AN ACTIVE REPLICATION SCHEME THAT TOLERATES FAILURES IN DISTRIBUTED EMBEDDED REAL-TIME SYSTEMS

Processors and communication links failures

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

[1]*INRIA Rhône-Alpes, 655 avenue de l'Europe, 38334 Saint-Ismier cedex, FRANCE*
{alain.girault,hamoudi.kalla}@inrialpes.fr

[2]*INRIA Rocquencourt, B.P.105 - 78153 Le Chesnay Cedex, FRANCE*
yves.sorel@inria.fr

Abstract Embedded real-time systems are being increasingly used in a major part of critical applications. In these systems, critical real-time constraints must be satisfied even in the presence of failures. In this paper, we present a new method-based on graph transformation that introduces fault-tolerance in building embedded real-time systems. The proposed method targets distributed architecture and can tolerate a fixed number of arbitrary processors and communication links failures. Because of the resource limitation in embedded systems, our method uses a software-based replication technique to provide fault-tolerance. Finally, since we use graph transformation to perform replication, our method may be used by any off-line distribution-scheduling algorithm to generate a fault-tolerant distributed schedule.

Keywords: Distributed and embedded systems, real-time systems, fault-tolerance, active replication, graph transformation.

1. Introduction

Distributed and embedded real-time systems, such as transportation (e.g., aircrafts and automobiles), nuclear, robotics, and telecommunication, requires high dependability (Avizienis et al., 2000), where system failures during execution can causes catastrophic damages. These systems must function with high availability even under hardware and software faults. Fault-tolerance (Jalote, 1994) then becomes an important key to establish dependability in these systems. Hardware and software redundancy are well-known effective methods for hardware fault-tolerance (Guerraoui and Schiper, 1996), where extra hardware (e.g., processors, communication links) and software (e.g., tasks, messages) are added into the system to deal with hardware faults. However, hardware techniques based on *hardware* solutions are not preferred in most embedded systems due to the limited resources available, for reasons of weight, encumbrance, energy consumption, or price constraints.

In this paper, we present our most recent work for integrating fault-tolerance in SYNDEX (http://www-rocq.inria.fr/syndex), a system level CAD software tool for optimizing the implementation of real-time embedded applications on multicomponent architectures. The method we present extends (Girault et al., 2001; Girault et al., 2003) by tolerating also communication links failures, and is more general than (Dima et al., 2001) since it can tolerate an arbitrary number of processors failures and an arbitrary number of communication links failures.

The paper is organized as follows. Section 2 describes the related work. Section 3 presents the various models used by our method and states our fault-tolerance problem. Section 4 presents the proposed approach for providing fault-tolerance. Section 5 explains how to use our solution with some existing distribution-scheduling heuristics to generate fault-tolerant schedules. Finally, section 6 concludes and proposes directions for future research.

2. Related work

Related work in software fault tolerance approaches for distributed and embedded real-time systems falls in several categories. Relatively to fault hypothesis, we are interested in three fault-tolerant approaches: processors fault-tolerance, communication links fault-tolerance, and processors/communication links fault-tolerance.

In the first category of approaches that tolerates processors failures, several algorithms-based on scheduling heuristics have been proposed. They are based on *active software redundancy* (Breland et al., 1994; Hashimoto et al., 2002) or *passive software redundancy* (Ahn et al., 1997; Oh and Son, 1997). In the active redundancy technique, multiple redundant copies of a task are scheduled on different processors, which are run in parallel to tolerate a fixed number of processor failures. For instance, an off-line scheduling algorithm that tolerates a single processor failure in multiprocessor systems is presented in (Hashimoto et al., 2002). In the passive redundancy technique, also called primary/backup approach, a task is replicated on primary and backups replicas, but only the primary replica is executed. If it fails, one of the backups is selected to become the new primary. For instance, a fault-tolerant real-time scheduling algorithm that tolerate one processor failure in a heterogeneous distributed system is presented in (Qin et al., 2002), where failures are assumed to be permanent.

