m_2 scheduled in bus B_2, and overlapping with another deallocated backup copy m_6^{Del}.

3. Illustrative example

In this section, we provided an example to illustrate the problem that we try to solve. The architecture model of our system is composed of three processors fully connected with three buses (as it is shown in Fig. 1), we assume that there will be at most one bus fault.

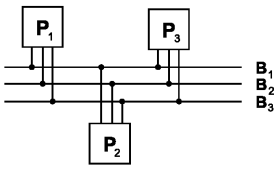


Fig. 1. Architecture model

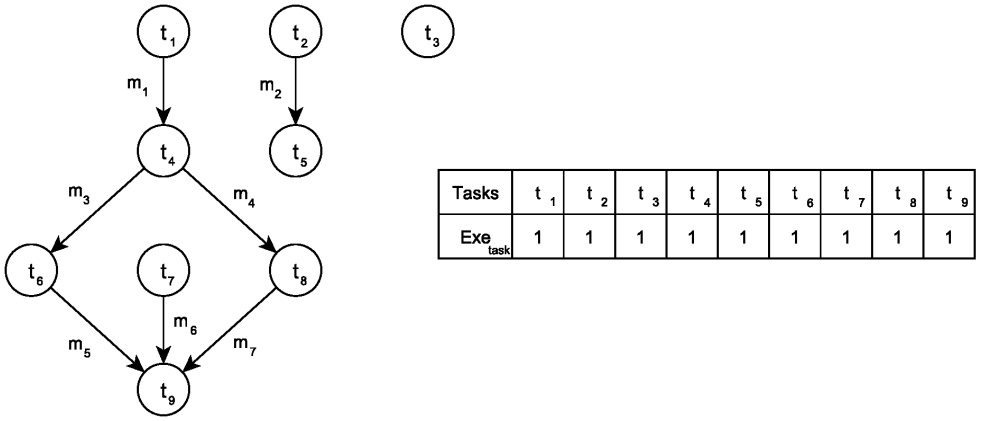
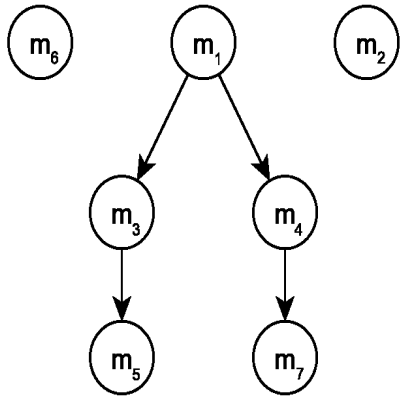


Fig. 2. Task model

Fig. 2 presents the tasks model of our example, with nine tasks t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8 and t_9. The edges show the task dependencies. In the graph, for instance, task t_4 can't start to be executed until task t_1 was executed successfully.

Fig. 3 presents the data model generated from the tasks model of Fig. 2. The data model regain and respects the tasks dependencies. In the graph, for instance, data m_3 and m_4 can't be sent until data m_1 was successively received, because in

tasks model of Fig. 2, task t_4 need m_1 to calculate m_3 and m_4. For this example, we assume that the time delay for the activating message $\|A_{\text{message}}\|$ is 1 time unit.



Data	m_1	m_2	m_3	m_4	m_5	m_6	m_7
Exe_{data}	2	2	2	2	2	2	1

Fig. 3. Data model

The problem to solve is finding an optimal schedule with a minimized length for these data and their backup copies, so that all data can successfully be transmitted, assuming that only one bus may fail.

Fig. 4 presents an example of a non fault-tolerant scheduling, the length of the scheduling for this case is equal to 8.

