Scheduling Analysis under Fault Bursts

Florian Many, David Doose

Scheduling Analysis under Fault Bursts

Florian MANY, David DOOSE
ONERA - DTIM
2 avenue Edouard Belin
31055 Toulouse Cedex 4
firstname.lastname@onera.fr

Abstract

Real time systems must respect their temporal constraints both in nominal and degraded conditions. Environment disturbances cause faults which are revealed by errors during task execution. Therefore, schedulers must be fault tolerant to guarantee no missed deadline. Phenomena like electromagnetic fields disturb real-time systems on a extended period of time. It is difficult to forecast faults and their consequences to build efficient fault-tolerant systems. The classical fault models deal with pseudo-periodic faults. They are not made for phenomena extended in time. This paper intends to describe electromagnetic disturbances in a new fault model, named fault burst *model. In adequation with the fault burst model, we provide error recovery strategies. Finally, we study the effects of strategies on the schedulability analysis to guarantee fault tolerance when* fault bursts *occur.*

1 Introduction

The reliability of a real-time system made of several tasks is not only based on the nominal behaviour of these tasks, but should also take into account the effect of hardware faults on the task behaviour. Two complementary approaches are used to limit the effect of faults: spacial redundancy (duplication, triplication of the most critical hardware) and time redundancy (robust data model, re-execution of erroneous code). In small systems like Unmanned Aerial Vehicles (UAV) or satellites, the use of spacial redundancy is limited, so it becomes essential to use time redundancies to guarantee the reliability of the real-time system.

The temporal distribution of faults must be known to establish an efficient choice of time redundancy. The classical fault model, named *pseudo-periodic fault* model, deals with isolated faults. A minimum time interval between two faults is assumed [10, 9, 14, 3, 8]. This fault model is efficient to manage either punctual faults or disturbances caused by electromagnetic compatibility. For example, in an aircraft, these distrurbances can ge generated by a power supply near an equipment. In this case, the punctual faults are periodic according to the electromagnetic field produced by the power supply.

However this fault model is not designed to manage phenomena that cause potentially random faults over a bounded time interval. This phenomena can be a dysfunction of a data sensor, like an object in front of a camera, or disturbances due to *some* electromagnetic fields, like those produced by the radar waves in airport areas [16, 17]. These phenomena must be included in a complementary model in order to evaluate the fault tolerance of real time systems in external conditions that can be agressive.

In this paper we define a fault model named *fault burst*, to describe this kind of phenomenon. To model the random faults we assume that the temporal distribution of faults is unknown during a *fault burst*. A *fault burst* is a bounded time interval during which the real-time system is disturbed. The notion of *burst* is tackled in [12] to fit the reality of transmission errors on CAN network. In these works a probabilistic error model is proposed where an error burst represents the gravity of transmission errors.

Moreover, we describe the concept of *error recovery strategy* which defines the scheduler behaviour after an error detection. The scheduler will correct the faulty task, and will eventually apply preventive corrections to potential faulty tasks.

We investigate task scheduling under the *fault burst* hypothesis and the chosen error recovery strategy and evaluate the fault tolerance of fixed-priority and preemptive schedulers [11, 7, 1]. The feasibility tests of a task set are based on the Worst Case Response Time (WCRT) computation [2, 1, 6, 19, 18]. WCRT is the difference between the release and the completion of task instance when the worst case scenario of task sequence happens. This WCRT must be less or equal than the task deadline to guarantee that a task is schedulable.

This paper is structured as follows. Section 2 describes the traditional *pseudo-periodic fault* model and our *fault burst* model. Section 3 describes *error recovery strategies* and *error recovery tactics*. Section 4 provides the schedulability analysis of the error recovery strategies introduced in section 2. Section 5 presents the results obtained by simulation, and finally section 6 presents our conclusions.

2 Temporal Distribution of Faults

A real time system can be disturbed by several types of phenomenon. These phenomena do not have the same temporal distribution of faults, and do not cause the same effects on real time systems. It is necessary to use fault models to take into account these phenomena in the validation process. This section focus on the fault features of system disturbances. We describe the *pseudo-periodic fault* model and the *fault burst* model. We present their main features and show their application domain.

2.1 Pseudo-Periodic Fault Model

The classical temporal distribution of faults is featured by the *pseudo-periodic fault* model. Pseudo-periodic faults represent a fault distribution such that two consecutive faults are separated by a known minimum time interval. We denote T_F this minimum time interval (Figure 1).

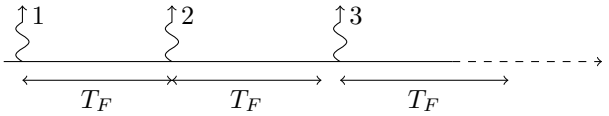


Figure 1. Pseudo-Periodic Fault Model

This temporal model deals with isolated faults, design faults and faults identified in electromagnetic compatibility. In the latter case, for example, faults due to the presence of a power supply near a computer unit can be modeled.

This assumption is lightly pessimistic as described on Figure 1. Faults 1 and 2 are separated by the exact minimum time interval T_F, but the fault 3 occurs later than T_F. So this assumption considers more fault occurrences than really possible by integrating them as periodic events.

