# An Integrated Strategy for Automatically Obtaining Degraded Functionality Mode

Chouki Abderrezak
Department of computer science,
University of Batna, Algeria
chouki.abderrezak@gmail.com

*Abstract-* **Our goal is to automatically define degraded functionality mode in case of transient timing fault or processor silent failure in embedded and real time systems. The proposed solution is different from the previously proposed approaches in a way that automatically defines degraded functionality mode to tolerate faults. Our approach is an integrated strategy into static real time scheduling algorithm to automatically obtain degraded functionality mode. Multiple queues are used for different importance levels. Activated task is placed in adequate queue according to its importance. Running task is interrupted by more important task, at the head of queue, which obtains its latest start time. Results show that the proposed solution can tolerate faults with automatically obtaining degraded functionality mode.**

## I. Introduction

Embedded systems account for a major part of critical applications (space, aeronautic, nuclear…) as well public applications (automotive, consumer, electronics…) [1].

Their main feature is c*ritical real time,* critical because the failure of system's component can lead to catastrophe.

For this, *dependability* of such real time system must be taken into account. A fundamental mechanism by which dependability can be achieved is *fault tolerance. Fault tolerance* in a real-time system implies that the system is able to deliver correct in a timely manner even in the presence of faults [7]. Levels of fault tolerance are:

- *Full fault tolerance:* the system continues to operate in the presence of faults, albeit for a limited period, with no significant loss of functionality or performance;
- *Graceful Degradation:* the system continues to operate in the presence of errors, accepting a partial degradation of functionality or performance during recovery or repair.

Most safety critical systems require full fault tolerance. However, this solution failed in case where failure hypothesis of the system are not respected, for example in presence of too failure of processors in multiprocessor system. For this, degraded functionality mode is necessary in such critical system.

In this paper, we propose a strategy, which is called ADAF (strategy to Automatically Define degrAded Functionality mode), that automatically defines degraded functionality mode. We do not address the scheduling, nor the placement of the tasks as discussed in [1].our approach is an integrated strategy into static real time scheduling algorithm to automatically obtain degraded functionality mode.

The remainder of this paper is organized as follows. In the next section we present related work. Next we describe our proposed approach. Finally, an example and conclusion are presented.

## II. Related work

In the literature, we can identify two full fault tolerance approaches:

- Hardware fault tolerance;
- Software fault tolerance.

Most hardware fault tolerance techniques based on hardware redundancy are not preferred in embedded systems due to the limited resources available .For this, software fault tolerance approach increasingly used in embedded systems.
Software fault tolerance techniques are:

- *TMR (Triple Modular Redundancy)*: multiple copies are executed and error checking is achieved by comparing after completion [1].
- *PB (Primary/ Backup)*: the tasks are assumed to be periodic and two instances of each task (a primary and a backup) are scheduled on a single processor system [2], [3], [10].
- *PE (Primary/Exception)*: it is the same as PB method except that exception handlers are executed instead of backup programs [11].

Contrary to previously techniques, that use full fault tolerance approach, we propose an automatic solution to define degraded functionality mode to tolerate faults.

## III. The Proposed Strategy

In this section, we present the proposed strategy ADAF. As we discussed earlier, full fault tolerance is used in previously proposed techniques. The proposed solution is different from the previously approaches in a way that can automatically obtain degraded functionality mode to tolerate faults. The ADAF consists of two steps, one offline and the second online:

1) Change of Deadline and determination of latest start time of tasks (offline);
2) The scheduling strategy (online).

Table I provides some terms and their meanings that will be used in the subsequent discussion.

