Task-Level Probabilistic Scheduling Guarantees for Dependable Real-Time Systems - A Designer Centric Approach

Hüseyin Aysan, Radu Dobrin, and Sasikumar Punnekkat
Mälardalen Real-Time Research Centre, Mälardalen University
Västerås, Sweden
{*huseyin.aysan, radu.dobrin, sasikumar.punnekkat*}*@mdh.se*

***Abstract*—Dependable real-time systems typically consist of tasks of mixed-criticality levels with associated fault tolerance (FT) requirements and scheduling them in a fault-tolerant manner to efficiently satisfy these requirements is a challenging problem. From the designers' perspective, the most natural way to specify the task criticalities is by expressing the reliability requirements at task level, without having to deal with low level decisions, such as deciding on which FT method to use, where in the system to implement the FT and the amount of resources to be dedicated to the FT mechanism. Hence, it is extremely important to devise methods for translating the high-level requirement specifications for each task into the low-level scheduling decisions needed for the FT mechanism to function efficiently and correctly.**

In this paper, we focus achieving FT by redundancy in the temporal domain, as it is the commonly preferred method in embedded applications to recover from transient and intermittent errors, mainly due to its relatively low cost and ease of implementation. We propose a method which allows the system designer to specify task-level reliability requirements and provides a priori probabilistic scheduling guarantees for real-time tasks with mixed-criticality levels in the context of preemptive fixed-priority scheduling. We illustrate the method on a running example.

***Keywords*-Fault tolerance; Time redundancy; Schedulability analysis; Real-time systems.**

I. Introduction

Embedded systems are deployed ubiquitously in applications that interact and control our lives including in safety critical real-time applications. These applications, typically have to satisfy complex requirements, mapped to task attributes and further used by the underlying scheduler in the scheduling decision. Additionally, these systems are often characterized by high dependability requirements, where fault tolerance techniques play a crucial role towards achieving them. Traditionally, such systems found in, e.g., aerospace, avionics or nuclear domains, typically employed the preemptive fixed priority scheduling (FPS) paradigm and were built with high replication and redundancy, with the objective to maintain the properties of correctness and timeliness even under error occurrences. However, in majority of modern embedded applications, due to space, weight and cost considerations, it may not be feasible to provide space redundancy. Such systems often have to exploit time redundancy techniques.

In order to provide real-time guarantees for fault tolerant systems, it is necessary to take into account the fault hypothesis, as no system can cope with an arbitrary number of faults over a bounded time interval [1]. Previous works assumed a worst case error distribution, e.g., in [2], single faults with a minimum inter-arrival time of largest period in the task set can be recovered if the processor utilization is less than or equal to 0.5 under rate monotonic (RM) scheduling, or schedulability-centric approaches based on fault assumptions modeled as stochastic events [1].

Modern dependable embedded systems typically consist of a mix of hard and soft tasks with varying criticality levels as well as associated FT requirements, hence usage of time redundancy should be prioritized according to the criticality levels of tasks to ensure efficient resource usage. One way of handling multiple criticality levels is by direct assignment to resources per task based on a pre-defined maximum number of feasible recovery attempts [3], that would deterministically ensure schedulability. However, from the designers' perspective, the most natural way to specify the task criticalities is by expressing reliability requirements at task level, preferably without having to deal with low level decisions, such as specifying the maximum number of feasible recovery attempts.

In this paper, we introduce a designer-centric approach to enable efficient fault tolerance in preemptive fixed priority scheduling. We build on the approach introduced in [1] and provide probabilistic guarantees for individual tasks to meet the designers' reliability requirements. We propose a scheduling technique to enable selective fault-tolerance for tasks with various FT requirements, assuming a probabilistic error distribution. We provide a schedulability analysis that takes these FT requirements as input and performs the schedulability test using the proposed scheduling technique.

The remainder of the paper is organized as follows. In the next section, we present the related work. In Section III, we describe the system characteristics, real-time task model and error model assumed in this paper, and FT strategy used in our analysis. Section IV describes our proposed methodology, illustrated by an example. We conclude the paper in Section V.