In *pseudo-periodic fault* model, if the frequency of fault occurences increases, T_F decreases. However, if the fault occurences happen on a bounded time interval, this temporal distribution is not taken into account by the *pseudo-periodic fault* model. For theses temporal distributions, the minimum time interval between two faults can approach zero. The use of *pseudo-periodic fault* model is equivalent to a Δ_F equal to zero. The task set will be not schedulable. To integrate fault distribution that are over a bounded time interval, we suggest a complementary fault model.

2.2 Fault Burst Model

Our approach deals with a temporal fault distribution named *fault burst*. The *fault burst* model represents a time interval Δ_F during which the fault distribution is unknown. Each beginning of a *fault burst* is separated from the next one by a minimum time interval T_F. No fault can occur outside Δ_F.

In this paper the minimum time interval between two consecutive faults in a fault burst is unknown. Consequently, Δ_F is considered as a black box which causes faults on task sets during the *fault burst*. The *fault burst* model is depicted on Figure 2.

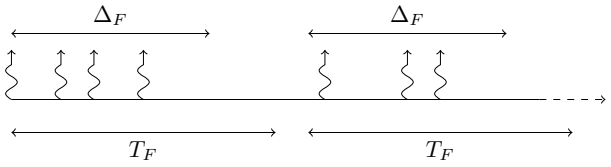


Figure 2. Fault Burst Model

This approach differs from the pseudo-periodic fault model. In the pseudo-periodic model, the focus is on the fault frequency. Our approach focuses on disturbance duration. If faults occur during a defined duration, the real-time system is disturbed randomly, and the pseudo-periodic model is not adapted to take into account this faulty configuration. So the fault burst model considers an unavailibility of the system. This model describes disturbances due to an electromagnetic field, a temporary lost of a resource due to a reset, etc...

3 Fault Tolerant Mechanisms

In this work, fault tolerance is obtained by error recovery at the task level. In addition to affecting eligible tasks to a processor, a scheduler defines the actions to undertake after error detection. Further, we detail these actions and call them *error recovery strategies*. To inform the scheduler of the current task behaviour we assume that detection mechanisms such as acceptance tests are available [15] or watchdog timers which interrupt the execution of the task once it has exceed its budget. Moreover, we assume that correction mechanisms can be applied to faulty tasks.

The recovery sequence which corrects a faulty task is composed of an error detection and an error correction, and is called *error recovery tactic*. It can be expressed as follows:

error recovery tactic = detection + correction

At task set level, we define the *error recovery strategy*, which defines the scheduler behaviour and the available *error recovery tactic* at the task level. It can be expressed as follows:

error recovery strategy = tactic + preemption treatment

3.1 Error Recovery Strategy

Error recovery strategies define the scheduler behaviour after an error detection. We distinguish the *simple strategy* and the *multiple strategy*.

Simple strategy means that the correction mechanism is applied only on the faulty task. If non detected errors exist in preempted tasks, the scheduler will wait for an effective detection and apply the correction to the corresponding task.

In a multiple strategy, if an error is detected in a task we assume that all preempted tasks are potentially faulty.

In this case, the scheduler applies an error correction to the current task, then interrupts all preempted tasks and executes an error correction on each of them.

3.2 Error Recovery Tactic

An error recovery tactic is composed of the detection and the correction of errors for a task. We detail each action below.

Error Detection We assume that detection mechanisms like acceptance tests are available in the real-time system and we focus on the time elapsed between the fault occurence and the effective error detection. This duration depends on the distribution of these detection mechanisms. The simplest model is a detection mechanism at the end of tasks. In this model, an error can only be detected just before the task completion, even if the error occurs just after the beginning of the task. A more complex model consists in splitting tasks in several parts, each part ending with a detection mechanism. The checkpointing mechanism is a particular error recovery tactic which includes task splitting and error correction [5, 4, 14, 3].

Error Correction This paper deals with time redundancies. We consider that error correction is based on re-execution of code. When an error is detected the scheduler can re-execute tasks either totally or partially. We assume that the effects of a fault on a task are eliminated by the application of an error correction method. The error recovery method is executed at the initial priority of the affected task. Preemptions by higher priority tasks can then occur. Faults can also affect the error correction method. In this case, the same error correction method will be applied on the faulty re-execution.

3.3 Focused Strategies

In this section, we describe several error recovery strategies. Schedulability analysis is provided in section 4 according the chosen strategy.

We consider that the *error recovery tactic* for the faulty task is an error detection that occurs at the end of the task (*End Detection*), and the error correction is made by a full re-execution of the task (*Full Re-execution*). The focused *error recovery strategies* define the behaviour of the scheduler towards the preempted tasks.

End Detection/Full Re-execution/Simple The error recovery tactic is applied only on a faulty task. This strategy is denoted *ED/FR/S*.

On Figure 3 we consider three tasks τ_1, τ_2 and τ_3 under a fixed priority scheduler where the WCET are 2, 3 and 2 time units respectively. Task τ_1 is the higher priority task and task τ_3 is the lower one. During execution an instance of τ_3 is preempted by an instance of τ_2. Before being preempted an error occurs in the task. This error will be detected at the end of the instance of τ_2. To complete τ_2 the re-execution duration is 3 time units.