TABLE I
SYMBOLS AND THEIR MEANINGS

| Symbol | Meaning |
|---|---|
| $C_i$ | The worst case execution time of the task $T_i$ |
| $D_i$ | The deadline of the task $T_i$ |
| $S_i$ | The latest possible start time of the task $T_i$ |
| $M_i$ | The importance of the task $T_i$ |

### *A. Change of Deadline and determination of latest start time*

In the first step of ADAF, we change deadline and latest start time of tasks to avoid case where a less important task retards the more important task. We change deadline of the task $T_i$ if there is a task $T_j$ with: $D_i + C_j > D_j$.
The new deadline of $T_i$ is: $D'_i = D_j - C_j$.
The latest possible start time of the task $T_i$ is calculated using the following formula: $S_i = Min\ ( D_i\ , D'_i\ ) - C_i$.
The first step of ADAF is formally described below:

```
The ADAF first step Algorithm
begin
    for each task $T_i$ do
        $D'_i \longleftarrow D_i$;
    end for
    for each two tasks ( $T_i$, $T_j$ ) do
        if $D_i < D_j$ and $D_i + C_j > D_j$ then
            $D'_i \longleftarrow D_j - C_j$;
        end if
    end for
    for each task $T_i$ do
        $S_i = Min ( D_i, D'_i ) - C_i$
    end for
end
```

### *B. The scheduling strategy*

In this section, we describe the second step of ADAF, which is the more important, that automatically defines degraded functionality mode. The second step is a scheduling strategy that ignores less important tasks in case of timing faults or processor silent failure. In this step, we assume that tasks are scheduled offline using an ordinary static real time scheduling algorithm.

In our strategy, we use multiple queues for different importance levels. Each activated task is placed in adequate queue according to its importance. The running task is interrupted by more important task, at the head of queue, which obtains its latest start time.

A static real time scheduling algorithm, with our integrated strategy, is formally described below:

```
The static real time scheduling algorithm
begin
Schedule tasks using an ordinary static real time scheduling algorithm;
Change deadline and determinate latest start time of tasks using the ADAF first step algorithm;
Place scheduled tasks in queues according to its importance;
While exist tasks in queues do
    Allocate the processor to the more important task at the head of queue;
    While not terminate currently running task do
        If exist a more important task obtains its latest start time then
            Interrupt currently running task;
            Allocate the processor to the more important task at the head of queue;
        end if
    end while
end while
end
```

## IV. An Example

In this section, an example is used to illustrate the effectiveness of the proposed strategy. We discuss two cases of failure: transient timing fault on single processor to show the effectiveness of ADAF in centralized systems, and processor silent failure on two processors system to show the advantages of ADAF in distributed systems.

### *A. Transient timing faults on single processor*

The following table shows an example of tasks, with its worst case execution time ($C_i$), deadline ($D_i$), importance ($M_i$), and latest possible start time ($S_i$), used to illustrate the effectiveness of our proposed solution ADAF in case of transient timing fault on single processor system.

TABLE II
An example of tasks

| task | $C_i$ | $D_i$ | $M_i$ | $S_i$ |
|---|---|---|---|---|
| $T_1$ | 5 | 12 | 3 | 7 |
| $T_2$ | 2 | 4 | 2 | 2 |
| $T_3$ | 1 | 3 | 1 | 2 |
| $T_4$ | 3 | 6 | 3 | 3 |

We assume that tasks are scheduled offline according to its deadlines in order: $T_3$, $T_2$, $T_4$ and $T_1$. We use the ADAF first step algorithm to change deadline and determinate the latest start time of tasks. The result is shown in the following table.

TABLE III
New deadline and latest start time of tasks

| task | $D'_i$ | $S_i$ |
|---|---|---|
| $T_1$ | 12 | 7 |
| $T_2$ | 3 | 1 |
| $T_3$ | 1 | 0 |
| $T_4$ | 6 | 3 |

In the follow, the Gantt chart of scheduled tasks with ADAF without faults.

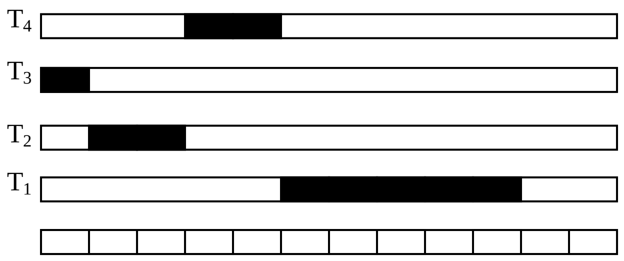

Fig. 1. Gantt chart of scheduled tasks with ADAF without faults.