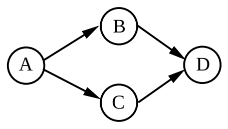
In the second category of approaches that tolerates communication links failures, several techniques have been proposed, which are based on *proactive* and *reactive* schemes. In the proactive scheme (Fragopoulou and Akl, 1995), multiple redundant copies of a message are sent along disjoint paths. However, in the reactive scheme (Sriram et al., 1999), one copy of the message, called primary, is sent, and if it fails, another copy of the message, called backup, will be transmitted.

Finally, the last category of approaches tolerates both processors and communication links failures (Gummadi et al., 2003; Zheng and Shin, 1998; Dima et al., 2001). For instance, in (Gummadi et al., 2003), failures are tolerated using the fault recovery scheme and a primary/backups strategy. Our solution is more general since it can tolerate *arbitrary* processors and communication links failures, and it may be used by *any* off-line distribution-scheduling heuristic to generate a fault-tolerant distributed code.

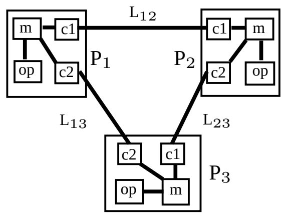
3. Models

3.1 Algorithm model

The algorithm is modeled by a *data-flow graph* ($\mathcal{A}lg$). Each vertex is an *operation* 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 $\triangleright$. Operations of $\mathcal{A}lg$ can be either an external input/output operation or a computation operation. Operations with no predecessor (resp. no successor) are the input interfaces (resp. output), handling the events produced by the sensors (resp. actuators). The inputs of a computation operation must precede its outputs. Moreover, computation operations are side-effect free, i.e. the output values depend only of the input values. The algorithm graph is executed repeatedly at each input event from the sensors in order to compute the output events for the actuators.



(a)



(b)

Figure 1. (a) Algorithm graph; (b) Architecture graph.

Figure 1(a) gives an example of $\mathcal{A}lg$, with four operations: A (sensor), B and C (computations), and D (actuator), and four data-dependences: $A \triangleright B$, $A \triangleright C$, $B \triangleright D$, and $C \triangleright D$.

3.2 Architecture model

The architecture is modeled by a graph ($\mathcal{A}rc$), where vertices are processors, and edges are bidirectional point-to-point communication links. In the sequel, we write "links" instead of "point-to-point communication links".

A processor P is a graph made of one operator op, one local memory m, and at least one communicator c. An operator op executes sequentially operations of $\mathcal{A}lg$, reads from and writes data into its local memory m. A communicator c_i cooperates with another communicator c_j in order to execute sequentially transfers of data stored in the memory (send or receive) between processors through a link.

Figure 1(b) gives an example of $\mathcal{A}rc$, with three processors P_1, P_2, and P_3, and three links L_{12}, L_{13}, and L_{23}, where each processor is made of one operator (op), one local memory (m), and two communicators ($c1$ and $c2$).

To each operator op we associate a list of pairs (o,d/op), where d is the worst case execution time (WCET) of the operation o on operator op. Also, to each communicator c, we associate a list of pairs (dpd,d/c), where d is the worst case transmission time (WCTT) of the data-dependence dpd on communicator c. Since the target architecture is heterogeneous, the WCET (resp. WCTT) for a given operation (resp. data-dependence) can be distinct on each operator (resp. communication link).

3.3 Failure model

We consider only processors and communication links failures, where failures are assumed to be a fail-silent (also known as fail-stop), i.e. a component works correctly or stops functioning (becomes silent). Recent studies on modern processors have shown that a fail-silent behavior can be achieved at a reasonable cost (Baleani et al., 2003). We assume that at most $\mathcal{N}_{PF}$ processors and $\mathcal{N}_{LF}$ links may fails.

As we consider off-line distribution-scheduling heuristics, the execution of operations and communications are time-triggered (Kopetz and Bauer, 2002), that is, each operation and communication is assigned two start-dates: St_{best} in the absence of failure and St_{worst} in the presence of failures.