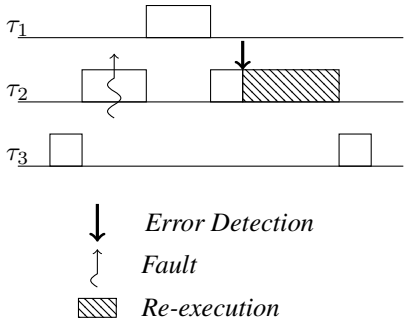


Figure 3. ED/FR/S Strategy

End Detection/Full Re-execution/Multiple The error recovery tactic is applied on both the faulty task and all preempted tasks. This strategy is noted *ED/FR/M*. This error recovery strategy is similar to the one used in [13].

This strategy is illustrated on Figure 4. The task set and its properties are the same as in Figure 3. When an error occurs during the instance of τ_1, instances of τ_2 and τ_3 are preempted. Therefore after the error detection re-execution of the three instances is made.

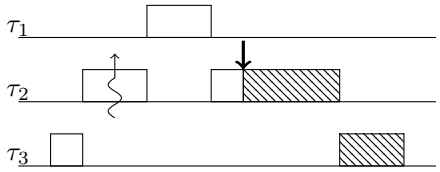


Figure 4. ED/FR/M Strategy

4 Schedulability Analysis

In the previous sections we suggested a new fault model named *fault burst* model and defined several error recovery strategies. The current section provides the schedulability analysis for task sets under *fault burst* according to the focused strategy. The section is structured as follows. Subsection 4.1 presents the computational model. Subsection 4.2 reminds the Worst Case Response Time computation under free fault hypothesis. Subsection 4.3 gives our assumptions to compute WCRT under *fault burst*. Subsection 4.4 describes the *fault burst* response time equation. Following subsections provide the schedulability analysis according the focused strategy.

4.1 Computational Model

Let Γ be a set of n tasks $\{\tau_1 ... \tau_n\}$. This task set must be scheduled by the system under free fault assumption. Each task $\tau_i \in \Gamma$ has a period T_i, a deadline D_i and a Worst Case Execution Time C_i. The task deadlines are less than equal to the task periods ($D_i \leq T_i$). We assume that tasks can either be periodic or sporadic. A sporadic task is such that a minimum time interval exists between two task instances. This time interval is considered as pseudo-periodic. The tasks are supposed to be independent, without precedence relations or shared resources.

The tasks are scheduled according to some fixed priority assignment algorithm (RM, DM) [11, 2]. The priorities are distinctly attributed for each task $\tau_i \in \Gamma$, so, for n tasks we consider n priority levels such that 1 is the highest priority level and n the lowest. We consider a preemptive algorithm and assume that overheads due to preemptions are negligible. As we only use error-recovery methods based on re-execution of code, we assume that this re-execution happens at the initial task priority. At any time the highest priority task from the set of runnable tasks is executed on the processor.

4.2 Free Fault Response Time

In this subsection we remind the computation of the free fault response time. This result is used in equation (5).

For a task τ_i the response time $\mathcal{R}_i$ is computed in the worst case scenario. This occurs when all other higher priority tasks τ_j are released simultaneously with τ_i. In this scenario, response time $\mathcal{R}_i$ is expressed as the sum of its WCET C_i and as an interference term I_i, due to preemption by higher priority tasks [6, 19]:

$$\mathcal{R}_i = C_i + I_i \tag{1}$$

This WCRT is computed under the preemptive scheduler hypothesis, so the interference term I_i is expressed as the maximum number of task instances that occur during the execution of τ_i (equation (2)). We denote $hp(i)$ the subset of tasks that have higher priority than τ_i. The ceiling operator $\lceil\ \rceil$ returns the smallest integer equal or greater than its argument.

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{\mathcal{R}_i}{T_j} \right\rceil C_j \tag{2}$$

As $\mathcal{R}_i$ appears on both sides of equation (1) it is solved by a fixed point algorithm. The relation is given by equation (3).

$$\mathcal{R}_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{\mathcal{R}_i^n}{T_j} \right\rceil C_j \tag{3}$$

The initial value of the iteration is $\mathcal{R}_i^0 = C_i + \sum_{j \in hp(i)} C_j$. We denote $\mathcal{R}_i^n$ the n^{th} value of $\mathcal{R}_i$. The algorithm ends as soon as $\mathcal{R}_i^{n+1} > D_i$, or earlier if $\mathcal{R}_i^{n+1} = \mathcal{R}_i^n$. In the first case the task is unschedulable and the task set is not feasible. The latter case means that the algorithm provides a WCRT $\mathcal{R}_i$ whose value is $\mathcal{R}_i^n$.

The inconvenient of this algorithm is that no value of WCRT is provided if the task is unschedulale. However, the advantage is that this algorithm is applicable if the task set is overloaded (in particular when *fault burst* occurs).