We assume that there is a timing fault t between 1 and 3 time unit. The following Gantt chart presents scheduled tasks with ADAF in case of the transient timing fault t on single processor system.

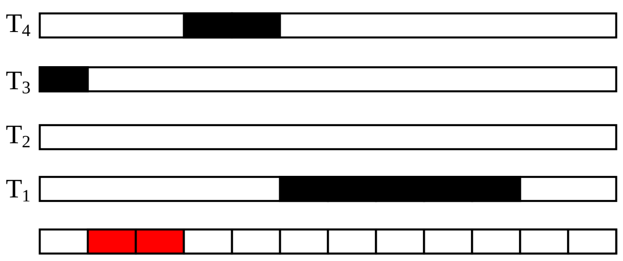

Fig. 2. Gantt chart of scheduled tasks with ADAF in case of transient timing fault on single processor system.

In case of the transient timing fault t, with using ADAF, Fig. 2 shows that $T_2$ is ignored. We automatically obtain degraded functionality mode with execution of tasks $T_1$, $T_3$ and $T_4$.

## *B. Processor silent failure on two processors system*

This section describes an example in case of processor silent failure on two processors system. We assume that the first processor is allocated to tasks presented in Table II and the second processor is allocated to tasks shown in the following table.

TABLE IV
SET OF TASKS ASSIGNED TO THE SECOND PROCESSOR

| task | $C_i$ | $D_i$ | $M_i$ | $S_i$ |
|---|---|---|---|---|
| $T'_1$ | 3 | 6 | 2 | 3 |
| $T'_2$ | 2 | 4 | 3 | 2 |
| $T'_3$ | 3 | 10 | 1 | 7 |
| $T'_4$ | 2 | 8 | 2 | 6 |

In case of second processor failure, tasks presented in the Table IV are migrated to the first processor.
At first we change deadline and latest possible start time of tasks presented in Table IV using ADAF first step algorithm. The result of this step is shown in the following table.

TABLE V
NEW DEADLINES AND LATEST POSSIBLE START TIME OF TASKS ASSIGNED TO THE SECOND PROCESSOR

| task | $D'_i$ | $S_i$ |
|---|---|---|
| $T'_1$ | 5 | 2 |
| $T'_2$ | 2 | 0 |
| $T'_3$ | 10 | 7 |
| $T'_4$ | 7 | 5 |

In case of second processor failure, and after migrate tasks; we change deadline and latest start time of more important tasks.
In our example, Table VI shows new deadline and latest start time of the more important tasks $T_1$, $T'_2$ and $T_4$.

TABLE VI
NEW DEADLINES AND LATEST START TIME OF MORE IMPORTANT TASKS

| task | $D'_i$ | $S_i$ |
|---|---|---|
| $T_1$ | 12 | 7 |
| $T'_2$ | 3 | 1 |
| $T_4$ | 6 | 3 |

The following Gantt chart presents scheduled tasks with ADAF in case of second processor silent failure on two processors system.

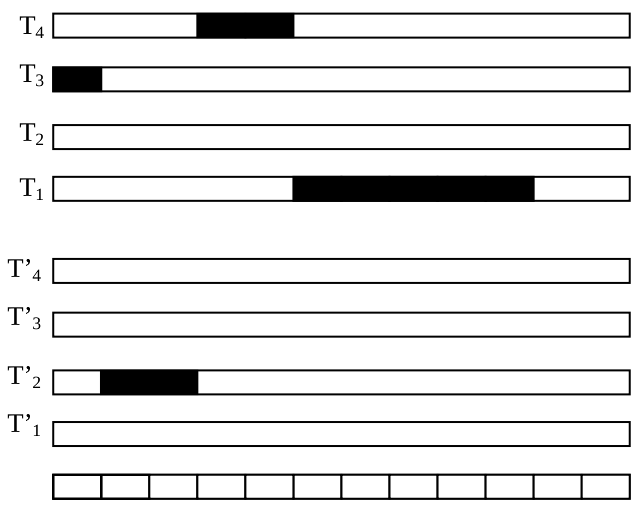

Fig. 3. Gantt chart of scheduled tasks with ADAF in case of second processor silent failure on two processors system.