4. The proposed approach

In this section, we present our approach based on software redundancy to tolerate processor and link failures. We propose to use *graph transformation* to perform software redundancy, where a given input algorithm graph ($\mathcal{A}lg$) is transformed into a new algorithm graph ($\mathcal{A}lg^*$) augmented with redundancies. Then, operations and data-dependences of $\mathcal{A}lg^*$ can be distributed and scheduled on a specified target distributed architecture ($\mathcal{A}rc$) to generate a fault tolerant distributed schedule. The global picture of our methodology is shown in Figure 2.

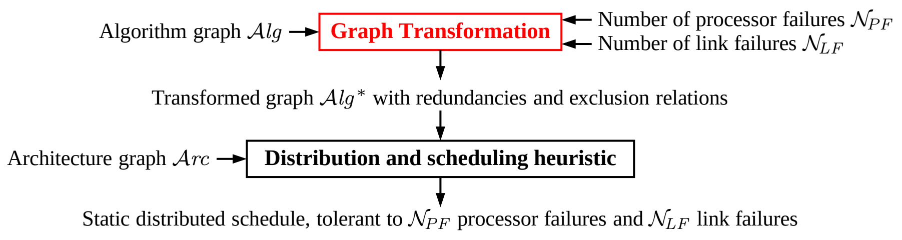

Algorithm graph $\mathcal{A}lg$
Graph Transformation
Number of processor failures $\mathcal{N}_{PF}$
Number of link failures $\mathcal{N}_{LF}$
Transformed graph $\mathcal{A}lg^*$ with redundancies and exclusion relations
Architecture graph $\mathcal{A}rc$
Distribution and scheduling heuristic
Static distributed schedule, tolerant to $\mathcal{N}_{PF}$ processor failures and $\mathcal{N}_{LF}$ link failures

Figure 2. Global picture of our methodology.

In this section, we concentrate on the first step, the transformation of $\mathcal{A}lg$ into $\mathcal{A}lg^*$. We first present a method that tolerates only processors failures, next we present a method that tolerates only links failures, and finally we present a combined method that tolerates both processors and links failures.

4.1 Tolerating processor failures

In order to tolerate at most $\mathcal{N}_{PF}$ processor failures, we propose to use the same principle as in (Girault et al., 2003): each operation has $\mathcal{N}_{PF}$+1 replicas scheduled on $\mathcal{N}_{PF}$+1 distinct processors. The system's communication links are assumed to be fault-free.

The transformation of $\mathcal{A}lg$ into $\mathcal{A}lg^*$ is performed in two steps. Initially, each operation o_i of $\mathcal{A}lg$ is replicated in $\mathcal{A}lg^*$ on $\mathcal{N}_{PF}$+1 *exclusive replicas* o_i^1, $\ldots$, $o_i^{\mathcal{N}_{PF}+1}$ (the set of o_i's replicas is noted $\mathcal{R}ep(o_i)$); two operations o_i^1 et o_i^2 are exclusive if and only if they are two identical replicas of the same operation o_i and they must be scheduled on distinct processors. In the second step, each replicated operation of $\mathcal{A}lg^*$ must receive its inputs data $\mathcal{N}_{PF}$+1 times from each of its predecessors. Therefore, each data-dependence $data_i$ of $\mathcal{A}lg$ is replicated in $\mathcal{A}lg^*$ on $\mathcal{N}_{PF}$+1 exclusive replicas $data_i^1$, $\ldots$, $data_i^{\mathcal{N}_{PF}+1}$; two data-dependences $data_i^1$ et $data_i^2$ are exclusive if and only if they are two identical replicas of the same data-dependence $data_i$ and they must be scheduled on disjoint paths (see Figure 3(a) for $\mathcal{N}_{PF}$=1).