II. Related work

Redundancy in the physical, temporal, information and analytical domains is the key for achieving fault tolerance and due to the wealth of research in this domain, a rich set of techniques has been successfully used in many critical applications. Similarly, due to the inherent criticality of real-time systems, several researchers have been trying to incorporate fault tolerance into various real-time scheduling paradigms. In [4] and [5], different approaches were presented to schedule primary and alternate versions of tasks to provide fault tolerance. Krishna and Shin [6] used a dynamic programming algorithm to embed backup schedules into the primary schedule. Ramos-Thuel and Strosnider [7] used the Transient Server approach to handle transient errors and investigated the spare capacity to be given to the server at each priority level. They also studied the effect of task shedding to the maximum server capacity where task criticality is used for deciding which task to shed. In [8], [9], the authors presented a method for guaranteeing that the real-time tasks will meet the deadlines under transient faults, by resorting to reserving sufficient slack in queue-based schedules. Pandya and Malek [2] showed that single faults with a minimum inter-arrival time of largest period in the task set can be recovered if the processor utilization is less than or equal to 0.5 under rate monotonic (RM) scheduling. Burns et. al. [10], [11], [1] provided exact schedulability tests for fault tolerant task sets under specified failure hypothesis. These analysis are applicable for FPS schemes, and, being exact analysis, can guarantee task sets with even higher utilization than guaranteed by Pandya and Malek's test [2]. Lima and Burns [12], [13] extended this analysis in case of multiple faults, as well as for the case of increasing the priority of a critical task's alternate upon fault occurrences, and in [14] an upper bound for fault-tolerance in real-time systems based on slack redistribution is presented. We have proposed a work to maximize the fault tolerant capability of Fixed Priority Schedules [15], assuming each task instance is hit by an error and subsequently adapted this approach to Controller Area Network (CAN) in [3]. While the above works have advanced the field of fault tolerant scheduling within specified contexts, each one has some of the shortcomings such as lacking to provide probabilistic scheduling guarantees, restrictive task and fault models, non-consideration of multiple task criticality levels, high computational requirements of complex on-line mechanisms, and scheduler modifications which may be unacceptable from an industrial perspective.

III. System and dependability model

A. Real-time task model

We assume a periodic (or a sproadic) task set, $\Gamma = \{\tau_1, \tau_2, .., \tau_n\}$, scheduled by any preemptive fixed-priority scheduling paradigm where each task represents a real-time thread of execution. Each task τ_i has a period (or a minimum inter-arrival time) T_i, a known worst-case execution time (WCET) C_i, a deadline D_i and a priority P_i. We assume a single processor platform and that the tasks have deadlines equal to or less than their periods.

A task set Γ consists of critical and non-critical tasks where the criticality of a task indicates the severity of the consequences caused by its failure and corresponds to the amount of resources allocated for error recovery in terms of guaranteed re-transmissions. Each critical task τ_i has an alternate task with a worst case execution time F_i and a deadline equal to the original deadline D_i. This alternate can typically be a re-execution of the same task, a recovery block, an exception handler or an alternate with imprecise computations. We denote the subset of critical tasks by Γ_c and the subset of non-critical tasks by Γ_{nc} , so that $\Gamma = \Gamma_c \cup \Gamma_{nc}$.

B. Error model

Safety-critical embedded systems typically work in harsh environments where they can experience frequent transient errors due to several causes such as power supply jitter, network noise and radiation. As per the published statistics, the ratio between the frequencies of transient and permanent faults is found to vary from 4 to 1000 [16]. We follow Laprie's dependability concepts [17], [18] and assume that systems are exposed to these faults with probabilities depending on the characteristics of the systems and the environments that they are operating in.

The probability of error occurrence can be calculated by using the Poisson probability distribution as described by Burns et al. [1]. Poisson distribution is a discrete probability distribution used for finding the probability of a number of events occurred in a fixed time period, assuming that the events occur at a constant rate and their occurrences are independent. In our case, the events are error occurrences, hence the error occurrence rate for transient errors is assumed to be constant. This rate depends on the type of environment (together with the processor characteristics/circuitry) which gives the λ value for that environment, that is the expected number of events in a unit time. The common values for λ range from 10^2 errors per hour in aggressive environments to 10^{-2} errors per hour in lab conditions as presented by Ferreira et al. [19] and Rufino et al. [20].