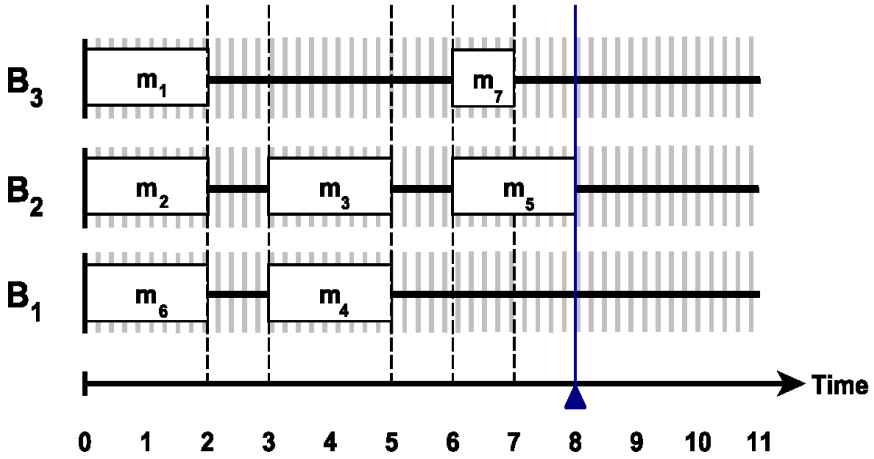


Fig. 4. No fault-tolerant data scheduling

In the case of our example, the advantage of combining both replication and deallocation in the same algorithm is shown by three kinds of optimal schedules to tolerate one bus fault (Figs 5, 6 and 7). All the optimal results can be obtained by the linear programming formulation presented in Section 4.

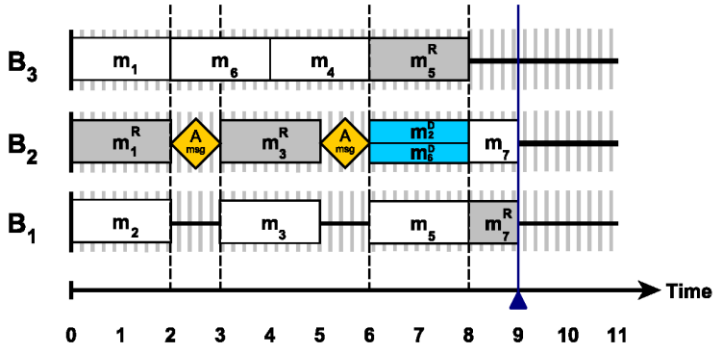


Fig. 5. Optimal fault-tolerant schedule with both replicated and deallocated backup copies

An optimal fault-tolerant schedule with both replicated and deallocated backup copies for our example is shown in Fig. 5. The minimal scheduling length is 9 time units. In this schedule we choose the deallocated backup copies for data m_2, m_6 and replicated backup copies for the rest of data. In this schedule, the deallocated backup copy of data m_2 is overlapping with that of data m_6 at Steps 7 and 8 on bus B_2 to reduce the scheduling length.

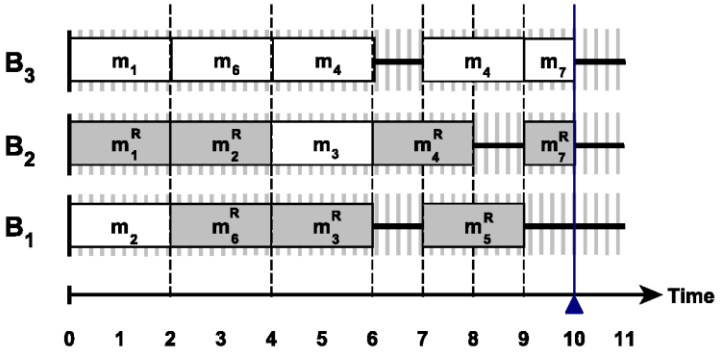


Fig. 6. Optimal fault-tolerant schedule with only replicated backups copies

An optimal fault-tolerant schedule with only replicated backup copies, with a length equal to 10 time units, is shown in Fig. 6.

An optimal fault-tolerant schedule with only deallocated backup copies is show in Fig. 7, the scheduling length is 17 time units. In this schedule, m_2^{Del} and m_1^{Del} are overlapping at Steps 4 and 5 on bus B_2, m_3^{Del} and m_4^{Del} are also overlapping at Steps 10 and 11, on the bus B_1. There is at least 1 time unit delay for $\|A_{\text{message}}\|$ between data m_i and its deallocated backup copies m_i^{Del}. If the primary copy is sent successfully, the activation message $\|A_{\text{message}}\|$ cancels its deallocated backup copy.