In case of second processor silent failure, Fig. 3 shows only $T_1$, $T_3$, $T_4$ and $T'_2$ are executed and the other tasks are ignored. We automatically obtain degraded functionality mode with the execution of tasks $T_1$, $T_3$, $T_4$ and $T'_2$.

## V. CONCLUSION AND FUTURE WORK

Critical real time is the main feature of embedded systems; critical because the failure of system's component can lead to catastrophe. For this, dependability of such embedded system must be taken into account from the beginning. A fundamental mechanism by which dependability can be achieved is fault tolerance.

Most safety critical systems require full fault tolerance approache. However, this solution failed in case where failure hypothesis are not respected.

For this, degraded functionality mode is necessary in such critical system. A strategy for automatically obtaining degraded functionality mode is proposed.

Results show that ADAF automatically obtains degraded functionality mode in order to tolerate transient timing faults and processor silent failure. Future work will be to extend the proposed approach to dynamic real time scheduling algorithm.

## REFERENCES

[1] A. Girault, H. Kalla, M.Sighireanu, and Y. Sorel, "An algorithm for automatically obtaining distributed and fault-tolerant static schedules," *In IEEE Int. Conf. on Dependable Systems and Networks, San-Francisco, USA*, June 2003.

[2] *Y. Oh, S.H. Son,* "An algorithm for real-time fault-tolerant scheduling in multiprocessor systems," *in Proc. Euromicro Workshop on Real-Time Systems, Greece, 1992*, pp. *190-195A.*

[3] S. Ghosh, R. Melhem, D. Mosse, and J. S.Sarma, "Fault-tolerant rate-monotonic scheduling," *Journal of Real-Time Systems*, 1998.

[4] B.W.Johson, "*Design and Analysis of Fault Tolerant Digital Systems*". Addison Wesley Pub. *Co, Inc, 1989.*

[5] B. Mirle and A.M.K. Cheng, "*Simulation of fault-tolerant scheduling on real-time multiprocessor systems using Primary Backup overloading,*" Technical Report UH-CS-06-04, University of Houston, May 21, 2006.

[6] F. Cridtan, "Understanding fault-tolerant distributed systems," *communication of the ACM*, Vol-34(2), pp 57-78, 1991.

[7] S. Punnekkat, "*Schedulability Analysis for Fault tolerant Real-Time Systems,*" PhD thesis, University of York, 1997.

[8] H. Kalla, "*Automatique Generation of Fault Tolerant, Reliable, and Real Time Distribution/Scheduling,*" PhD thesis, Polytechnic National Institute of Grenoble, December 2004.

[9] M.Y.K. Kwok and I. Ahmad, "Dynamic critical- path scheduling: an effective technique for allocating task graphs to multiprocessors,"*IEEE Transaction on parallel and Distributed Systems*, Vol 7, NO. 5, May 1996.

[10] H. Zou and F. Jahabian, "Real-time primary-backup (RTPB) replication with temporal consistency guarantees,"*IEEE Transactions on Parallel and Distributed systems*, v.10 n.6, p.533-548, 1998.

[11] O. Gonzlez, H. Shrikumar,J.A. Stankovic, *and* K. Ramanritham, "Adaptative fault tolerant and graceful degradation under dynamic hard real-time scheduling". *In Proc.of the 18 th IEEE Real- Time Systems symposium, San Fransisco*, CA, Dec.2-5, 1997.