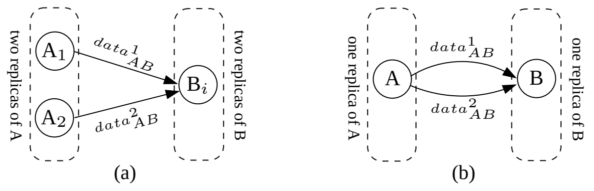


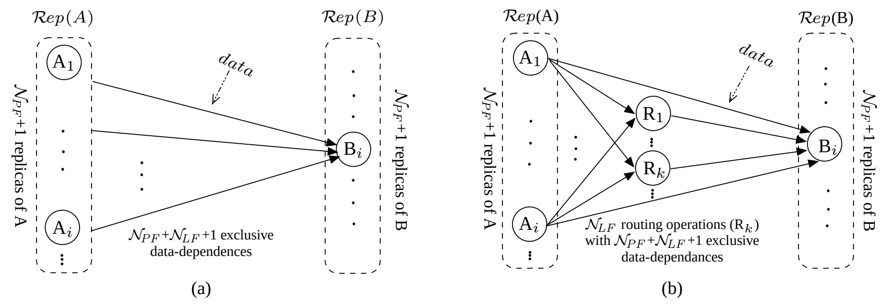
Figure 3. (a) Processor failures; (b) Link failures.

4.2 Tolerating link failures

To tolerate at most $\mathcal{N}_{LF}$ link failures, we propose to use the same principle as in tolerating processors failures, which is based on the following graph transformation. We don't need to replicate operations because the processors are assumed to be fault-free. Therefore, each data-dependence $data_i$ of $\mathcal{A}lg$ is replicated in $\mathcal{A}lg^*$ on $\mathcal{N}_{LF}$+1 exclusive replicas $data_i^1$, $data_i^2$, $\ldots$, $data_i^{\mathcal{N}_{LF}+1}$ between any two dependent operations (see Figure 3(b) for $\mathcal{N}_{LF}$=1).

4.3 Tolerating processor and link failures

To tolerate at most $\mathcal{N}_{PF}$ processor and $\mathcal{N}_{LF}$ link failures, we first replicate each operation o_i of $\mathcal{A}lg$ in $\mathcal{A}lg^*$ on $\mathcal{N}_{PF}$+1 exclusive replicas (set $\mathcal{R}ep(o_i)$), as shown in Figure 4(a), wherein operations A and B of Figure 1(a) are replicated on $\mathcal{N}_{PF}$+1 exclusive replicas. Then, each replicated operation of $\mathcal{A}lg^*$ must receive its input data $\mathcal{N}_{PF}$+$\mathcal{N}_{LF}$+1 times from each of its predecessors. Therefore, each data-dependence $data_i$ of $\mathcal{A}lg$ is replicated in $\mathcal{A}lg^*$ on $\mathcal{N}_{PF}$+$\mathcal{N}_{LF}$+1 exclusive replicas, as shown in Figure 4(a), wherein the data-dependence A $\triangleright$ B is replicated $\mathcal{N}_{PF}$+$\mathcal{N}_{LF}$+1 times between the replicas $\mathcal{R}ep$(A) of A and each replica of B.



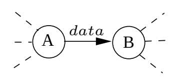
$\mathcal{R}ep(A)$
$\mathcal{R}ep(B)$
A_1
A_i
B_i
$data$
$\mathcal{N}_{PF}$+1 replicas of A
$\mathcal{N}_{PF}$+1 replicas of B
$\mathcal{N}_{PF}$+$\mathcal{N}_{LF}$+1 exclusive data-dependences
(a)
$\mathcal{R}ep$(A)
$\mathcal{R}ep$(B)
R_1
R_k
$\mathcal{N}_{LF}$ routing operations (R_k) with $\mathcal{N}_{PF}$+$\mathcal{N}_{LF}$+1 exclusive data-dependances
(b)

Figure 4. (a) Initial transformations for $\mathcal{N}_{PF} \geq 0$ and $\mathcal{N}_{LF} \geq 0$; (b) Final transformations.