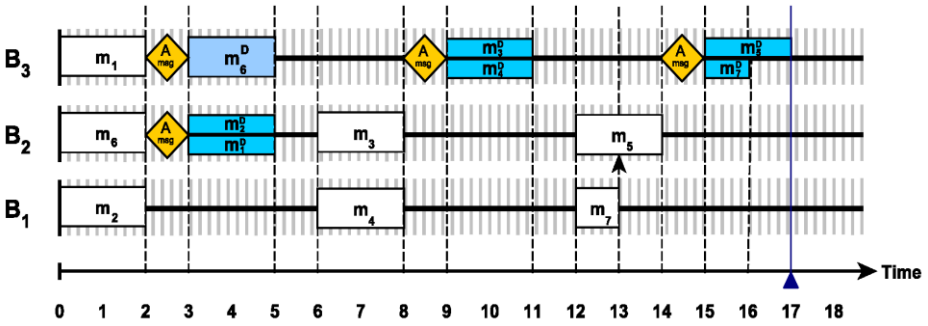


Fig. 7. Optimal fault-tolerant schedule with only deallocatted backups copies

In this example, the optimal schedule with both deallocation and replication has 47.05% reduction over deallocation only in scheduling length, when he loses only 12.5% compared to a non-fault tolerant solution. And it also reduces the scheduling length by 10% comparing with the optimal schedule with replication only. The performance is improved significantly.

4. The proposed approach

In this section, we define the scheduling problem as an optimization problem and we use the linear programming to find the optimal scheduling of data dependency with their backup copies to tolerate one-bus fault, with minimal length.

As our solution is based on a hybrid approach that combines both passive and active redundancy we use two types of backup copies, replicated and deallocated. We model data dependencies scheduling with a binary variables to determine the order of data. $M^{\mathrm{Pr}}_{i,t,j}$ is a binary variable such that $M^{\mathrm{Pr}}_{i,t,j} = 1$ if and only if the primary copy of data m_i was successfully sent on bus B_j at step *t*. Similarly, the binary variables $M^{\mathrm{Rep}}_{i,t,j}$ and $M^{\mathrm{Del}}_{i,t,j}$ are used for the replicated and deallocated backup copies.

$$\forall i \in \{1,\cdots,n\}, \forall j \in \{1,\cdots,n_{\mathrm{bus}}\} \ \ \forall t \in \{1,\cdots,L_{\mathrm{sch}}\}, \quad M^{\mathrm{Pr}}_{i,t,j}, M^{\mathrm{Rep}}_{i,t,j}, M^{\mathrm{Del}}_{i,t,j} \in \{0,1\}, \tag{1}$$

L_{sch} is an upper bound of the scheduling length.

The objective function of our linear problem is the minimization of the scheduling length, which can be mathematically formulated as

$$\text{Minimize} \sum_{t=1}^{L_{\mathrm{sch}}} \sum_{j=1}^{n_{\mathrm{bus}}} t M^{\mathrm{Pr}}_{\mathrm{out},t,j}, \tag{2}$$

where m_{out} is a fictitious node added to data model (DAG), to compute total scheduling length (Fig. 8).

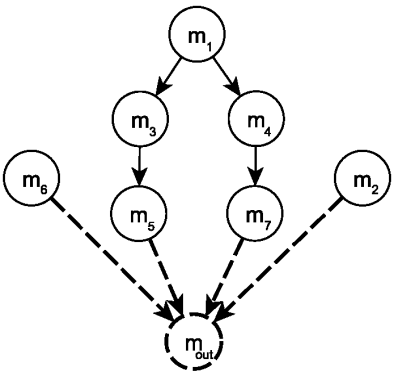


Fig. 8. The fictitious node add to the DAG

The minimization of the total length of the scheduling is given under the following constraints:

Data mapping constraint. The primary copy of each data is scheduled once and only once,

$$\forall i \in \{1,\cdots, n\} \quad \sum_{t=1}^{L_{\text{sch}}} \sum_{j=1}^{n_{\text{bus}}} M_{i,t,j}^{\text{Pr}} = 1. \tag{3}$$

The backup copy of each data is scheduled once and only once,

$$\forall i \in \{1,\cdots, n\} \quad \sum_{t=1}^{L_{\text{sch}}} \sum_{j=1}^{n_{\text{bus}}} M_{i,t,j}^{\text{Rep}} + M_{i,t,j}^{\text{Del}} = 1. \tag{4}$$

At any time, a bus B_x is used either for the transmission of a primary copy of data or its replicated backup copy,

$$\forall t \in \{1,\cdots, L_{\text{sch}}\},\ \forall k \in \{1,\cdots, n_{\text{bus}}\} \quad \sum_{i=1}^{n} M_{i,t,j}^{\text{Pr}} + M_{i,t,j}^{\text{Rep}} \le 1. \tag{5}$$

In a multi-bus system, to tolerate one-bus fault, we can use only one backup copy for each data dependency. This backup copy can be either replicated or deallocated.