4.3 General Assumptions

The H1 hypothesis is a classical assumption in fault tolerance scheduling [13, 8]:

H1: The period between two *fault bursts* is greater than equal to the greatest task deadline of the task set (equation (4))

$$T_F \geq max_{i \in n}(D_i) \tag{4}$$

The hypothesis H1 is consistent with phenomena like *High-Intensity Radiated Fields* (HIRF) described in [16]. The exposure time of an aircraft fly-by or over ground HIRF transmitters (radar transmitters) does not exceed tenths of second, the elapsed time between two aircraft exposures is about few seconds [1].

From the H1 hypothesis we infer that each task instance is disturbed by at most one *fault burst*. We deduce the P1 property:

P1: There is at most one *fault burst* by WCRT computation.

If the focused task τ_i is unschedulable, the fixed point algorithm will stop before the next *fault burst* since $\mathcal{R}_i > D_i$. Otherwise, the task is schedulable, the WCRT computed is less than equal to the task deadline and the algorithm will end before the next *fault burst*. Therefore the P1 property guarantee the ending of the fixed point algorithm.

We assume that each equipment has an instantaneous update rate. No reset or restart are necessary to have the nominal behaviour of the equipment.Therefore, at the end of a *fault burst*, each equipment recovers its nominal behaviour. We can then deduce hypothesis H2. However latent errors in the current task and in the preempted tasks cannot be detected before an error detection.

H2: At the end of Δ_F, equipments affected by disturbances recover their nominal behaviour.

4.4 Fault Burst Response Time Equation

Response time $\mathcal{R}_i^{\Delta_F}$ of a task τ_i is expressed as the sum of its WCRT $\mathcal{R}_i$ under the no-failure hypothesis, the *fault burst* Δ_F, the interferences $I_i^{\Delta_F}$ due to preemptions by higher priority after the end of the *fault burst* tasks and an additional factor due to the error recovery strategy F_i (equation (5)) . This expression is similar to [3].

$$\mathcal{R}_i^{\Delta_F} = \mathcal{R}_i + \Delta_F + (I_i^{\Delta_F} + F_i) \tag{5}$$

Equation 5 is built as a sum of worst case scenarios. After the task release the WCRT $\mathcal{R}_i$ should be necessary for the task to complete. A *fault burst* occurs just before the effective completion of the task and causes an execution error. During the *fault burst*, we do not know the fault distribution and their consequences on the task set. In the worst case the fault distribution inside the *fault burst* is such that no task completes. We assume so that until the end of the *fault burst* the considered task is not completed. Higher priority tasks can have preempted the considered

[1]The worst case exposure to a typical rotating air search radar would be 15 sweeps (2 seconds between sweeps) each lasting less than 100 ms.

task. The elapsed timed since the task release is, therefore, equal to $\mathcal{R}_i + \Delta_F$.

Then we study the worst case scenario for the system recovery. We always consider that the running task after the end of the *fault burst* is erroneous. Moreover, we consider that the fault which causes the error occurs just after the beginning of a task execution. This execution can be either the normal execution or the re-execution. So we assume that after the end of *fault burst* the WCET of the running task is necessary to detect the error and to trigger the corrections according to the focused strategy. As the corrections are executed at the initial priority of the corresponding task, they can be preempted by the higher task priority. We defined the term $I_i^{\Delta_F}$ to integrate all the preemptions from the elapsed time $\mathcal{R}_i + \Delta_F$. Its expression is given by equation (6).

$$I_i^{\Delta_F} = \sum_{hp(i)} \left\lceil \frac{\mathcal{R}_i^{\Delta_F} - (\mathcal{R}_i + \Delta_F)}{T_j} \right\rceil C_j \qquad (6)$$

From the execution point of view error detection and error corrections are considered as additional time. This additional time can be computed independently since it depends only on the chosen strategy. We denote this recovery term F_i. The number of detections and corrections included in the term F_i varies according to the *simple strategy* and the *multiple strategy*.

Finally, from the *fault burst* end, we must consider the sum of $I_I^{\Delta_F} + F_i$ until the task completion. Therefore we obtain equation (5). To compute the WCRT of the task we apply the fixed point theorem described in 4.2.

4.5 Recovery Term F_1 for the Highest Priority Task

The computation of the term F_i for the highest priority task ($P = 1$) does not depend on the considered strategy. This task cannot be preempted by another task. We assume that a *fault burst* occurs just before the completion of the highest priority task. In the worst case scenario, no re-execution is successful during the *fault burst*, and the last re-execution begins just before the end of the *fault burst*. Therefore two additional re-executions are necessary to successfully complete the task. At the end of the first re-execution error detection occurs out of a *fault burst*. The term F_i is computed as twice of the value of WCET of the highest priority task (equation (7)).

$$F_1 = 2 \times C_1 \qquad (7)$$

Whatever the focused strategy, the WCRT $\mathcal{R}_1^{\Delta_F}$ is therefore expressed as the sum of three times the WCET and *fault burst* duration(equation (8)).

$$\mathcal{R}_1^{\Delta_F} = 3 \times C_1 + \Delta_F \qquad (8)$$

4.6 ED/FR/S Strategy

We consider that error detection happens at the end of task instance. The correction method is full re-execution. We apply *simple strategy*, so an error detection causes a full re-execution of the task without forcing a full re-execution of preempted tasks.