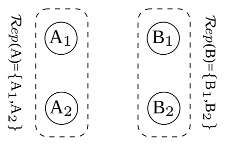
The problem of this scheme is to find a distribution of these exclusive dependences of $\mathcal{A}lg^*$ between the replicas $\mathcal{R}ep$(A) of A. The requirement is to tolerate only $\mathcal{N}_{PF}$ processor failures and $\mathcal{N}_{LF}$ link failures. Therefore, we propose a distribution which is *less expensive* in terms of communications.

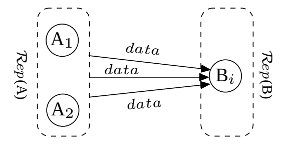
To present as clearly as possible our distribution technique, we present initially its principles in the case $\mathcal{N}_{PF}$=1 and $\mathcal{N}_{LF}$=1 for the algorithm sub-graph of Figure 5(a). Figures 5(b) and 5(c) are the two first steps of our approach, before the distribution itself, which is performed in two steps, illustrated in Figures 5(d) and 5(e). Since $\mathcal{N}_{PF}$=1 and $\mathcal{N}_{LF}$=1, $\mathcal{R}ep$(A) and $\mathcal{R}ep$(B) contain each two replicas. Furthermore, the data-dependence A $\triangleright$ B is replicated three times.

First, we connect each replica of A with one of these three data-dependences, as shown in Figure 5(d). Next, we first replace the third data-dependence by a new operation R_1. We call this new operation a *routing operation*, its duration is *null* (the set of all routing operations from A to B is noted $\mathcal{R}ep(A \triangleright B)$). Finally, we connect all the replicas $\mathcal{R}ep(A)$ of A to R_1 which is also connected to each replica of B, as shown in Figure 5(e). The purpose is to make sure that each replica of B has *three* distinct sources from which it will receive the data-dependence $A \triangleright B$, so that if any two sources fail (two because here $\mathcal{N}_{PF}$+$\mathcal{N}_{LF}$=2), then these failures will be masked by the third source. Thus, for any i, operations A_1, A_2 and R_i are exclusive and must be implemented on distinct processors. Also, the data-dependences $A_1 \triangleright B_i$, $A_2 \triangleright B_i$, and $R_i \triangleright B_i$ are exclusive and must be implemented on disjoint paths. Such exclusive relations are given with the final transformed graph $\mathcal{A}lg^*$ to the distribution/scheduling heuristic (see Figure 2).

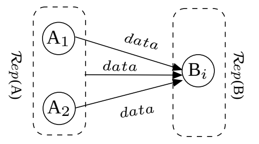


(a) Initial algorithm sub-graph.

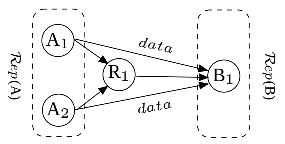

$\mathcal{R}ep(A)=\{A_1,A_2\}$
A_1
A_2
B_1
B_2
$\mathcal{R}ep(B)=\{B_1,B_2\}$
(b) Operation redundancy.



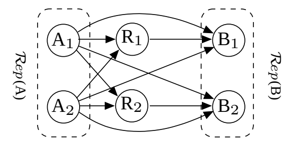
(c) Data-dependance redundancy.



(d) Data-dependance distribution (1).



(e) Data-dependance distribution (2).



(f) Final algorithm sub-graph.

Figure 5. Distribution scheme for $\mathcal{N}_{PF}$=1 and $\mathcal{N}_{LF}$=1.