Dependency constraint. Backup copies must meet the same precedence relationships as their primary copies.

$m_i \to m_k \in P$ means that data m_k require data m_i to be calculated; data m_k cannot be send until data m_i was received and used to calculate m_k :

$$\forall i,k \in \{1,\cdots,\ n\} \quad \sum_{t=1}^{L_{\text{sch}}} \sum_{j=1}^{n_{\text{bus}}} t^* M_{i,t,j}^{\text{Rep}} + \text{Exe}(m_i) \le \sum_{t=1}^{L_{\text{sch}}} \sum_{j=1}^{n_{\text{bus}}} t^* M_{k,t,j}^{\text{Rep}}. \tag{6}$$

For each $m_i \to m_k \in P$, a new directed acyclic graph that represents all dependencies between possible backup copies and their primary copies is generated. Fig. 9 presents the part of DAG for $m_i \to m_k$.

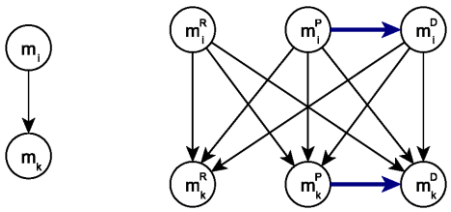


Fig. 9. The new DAG for $m_i \rightarrow m_k$

Fig. 10 presents the complete DAG for the data model in Fig. 3.

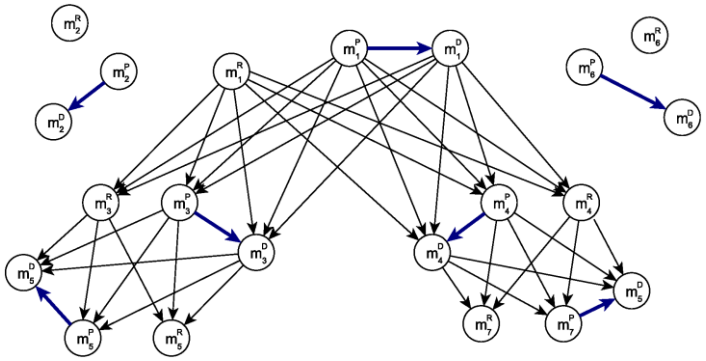


Fig. 10. The complete DAG for data model of our example

But for our example we have chose deallocated backup copies for m_2 and m_6, and replicated backup copies for m_1, m_3, m_4, m_5 and m_7, so Fig. 11 presents the reduced DAG respecting this choice.

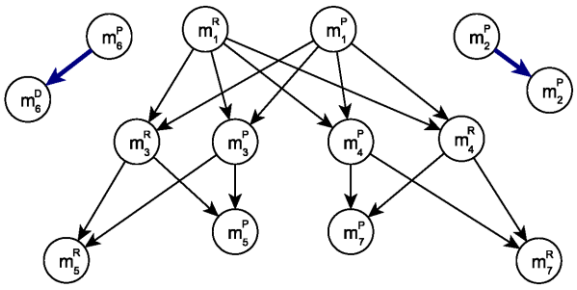


Fig. 11. The reduced DAG

Fault tolerant constraint: the primary copy and its backup copy should not, in no case, be assigned to the same bus:

$$\forall i \in \{1, \cdots, n\}, \forall j \in \{1, \cdots, n_{\text{bus}}\}$$

$$\sum_{t=1}^{L_{\text{sch}}} M_{i,t,j}^{\text{Pr}} + M_{i,t,j}^{\text{Pep}} + M_{i,t,j}^{\text{Del}} \leq 1. \quad (7)$$

Execution constraint: For the tasks dependencies $t_x \rightarrow t_y \rightarrow t_z$ (as it is shown in Fig. 12) the data m_b can't be scheduled until data m_a was successfully received by task t_y and this latter was completely executed,