4.6.1 Computation of the Recovery Term F_i

Under fault burst hypothesis there is at most one instance of higher priority tasks that is potentially affected. When the *fault burst* is over the worst case occurs if the WCET of each task is necessary to detect another error. As a result, a full re-execution is required to eliminate errors. So, the time cost F_i is maximal for the considered task. Its value is equal to the sum between two WCET of higher priority tasks and the concerned task (equation (9)).

$$\forall (i > 1)\ F_i = 2 \times \sum_{hp(i)} C_j + 2 \times C_i \qquad (9)$$

Intuitively the factor 2 in the equation (9) corresponds to the application of the error recovery tactic for each higher priority task and concerned task. In the worst case, one detection and one correction are necessary to guarantee fault tolerance of the task set. Figure 5 describes a timeline for a task set. The WCRT $\mathcal{R}_3^{\Delta_F}$ is computed for the third task. In the worst case, after the *fault burst* ends, tasks τ_1, τ_2, τ_3 are faulty. The ED/FR/S strategy will be applied as follows. For each task, a WCET is necessary to detect the error. Then, each task is re-executed at its initial priority. This is the reason why an instance of τ_1 can preempt the re-execution of τ_2. The first part of the picture contains $\mathcal{R}_3$, the second part Δ_F and the third one represents $F_3 + I_3^{\Delta_F}$

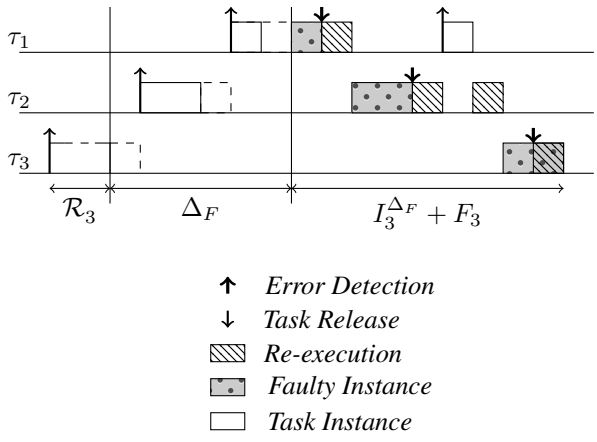


Figure 5. $\mathcal{R}_3^{\Delta_F}$ in ED/FR/S Strategy

4.6.2 Example

Table 1 shows an example of a three task set. The scheduler is Rate Monotonic, and the duration of *fault burst* Δ_F is 50 time units.

The *fault burst* impact is higher on lower priority tasks. In the worst case, the extra necessary time to complete the task τ_3 is equal to 540 time units whereas this extra time is equal to 70 for the task τ_1.

Task set $\Delta_F = 50$						
P	T	C	D	$\mathcal{R}$	F	$\mathcal{R}^{\Delta_F}$
1	300	10	300	10	20	80
2	500	50	500	60	120	240
3	800	150	800	210	420	750

Table 1. ED/FR/S Strategy: results

In our example, the task set is schedulable. We note that the WCRT $\mathcal{R}^{\Delta_F}$ is more than twice as high as the WCRT $\mathcal{R}$.

4.7 ED/FR/M Strategy

We consider an error detection happening at the end of the task execution. The correction method is full re-execution. We apply *multiple strategy* so that an error detection causes a full re-execution of the task and forces a full re-execution of the preempted tasks.

4.7.1 Computation of the Recovery Term F_i

Under the fault burst hypothesis there is at most one instance of higher priority tasks that is potentially affected. When the *fault burst* is over, *multiple strategy* is applied as soon as an error is detected. So, we consider that the worst case occurs if an error is detected after the longest WCET. The time cost of correction is maximum if all the higher priority tasks are fully re-executed. We deduce the corresponding F_i defined by equation (10).

$$\forall (i > 1)\ F_i = \sum_{j \in hp(i)} C_j + \max_{j \in hp(i)} C_j + C_i \qquad (10)$$

Intuively the equation (10) means that for the task subset composed by the focused task and its higher priority tasks, an error detection causes the re-execution of each task in the task subset.

Figure 6 describes a timeline for a task set. The WCRT $\mathcal{R}_3^{\Delta_F}$ is computed for the third task. When the *fault burst* ends task τ_1 is faulty. So, after the error detection, the task is re-executed. The scheduler applies an ED/FR/M strategy, and re-executes preventively all the preempted tasks. The preemption by the higher priority task is taken into account. The first part of the picture contains $\mathcal{R}_3$, the second part Δ_F and the third one represents $F_3 + I_3^{\Delta_F}$.