In the general case, $\mathcal{N}_{PF} \geq 0$ and $\mathcal{N}_{LF} \geq 0$, the transformation scheme of $\mathcal{A}lg$ on $\mathcal{A}lg^*$ is illustrated in Figure 4(b), where each operation is replicated in $\mathcal{A}lg^*$ on $\mathcal{N}_{PF}$+1 exclusive replicas, and each data-dependence is replaced by $\mathcal{N}_{LF}$ routing operations R_k, $\mathcal{N}_{PF}$+1 data-dependences between $\mathcal{R}ep(A)$ and each R_k, and one data-dependence between each R_k and each B_i. The operations in $\mathcal{R}ep(A)$ and in $\mathcal{R}ep(A \triangleright B)$ are exclusive and must be implemented on distinct processors. Also, all replicated data-dependences $A \triangleright B$ and $R \triangleright B$ are exclusive and must be implemented on disjoint paths.

5. Distribution/scheduling heuristic

Our method may use the distribution-scheduling heuristic DSH proposed in (Grandpierre et al., 1999). As required by our graph transformation method, we modify the DSH heuristic to take into account the exclusive relations between operations and data-dependences. The modified heuristic is formally described in Figure 6. We use the two functions $succ(o)$ and $pred(o)$ to denote the sets of successor and predecessor operations of o in $\mathcal{A}lg^*$.

```
Inputs = a transformed algorithm graph $\mathcal{A}lg^*$, a list of exclusions $\mathcal{E}xcl$, an architecture
        graph $\mathcal{A}rc$, some real-time and distribution constraints;
Output = a fault-tolerant distributed static schedule of $\mathcal{A}lg^*$ onto $\mathcal{A}rc$;
Begin
  $O^{(0)}_{cand} := \{$operations of $\mathcal{A}lg^*$ without any predecessor$\}$;   $O^{(0)}_{sched} := \emptyset$;
  Repeat
    1. For each $B_i \in O^{(n)}_{cand}$ do
        1.1. If $B_i$ is a routing operation then Compute the set $\mathcal{P}(B_i)$ of processors,
             such that for each $p \in \mathcal{P}$:
               a. There exist $\mathcal{N}_{PF}$+1 disjoint paths from processors implementing
                  operations of $\mathcal{R}ep(A_j)$ to $p$, for any $A_j$ predecessor of $B_i$;
               b. $p$ must comply to all exclusions in $\mathcal{E}xcl$ concerning $B_i$;
        1.2. Else Compute the set $\mathcal{P}(B_i)$ of processors, such that for each $p \in \mathcal{P}$:
               a. There exist $\mathcal{N}_{PF}$+$\mathcal{N}_{LF}$+1 disjoint paths from processors
                  implementing operations in $\mathcal{R}ep(A_j) \cup \mathcal{R}ep(A_j \triangleright B_i)$ to $p$;
               b. $p$ must comply to all exclusions in $\mathcal{E}xcl$ concerning $B_i$; End if
        1.3. If the set $\mathcal{P}(B_i)$ is empty then return "Fail"; End if
       End for each
    3. Select the most urgent operation $B_{urgent}$ of $\mathcal{A}lg^*$ using the schedule pressure
       and the sets $\mathcal{P}(B_i)$;
    4. Select the best processor $p_{best}$ from the set $\mathcal{P}(B_{urgent})$;
    5. For all $A_j$ predecessor of $B_{urgent}$, schedule all the data-dependences $A_j \triangleright B_{urgent}$
       on several paths leading to $p_{best}$, such that these paths comply to the exclusions
       in $\mathcal{E}xcl$ concerning $A_j \triangleright B_{urgent}$;
    6. Schedule $B_{urgent}$ onto the processor $p_{best}$;
    7. $O^{(n+1)}_{sched} := O^{(n)}_{sched} \cup \{B_{urgent}\}$;
    8. $O^{(n+1)}_{cand} := O^{(n)}_{cand} - \{B_{urgent}\} \cup \{o \in succ(B_{urgent}) \mid pred(o) \subseteq O^{(n+1)}_{sched}\}$;
  Until $O^{(n+1)}_{cand} = \emptyset$;
End
```

Figure 6. The DSH distribution-scheduling heuristic.