(8) $$\forall j \in \{1, \cdots, n_{\text{bus}}\} \quad t * M^{\text{Pr}}_{i,t,j} + \text{Exe}_{\text{task}}(t_y) \le t' M^{\text{Pr}}_{i,t',j}.$$

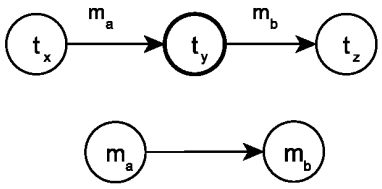


Fig. 12. Execution constraint

The same can be said for all other combinations, in our case we can count nine possibilities: (Pr, Pr), (Pr, Rep), (Pr, Del), (Rep, Pr), (Rep, Rep), (Rep, Del), (Del, Pr), (Del, Rep), (Del, Del).

Overlapping and resource constraint. The overlapping of two deallocated backup copies or more must respect the following conditions: their primary copies must be assigned to different buses and must be completely independent, i.e., they have no dependency.

Only, the deallocated backup copies, may overlap. If there are two deallocated backup copies $M^{\text{Del}}_{i,t,j}$ and $M^{\text{Del}}_{k,t,j}$ overlapping on the same bus B_j, then their primary copies are assigned to two others different buses than B_j:

(9) $$\forall i, k \in \{1, \cdots, n\} \quad M^{\text{Del}}_{i,t,j} = M^{\text{Del}}_{k,t,j} = 1 \Rightarrow \sum_{t=1}^{L_{\text{sch}}} M^{\text{Pr}}_{i,t,j} + M^{\text{Pr}}_{k,t,j} = 0,$$

(10) $$\forall i, k \in \{1, \cdots, n\}, \ \forall j \in \{1, \cdots, n_{\text{bus}}\} \quad \sum_{t=1}^{L_{\text{sch}}} M^{\text{Pr}}_{i,t,j} + M^{\text{Pr}}_{k,t,j} \le 0.$$

The solution obtained is thus the best result, but since the problem is NP-hard, this formulation generally takes a long time to obtain an optimal solution, and for some cases we cannot find a feasible solution in an acceptable time. That is why we propose our second solution, based on a heuristic algorithm called FTA-RD. 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 faults.

Our scheduling algorithm is a greedy list scheduling heuristic, which schedules one operation at each step. The input to our algorithm is an instance of the tasks graph $G_{\text{task}} = (T, E, \text{Exe}_{\text{task}})$ and the architecture graph $G_{\text{arch}} = (P, B)$, and the time cost of the activate message $\|A_{\text{message}}\|$; it generates a distributed static schedule of a given task model onto a given architecture model, which minimizes the system's run-time, and tolerates one bus fault. It is obvious that the FTA-RD algorithm's time complexity is polynomial time.

The FTA-RD scheduling algorithm is shown in Fig. 13.

Algorithm FTA-RD
Input: $G_{Task} = (T, E, Exe_{task})$, $G_{Arch} = P, B, N_{bus}, |A_{message}|$;
Output: A reliable fault-tolerant schedule, $BACKuP_{type}(m_i)$: Backup copy type of each data m_i ;

Initialization step

- Initialize the list of scheduled data : $M_{sched} := \emptyset$;
- $BACKuP \leftarrow DEAL$;
- Generate Data model graph $G_{Data} = (M, P, Exe_{data})$ from the tasks model graph G_{Task};
- Create a Queue M_{cand} of data candidates to schudling;
- Calculate the new data transfer time.

Scheduling step

While $(M_{cand} \neq \emptyset)$ **do**

(1) $m_i \leftarrow head[M_{cand}]$;

(2) Schedule m_i at the bus B_j at earliest available time. B_j is the bus with the least utilization.