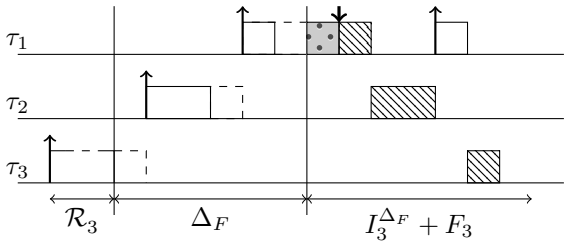


Figure 6. $\mathcal{R}_3^{\Delta_F}$ in ED/FR/M Strategy

4.7.2 Example

Table 2 uses the same task set as Table 1. We compute F and $\mathcal{R}^{\Delta_F}$ for each task. We can see that the impact of *fault burst* decreased on tasks τ_2 and τ_3. The task set is still schedulable.

Task set $\Delta_F = 50$						
P	T	C	D	$\mathcal{R}$	F	$\mathcal{R}^{\Delta_F}$
1	300	10	300	10	20	80
2	500	50	500	60	70	230
3	800	150	800	210	260	590

Table 2. ED/FR/M Strategy: results

The ED/FR/M strategy provides better results than ED/FR/S strategy. The WCRT $\mathcal{R}_3^{\Delta_F}$ is 33% lower than in the ED/FR/S strategy.

This scheduling analysis shows that under the *fault burst* hypothesis multiple strategy is generally more efficient than simple strategy. Indeed the worst case response time is less pessimistic since the time cost due to error detection and correction is reduced. In practice, we can note that the F_i computation under ED/FR/M strategy represents the case where the longest WCET is the higher task one. Therefore, we over evaluate the value of the F_i.

4.8 ED/FR/M Refined Strategy

Previously, we considered that the maximum value of F_i was the sum of the maximum time detection and maximum time correction. When the *fault burst* is over, task instances are executed following their priority (the highest task priority available uses the processor). So, if an error is detected in the current task instance, re-execution concerns only this task instance and lower piority preempted tasks.

4.8.1 Computation of the Recovery Term F_i

The refinement consists in finding the maximum value of the detection and the correction sequences according to the current task instance. As we do not know the distribution of faults inside the *fault burst*, we cannot determine which task instance will be executed at the end of the *fault burst*, so we compute each sequence. The worst case detection is still the WCET of the current task. The corresponding formula of F_i is described by equation (11).

$$\forall (i > 1)\ F_i = \max_{j \in hp(i)} \left(C_j + \sum_{k=i-1}^{k=j} C_k \right) + C_i \qquad (11)$$

Figure 7 describes a timeline for a task set. The WCRT $\mathcal{R}_3^{\Delta_F}$ is computed for the third task. The worst case is built as follows. There is no instance of the highest priority task at the end of the *fault burst*. That means that either an instance was completed or no instance was released in the *fault burst*. Task τ_2 has the longuest WCET. This task is faulty at the end of the *fault burst*. In the worst case a

WCET is necessary to detect the fault. The re-execution occurs and the scheduler re-executes the preempted task τ_3. The first part of the picture contains $\mathcal{R}_3$, the second part Δ_F and the third one represents $F_3 + I_3^{\Delta_F}$.

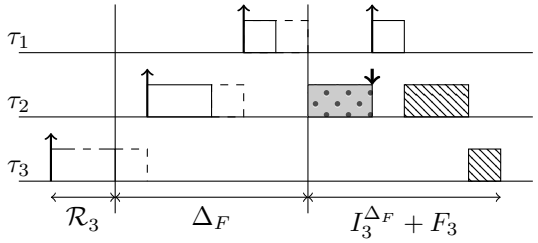


Figure 7. $\mathcal{R}_3^{\Delta_F}$ in ED/FR/M refined Strategy

4.8.2 Example

Table 3 refines the results of the previous Table 2 for task τ_3. We can notice that this refinement imply a computation time longer than in the example 4.7.2. Indeed the number of necessary steps to provide the value of $\mathcal{R}_i^{\Delta_F}$ is increased to obtain a more precise result.

Task set $\Delta_F = 50$						
P	T	C	D	$\mathcal{R}$	F	$\mathcal{R}^{\Delta_F}$
1	300	10	300	10	20	80
2	500	50	500	60	70	230
3	800	150	800	210	250	580

Table 3. ED/FR/M Refined Strategy: results

In fact, the worst case scenario happens when the higher priority task τ_2 fails. So, task τ_1 has no influence in the F_3 computation. The WCRT $\mathcal{R}_3^{\Delta_F}$ is better than previously.

5 Performance Evaluation

This section characterizes the applicability of the described strategies. We evaluate the fault tolerance under *fault burst* of schedulable task sets under no-failure hypothesis and we compare the error recovery strategies.

5.1 The Nature of the Experiments

For a given processor utilisation, 1000 task sets (10 tasks per task set) were randomly generated. All the task set are schedulable under the no-failure hypothesis. These task set features are similar to [10]. Deadlines were allowed to be equal to periods. The minimum and maximum values of processor utilisation are 30% and 95% accordingly. The Rate Monotonic algorithm is used to assign priorities for each task in a task set. The simulation is made as follows. The *fault burst* varies from 10% to 35% of the longuest task period. These *fault burst* values are consistent with[16, 17] guidance. For each *fault burst* value and each task set we applied the three strategies.