At each step (n) of the heuristic, for each candidate operation B_i in $O^{(n)}_{cand}$, we compute the set $\mathcal{P}(B_i)$ of processors that can execute B_i and comply to the concerned exclusions of $\mathcal{E}xcl$. Then, the most urgent candidate operation B_{urgent} is selected to be scheduled thanks to the *schedule pressure* function defined in (Grandpierre et al., 1999). Then, among the set $\mathcal{P}(B_{urgent})$, the processor p_{best}, where B_{urgent} will finish at the earliest date, is selected to execute B_{urgent}. But before B_{urgent} is actually scheduled onto p_{best}, all the required data-dependences are scheduled on paths, possibly disjoint depending again on the concerned exclusions of $\mathcal{E}xcl$. Finally, the lists $O^{(n)}_{sched}$ of scheduled operations and $O^{(n)}_{cand}$ of candidate operations are updated.

Finally, the proposed general transformation scheme enables us to generate a fault-tolerant distributed schedule of the new algorithm graph $\mathcal{A}lg^*$ onto the architecture graph $\mathcal{A}rc$. The following theorem proves that it is, by construction, tolerant to any combination of at most $\mathcal{N}_{PF}$ processors failures and at most $\mathcal{N}_{LF}$ communication links failures.

THEOREM 1 *Let $\mathcal{A}lg$ be an algorithm graph and $\mathcal{A}rc$ an architecture graph. Let $\mathcal{A}lg^*$ be the new algorithm graph obtained by applying the transformation of Figure 4(b) to $\mathcal{A}lg$. Let $\mathcal{S}ys$ be the system obtained by distributing and scheduling $\mathcal{A}lg^*$ onto $\mathcal{A}rc$ w.r.t. the exclusion relations required by the graph transformation having led to $\mathcal{A}lg^*$. If at most $\mathcal{N}_{PF}$ processor failures and $\mathcal{N}_{LF}$ communication links failures occur in $\mathcal{S}ys$, then at least one replica of each operation will remain active.*

Proof. Since each operation is replicated $\mathcal{N}_{PF}$+1 times, since all these $\mathcal{N}_{PF}$+1 replicas are scheduled onto distinct processors, and since at most $\mathcal{N}_{PF}$ processors can fail simultaneously, then at least one replica of each operation is scheduled onto a processor that will remain valid. We therefore need to prove that any operation scheduled onto an active processor is active, i.e., that it receives correctly all its required inputs from all its predecessor operations.

Let o_j^m be such an operation, namely the mth replica of operation o_j. For each of its predecessor operations o_i, thanks to the same argument as above, there exists at least one replica o_i^k scheduled onto a valid processor. By construction, there exist $\mathcal{N}_{LF}$+1 data-dependences between o_i^k and o_j^m. Thanks to the exclusion lists given to the distribution/scheduling heuristic, these $\mathcal{N}_{LF}$+1 data-dependences are scheduled onto $\mathcal{N}_{LF}$+1 disjoint paths. Since at most $\mathcal{N}_{LF}$ communication links can fail simultaneously, then at least one of these data-dependences will remain valid. Hence o_j^m will receive correctly its input from o_i^k. We have thus proved that any operation scheduled onto an active processor will receive correctly all its input data from all its predecessors in $\mathcal{A}lg$, and will therefore be executed correctly. $\square$

6. Conclusion

We have investigated methods to mask hardware failures in heterogeneous distributed systems with point-to-point communication links. We have proposed a new method that tolerates at most $\mathcal{N}_{PF}$ arbitrary processors and at most $\mathcal{N}_{LF}$ arbitrary communication links failures. It is a software solution, based on active redundancy to mask the hardware failures. It proceed in two steps: first a graph transformation, and then an off-line distribution-scheduling heuristic. The graph transformation adds software redundancy to the original algorithm graph $\mathcal{A}lg$: we obtain a new algorithm graph $\mathcal{A}lg^*$ with redundancy, along with exclusion relations. Then, the distribution-scheduling heuristic is applied to map $\mathcal{A}lg^*$ onto a given architecture graph $\mathcal{A}rc$. As a result, it generates a static schedule of $\mathcal{A}lg^*$ onto $\mathcal{A}rc$, which is tolerant to the required processors and communication links failures.