(3) $M_{sched} := M_{sched} \cup \{m_i\}$

(4) **If** m'_i can be overlapping with an other deallocating copy **Then**

- Schedule an overlapping backup copy m'_i for m_i.
- $M_{sched} \leftarrow M_{sched} \cup \{m'_i\}$.
- $BACKuP_{type}(m_i) \leftarrow DEAL$.
- $BACKuP \leftarrow DEAL$;

Else;

Switch $BACKuP_{type}(m_i)$:

{

Case 'DEAL' :

- Schedule an new deallocating backup copy m'_i for m_i at bus B_k at earliest available time, B_k is the bus with the least utilization and it is different of B_j $(B_k \neq B_j)$.
- $M_{sched} \leftarrow M_{sched} \cup \{m'_i\}$.
- $BACKuP_{type}(m_i) \leftarrow REPL$.
- $BACKuP \leftarrow REPL$;
- break;

Case 'REPL' :

- Schedule an new replica backup copy m'_i for m_i at bus B_k at earliest available time, B_k is the bus with the least utilization and it is different of B_j $(B_k \neq B_j)$.
- $M_{sched} \leftarrow M_{sched} \cup \{m'_i\}$.
- $BACKuP_{type}(m_i) \leftarrow DEAL$.
- $BACKuP \leftarrow DEAL$;
- break;

}

End Switch

End IF;

End while

Scheduling length reduction step

- Re-choose the backup copy type for the last data task m_i.
- $M_{sched} \leftarrow M_{sched} \cup \{m'_i\}$.

End

Fig. 13. The FTA-RD scheduling algorithm

FTA-RD operates as follows:

1. First, it examines the data dependencies one by one to determine when and which bus each will be sent to. It finds the bus with the earliest available time and the least used one (the use of a bus is measured by the total time of transmission of all data assigned to that bus); therefore, this assignment, so made, ensures load balancing.

2. Once the primary copy is scheduled, the algorithm attempts to schedule the backup copy, first of all it tries to overlap with an existing deallocated backup copy, if not, it allocates a new replicated or a new deallocated backup copy according to the $BACKuP_{type}$.

3. Finally, to reduce the scheduling length, the backup type for the last data is redefined.

5. Simulations, results and discussion

In this section, we present the result of simulations, we compare the proposed scheduling algorithm FTA-RD with solution based on linear programming formulation, and solutions based only on replication or deallocation, in scheduling length. We have applied the FTA-RD heuristic to an example of an architecture graph composed of five processors and three buses. The failure rates of the processors are respectively 10^{-5}, 10^{-5}, 10^{-4}, 10^{-5} and 10^{-6}, and the failure rate of the Buses SAM_{MP1}, SAM_{MP2}, and SAM_{MP3} are respectively 10^{-6}, 10^{-5} and 10^{-4}.

The algorithms graphs used are those of DSP benchmark from the DSPstone [13], the number of tasks and data dependencies in each benchmark is listed in Table 1.

Table 1. Benchmarks Information

Benchmark I	Tasks	Data-dependencies
FIR filter	12	8
2-motiv	12	9
2 iir filter	17	13
2-deq filter	24	20
IIR biquad section	36	31
2-rls-lat	28	33
8latiir	46	38

We use Simulation Tool for Real time Multiprocessor scheduling (STORM) to calculate the data length. From the specification of the characteristics of software architecture (the tasks to schedule), hardware architecture (the resources for implementing these tasks) and the choice of a scheduling policy, the tool simulates the execution of these tasks on these resources according to the rules of this policy [14]. Fig. 14 shows the task model and the allocation of buses in STORM editor.

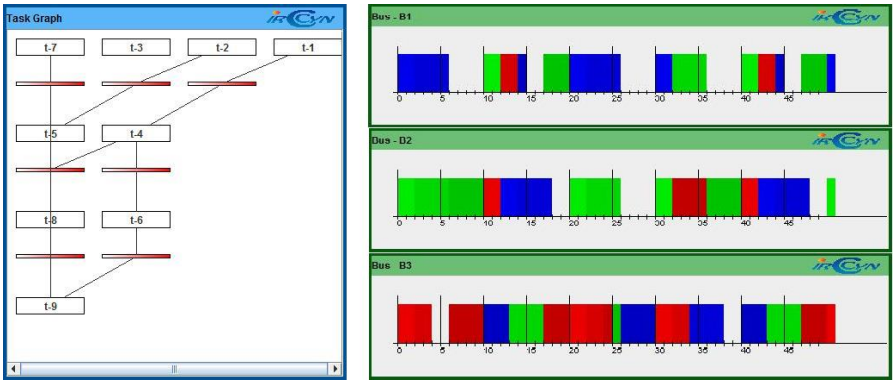


Fig. 14. Task model and bus's allocation

Results show that the FTA-RD heuristic with both the replicated and deallocated backup copies performs better than the one with replicated and deallocated heuristics only, in all benchmarks. We can see that FTA-RD heuristic can reduce the scheduling length by 6.19% on an average when compared to only-replication heuristic (Fig. 15), and 19.29% on average when compared to only-dealocation heuristic (Fig. 16). We can see also that our heuristic loses 8.41% on average in the scheduling length compared to the optimal solution obtained by linear programming formulation (Fig. 17).