5.2 Simulation Results

Figure 8 presents the simulation results on 3-D curve for ED/FR/S strategy. Figure 9 presents the simulation results for ED/FR/M refined strategy. The X-axis represents the processor utilisation, the Y-axis the *fault burst* and the Z-axis the number of schedulable task sets. The figures both show that the more the *fault burst* is increasing, the more the number of schedulable task sets is decreasing. The same classical comment is made for the increase of processor utilisation.

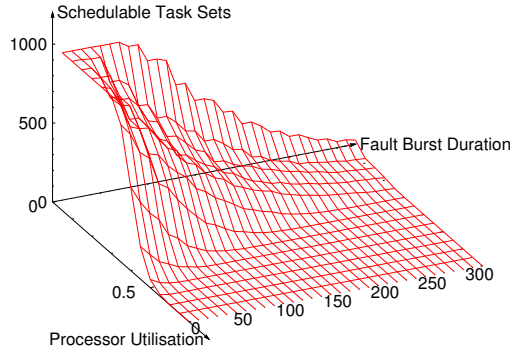


Figure 8. ED/FR/S Simulation

When *fault burst* is equal to 0, the case is equivalent to the pseudo periodic fault. In this case, the ED/FR/S strategy is able to validate some task sets until 55% of processor utilisation. The ED/FR/M refined strategy improves this result until 65% of processor utilisation. Now we consider the case where the processor utilisation is equal to 50%. This value is a sufficient condition to validate a task set when a pseudo-periodic fault occurs, according to [13]. In this case the ED/FR/S strategy validates some task sets until a *fault burst* equals 3%. The ED/FR/M refined strategy improves this result to a *fault burst* equal to 14%. The ED/FR/M refined strategy is more efficient than ED/FR/S strategy when *fault burst* occurs.

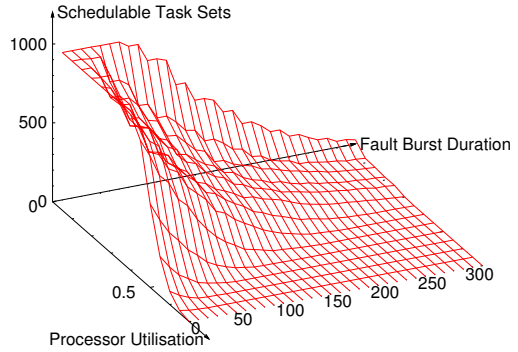


Figure 9. ED/FR/M Refined Simulation

These simulations show the capacity of our strategies

to guarantee the fault tolerance from the scheduling point of view. The possibility of the ED/FR/M refined strategy to schedule task sets with 50 % of processor utilisation under *fault burst* greater than 10 % is a good result for real-time system conception.

5.3 Comparison between Strategies

These cases show that in case of *fault bursts* the *multiple strategy* is more efficient than the *simple strategy*. The task sets validated by *simple strategy* are also validated by *multiple strategy*. For each couple of *fault burst* and processor utilisation, the number of schedulable task sets with the ED/FR/M refined strategy is equal or greater than the one with the ED/FR/S strategy.

Figure 10 shows the comparison between the previous strategies for a *fault burst* fixed to 10% of the longuest task period of the set. The X-axis represents the processor utilisation and the Y-axis the number of schedulable task sets. We can see that the two curves decrease continuously when the processor utilisation increases. The two strategies are equivalent up until the processor utilisation reaches 30%. From this value on, the ED/FR/M refined strategy becomes more efficient.

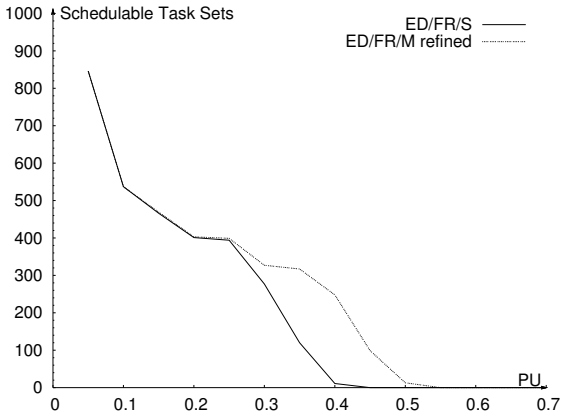
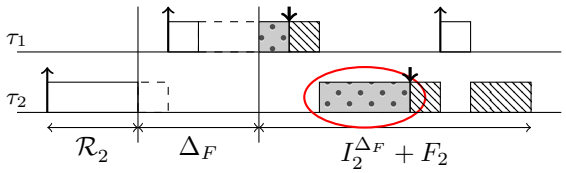


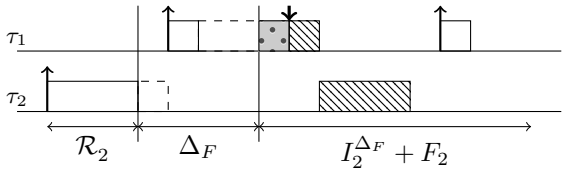
Figure 10. ED/FR/S and ED/FR/M Refined Strategies for a *fault burst* of 10%

Figure 11 describes qualitatively the difference between the *simple strategy* and the *multiple strategy*. The rounded faulty task τ_2 on the left subfigure does not exist on the right one. The principle of ED/FR/S strategy is n detections n corrections. Only the erroneous tasks are re-executed. The ED/FR/M strategies are based on 1 detection n corrections.