Currently, we are working on a new solution to take into account distributed architectures with bus communication links. We also plan new solution to take sensors/actuators failures into account.

References

Ahn, K., Kim, J., and Hong, S. (1997). Fault-tolerant real-time scheduling using passive replicas. In *PRFTS'97*, Taipei, Taiwan.

Avizienis, A., Laprie, J.-C., and Randell, B. (2000). Fundamental concepts in dependability. In *ISW-2000*, pages 7–12, Boston, Massachusetts, USA.

Baleani, M., Ferrari, A., Mangeruca, L., Peri, M., Pezzini, S., and Sangiovanni-Vincentelli, A. (2003). Fault-tolerant platforms for automotive safety-critical applications. In *CASES'03*, San Jose, USA. ACM.

Breland, M.A., Rogers, S.A., Brat, G., and Nelson, K.L. (1994). Transparent fault-tolerance for distributed Ada applications. In *Proceedings of the Conference on TRI-Ada '94*, pages 446–457. ACM Press.

Dima, C., Girault, A., Lavarenne, C., and Sorel, Y. (2001). Off-line real-time fault-tolerant scheduling. In *Euromicro PDP'01*, pages 410–417, Mantova, Italy.

Fragopoulou, P. and Akl, S.G. (1995). Fault tolerant communication algorithms on the star network using disjoint paths. In *Proceedings of the HICSS'95*, Kingston, Canada.

Girault, A., Kalla, H., Sighireanu, M., and Sorel, Y. (2003). An algorithm for automatically obtaining distributed and fault-tolerant static schedule. In *DSN'03*, San Francisco, USA.

Girault, A., Lavarenne, C., Sighireanu, M., and Sorel, Y. (2001). Fault-tolerant static scheduling for real-time distributed embedded systems. In *ICDCS'01*, pages 695–698, Phoenix, USA. IEEE. Extended abstract.

Grandpierre, T., Lavarenne, C., and Sorel, Y. (1999). Optimized rapid prototyping for real-time embedded heterogeneous multiprocessors. In *7th International Workshop on Hardware/Software Co-Design, CODES'99*, Rome, Italy.

Guerraoui, R. and Schiper, A. (1996). Fault-tolerance by replication in distributed systems. In *Proceeding Conference on Reliable Software Technologies*, pages 38–57. Springer-Verlag.

Gummadi, K.P., Pradeep, M.J., and Murthy, C.S. Ram (2003). An efficient primary-segmented backup scheme for dependable real-time communication in multihop networks. *IEEE/ACM Trans. on Networking*, 11(1).

Hashimoto, K., Tsuchiya, T., and Kikuno, T. (2002). Effective scheduling of duplicated tasks for fault-tolerance in multiprocessor systems. *IEICE Transactions on Information and Systems*.

Jalote, P. (1994). *Fault-Tolerance in Distributed Systems*. Prentice Hall, Englewood Cliffs, New Jersey.

Kopetz, H. and Bauer, G. (2002). The time-triggered architecture. *Proceedings of the IEEE, Special Issue on Modeling and Design of Embedded Software*.

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

Qin, X., Jiang, H., and Swanson, D.R. (2002). An efficient fault-tolerant scheduling algorithm for real-time tasks with precedence constraints in heterogeneous systems. In *ICPP'02*, pages 360–386, Vancouver, Canada.

Sriram, R., Manimaran, G., and Murthy, C. Siva Ram (1999). An integrated scheme for establishing dependable real-time channels in multihop networks. In *ICCC'90*, pages 528–533.

Zheng, Q. and Shin, K. G. (1998). Fault-tolerant real-time communication in distributed computing systems. In *IEEE Trans. on Parallel and Distributed Systems*, pages 470–480.