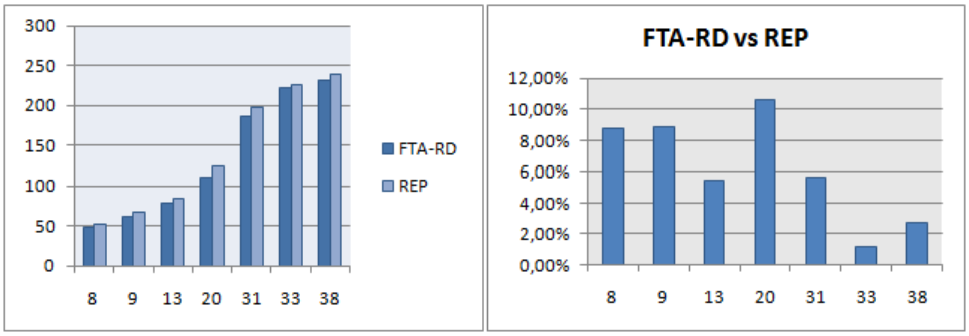


Fig. 15. Scheduling length of FTA-RD heuristic and only-replication heuristic

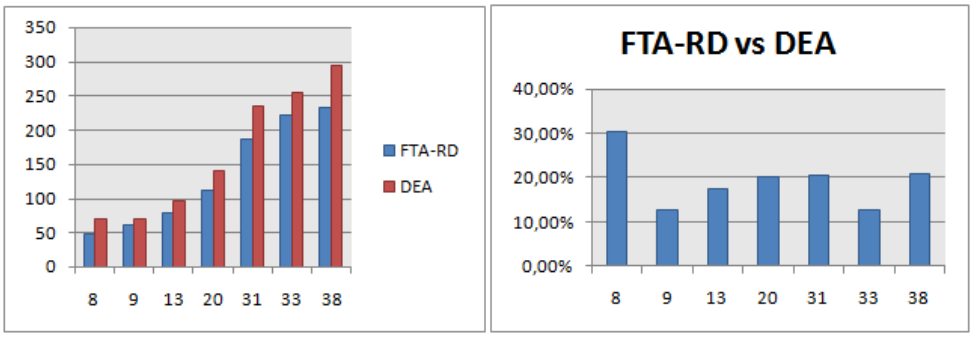


Fig. 16. Scheduling length of FTA-RD heuristic and only-deallocation heuristic

We can see from the results that:

- When the number of data dependencies is a small one, the algorithm that uses only the replicated backup copies is more efficient than one that uses deallocated backup copies. This is explained by the fact that when the number of data is low, the opportunity for backup overlapping is also low.
- When the number of data dependencies is more important, more deallocated backup copies can be overlapped, which allows reduction of scheduling length.

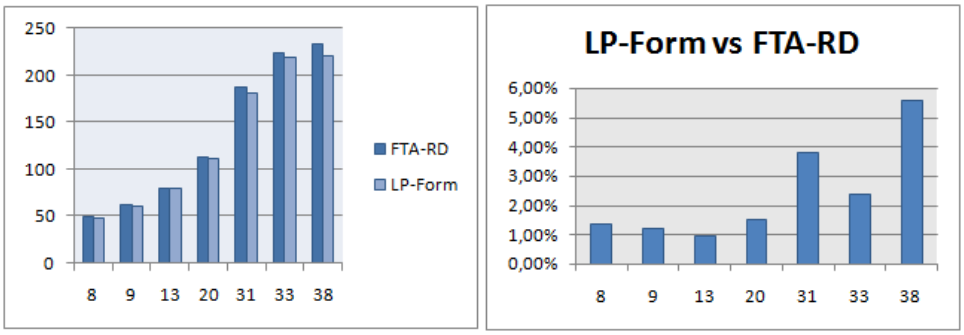


Fig. 17. Scheduling length of FTA-RD heuristic and LP-formulation solution

Also, delay due to the activation message, may have a negative effect on the schedule length.