In practice, the preventive re-executions can concern no erroneous tasks. As the response time is based on the worst case, the preventive corrections improve the WCRT approximation by limiting the number of error detections. From the temporal point of view, we potentially increase the number of non necessary re-executions. However from the validation point of view, the approximation of WCRT is less pessimistic and the number of schedulable



(a) ED/FR/S Strategy



(b) ED/FR/M Strategy

Figure 11. Qualitative Explanation of ED/FR/M Strategy Efficiency

task sets increases. The medium case response time is degraded in favour of the worst case response time

6 Conclusions

In this paper we defined the *fault burst* model to take into account temporal fault distributions which disturb the real time system over a bounded time interval. This model can describe the unavailibility or the malfunctionning of a data sensor and the disturbances due to electromagnetic fields like *High-Intensified Radiated Fields*(HIRF). The *fault burst* model represents a temporal fault distribution that is complementary to the *pseudo-periodic fault* model.

Furthermore, we dealt with fault tolerant mechanisms. We introduced the error recovery strategy that defines the scheduler behaviour when an error is detected on a running task. We defined two error recovery strategies and assumed that the error detection can occur at the end of the task. The correction is made by a full re-execution of the erroneous task. The difference between the two strategies is the treatment of the preempted tasks.

We provided the schedulability analysis under the *fault burst* model for each described strategy. We defined the Worst Case Response Time equation and the worst case scenarios. To evaluate the efficiency of the strategies we defined a test protocol and made several simulations. We demonstrated that the ED/FR/M strategy is more efficient than the ED/FR/S strategy when the *fault burst* and the processor utilisation increases.

In the future, we need to investigate formal proofs for the correctness of our equations. In addition the improvement of previous results can be obtained following two guidelines. First, we can refine fault models to take into account material failures. The main idea is to approach to a model that includes safety analysis as well. Precision of fault recovery strategies will be improved by this refinement. Second, response time analysis enjoys from a lower algorithmicc complexity. However worst case response

times are quite pessimistic. We will investigate other approaches to be more precise.

References

[1] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[2] N. Audsley, A. Burns, M. Richardson, and A. Wellings. Hard Real-Time Scheduling: The Deadline Monotonic Approach. In *IFAC/IFIP Workshop,Atlanta, Georgia*, pages 127–132, May 1991.

[3] A. Burns, R. Davis, and S. Punnekkat. Feasibility Analysis of Fault-Tolerant Real-Time Task Sets. In *Proceedings of the Eighth Euromicro Workshop Real-Time Systems*, pages 29–33, 1996.

[4] E. Gelenbe. On the optimum checkpoint interval. *Journal of the ACM (JACM)*, 26(2):259–270, 1979.

[5] D. Jasper. A discussion of checkpoint restart. *Software Age*, 3(10):9–14, 1969.

[6] M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal*, 29(5):390–395, 1986.

[7] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization adverage case behavior. *Real Time Systems Symposium, 1989., Proceedings.*, pages 166–171, 1989.

[8] A. Liestman and R. Campbell. A Fault-Tolerant Scheduling Problem. *IEEE Transactions on Software Engineering*, 12(11):1089–1095, 1986.

[9] G. Lima and A. Burns. An effective schedulability analysis for fault-tolerant hardreal-time systems. In *Real-Time Systems, 13th Euromicro Conference on, 2001.*, pages 209–216, 2001.

[10] G. Lima and A. Burns. An optimal fixed-priority assignment algorithm for supporting fault-tolerant hard real-time systems. *IEEE Transactions on computers*, 52(10):1332–1346, 2003.

[11] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[12] N. Navet, Y.-Q. Song, and F. Simonot. Worst-case deadline failure probability in real-time applications distributed over controller area network. *Journal of Systems Architecture*, 46(7):607–617, 2000.

[13] M. Pandya and M. Malek. Minimum achievable utilization for fault-tolerant processing ofperiodic tasks. *IEEE Transactions on Computers*, 47(10):1102–1112, 1998.

[14] S. Punnekkat. *Schedulability analysis for fault tolerant real-time systems*. PhD thesis, University of York - Department of Computer Science, 1997.

[15] B. Randell. System structure for software fault tolerance. *ACM SIGPLAN Notices*, 10(6):437–449, 1975.

[16] RTCA and EUROCAE. Guide to Certification Of Aircraft in a High Intensity Radiated Field (HIRF) Environment. Technical Report ED 107 - ARP 5583, March 2001.

[17] RTCA and EUROCAE. Environmental Conditions and Testprocedures for Airborne Equipment. Technical Report ED 14E - DO 160E, March 2005.

[18] M. Spuri. Analysis of Deadline Scheduled Real-Time Systems. Technical Report RR-2772, 1996.

[19] K. Tindell. An extendible approach for analyzing fixed priority hard real-time tasks. Technical Report YCS189, 1992.