The use of both replicated and deallocated backup copies allows the best results, that in the case of the use of only one types of backup copies.

The results obtained by the linear programming formulation are better than those obtained by the FTA-RD heuristic, the problem is that in many cases, the calculation time of this formulation is too long to produce results. The advantage of the FTA-RD heuristic is that it has a polynomial running time.

6. Conclusion

In this paper, we have studied the problem of fault-tolerance in embedded real-time systems and proposed a software implemented fault-tolerance solution for multi-buses architectures based on software redundancy. We have proposed a new scheduling heuristic, called FTA-RD, which produces automatically a static distributed fault-tolerant schedule when data dependencies are represented by directed acyclic graphs with two types of backup copies under the assumption that there will be at most one bus fault. The simulations show a significant improvement compared to algorithms with only one type of backup copy. Finally, we plan to carry out an experiment involving our method on an electric autonomous vehicle, with a 5-processor multi-buses architecture.

References

1. Jalote, P. Fault Tolerance in Distributed Systems. Prentice-Hall, Inc., 1994.
2. Koptez, H. Real-Time Systems: Design Principles for Distributed Embedded Applications. Springer Science & Business Media, 2011.
3. Grünsteidl, G., H. Kantz, H. Kopetez. Communication Reliability in Distributed Real-Time Systems. – In: Distributed Computer Control Systems 1991: Towards Distributed Real-Time Systems with Predictable Timing Properties, 2014, p. 123.
4. Jun, Z., E. Sha et al. Efficient Fault-Tolerant Scheduling on Multiprocessor Systems via Replication and Deallocation. – International Journal of Embedded Systems, Vol. **6**, 2014, No 2-3, pp. 216-224.
5. Kandasamy, N., J. P. Haye, B. T. Murray. Dependable Communication Synthesis for Distributed Embedded Systems. – Lecture Notes in Computer Science, 2003, pp. 275-288.
6. Dulman, S., T. Nieberg, J. Wu et al. Trade-Off between Traffic Overhead and Reliability in Multipath Routing for Wireless Sensor Networks. – IEEE, 2003.
7. Dima, C., A. Girault, C. Lavarenne et al. Off-Line Real-Time Fault-Tolerant Scheduling. – In: Proc. of 9th Euromicro Workshop on Parallel and Distributed Processing, 2001. IEEE, 2001, pp. 410-417.
8. Pinello, C., P. C. Luca, L. S.-V. Alberto. Fault-Tolerant Deployment of Embedded Software for Cost-Sensitive Real-Time Feedback-Control Applications. – In: Proc. of Conference on Design, Automation and Test in Europe-Volume 2. IEEE Computer Society, 2004, p. 21164.
9. Krishna, C. M. Fault-Tolerant Scheduling in Homogeneous Real-Time Systems. – ACM Computing Surveys (CSUR), 2014, Vol. **46**, No 4, p. 48.
10. Song, H., H. Wu. The Applied Research of Support Vector Machine in Bus Fault Identification. – In: Proc. of 6th International Conference on Natural Computation (ICNC), 2010, IEEE, 2010. pp. 1326-1329.
11. Izosimov, V., P. Pop, P. Eles et al. Scheduling and Optimization of Fault-Tolerant Embedded Systems with Transparency/Performance Trade-Offs. – ACM Transactions on Embedded Computing Systems (TECS), Vol. **11**, 2012, No 3, p. 61.
12. Zhu, X., X. Qin, M. Qiu. QoS-Aware Fault-Tolerant Scheduling for Real-Time Tasks on Heterogeneous Clusters. – Computers, IEEE Transactions on, Vol. **60**, 2011, No 6, pp. 800-812.
13. Zivojnovic, V., J. M. Velarde, C. Schlager et al. DSPstone: A DSP-Oriented Benchmarking Methodology. – In: Proc. of International Conference on Signal Processing Applications and Technology, 1994, pp. 715-720.
14. RTS Group, STROM. IRCCyN Laboratory in Nantes, France. **http://www.rts-software.org/**