The probability of n events during a time period of t is calculated as shown below.

$$Pr_n(t) = \frac{e^{-\lambda t}(\lambda t)^n}{n!}$$

If we assume that the event is an error, then the probability of no error during the lifetime or mission time (L) of the system is given by

$$Pr_{no\_error}(t) = e^{-\lambda L}$$

Thus, the probability of at least one error during L is

$$Pr_{at\_least\_one\_error}(t) = 1 - e^{-\lambda L}$$

Lifetime or a mission time of a system can vary largely depending on the domain, typically ranging from 1 hour for a plane to take a short trip to 15 years for a satellite to complete its lifetime.

In this paper, we are interested in the probabilities of the tasks meeting their deadlines for different error rate assumptions that define their criticality levels as described in the next subsection.

C. *Dependability requirements specification*

We assume that the system designers define the criticality levels for each task in terms of failure probability (or reliability) per hour. These reliability figures will then be used to derive the error inter-arrival time thresholds, T_{E_i}, for each task τ_i. These thresholds determine the maximum number of permitted error recovery actions. If the actual error inter-arrival times are greater than or equal to these thresholds then the specified reliability requirements are guaranteed to be satisfied, i.e. deadlines will be met even under error occurrences.

In [1], a single T_E value is valid for the whole task set (based on a single level of criticality), which means that the reliability requirement is specified for the whole system. They assumed that if the actual shortest interval between two errors in a mission time W is less than T_E, then the task set is unschedulable and then showed that the probability of unschedulability $Pr(U)$ is equal to $Pr(W < T_E)$. In the next section, we show how we use the specified reliability requirements in providing probabilistic guarantees for mixed-criticality task sets.

D. *FT strategy*

Our primary concern is providing probabilistic schedulability guarantees to all critical tasks where we employ time redundancy for error recovery. The error coverage achieved by this approach is shown in Figure 1. The basic assumption here is that the effects of a large variety of transient and intermittent hardware faults can effectively be tolerated by a simple re-execution of the affected task whilst the effects of software design faults could be tolerated by executing an alternate action such as recovery blocks or exception handlers. Both of these situations could be considered as execution of another task (either the primary itself or an alternate) with a specified computation time requirement.

We assume that each task failure is detected before the completion of the failed task instance. Although somewhat pessimistic, this assumption is realistic since in many implementations, errors are detected by acceptance tests which are executed at the end of task execution or by watchdog timers that interrupt the task once it has exhausted its budgeted worst case execution time. In case of tasks communicating via shared resources, we assume that an acceptance test is executed before passing an output value to another task to avoid error propagations and subsequent domino effects.

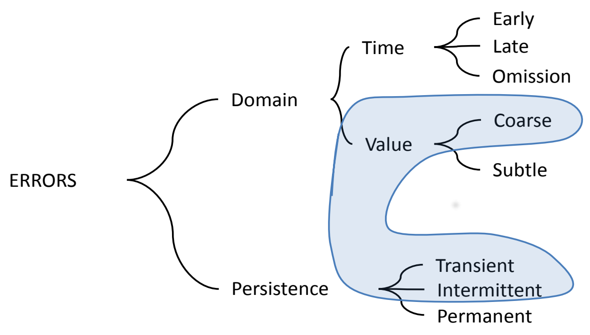

Figure 1. Coverage of error types achieved by time redundancy

In order to provide schedulability guarantees for tasks with different reliability requirements (with different T_{E_i} values), we assume that the scheduler prevents re-execution of task τi in case two errors occur with a separation shorter than the error inter-arrival time threshold T_{E_i} for that task. One way to prevent these erroneous re-executions would be to maintain a timer for each task that measures the time between the last two errors affecting that specific task. Such a timer could be used to prevent re-execution when that time is too short. Obviously, for non-critical tasks, no recovery is performed in case of failures of any task instances.

IV. Methodology

A. *Overview*

Our ultimate goal is to provide a schedulability analysis for the mentioned selective FT scheduling technique for scheduling the tasks with mixed reliability requirements, ranging from non-critical task, failure of which does not adversely affect the systems' correct and dependable functioning, to highly critical task, where one or more times of recovery actions might be necessary to perform in order to ensure dependability. The methodology consists of two steps:

1) The first step is to determine the minimum error inter-arrival time T_{E_i} for each task $\tau_i \in \Gamma_c$ using the reliability figures given by the system designers.
2) The final step is the schedulability test using the task attributes together with the T_{E_i} values.

B. *Proposed approach*

We use a simple example throughout the description of our approach. Let our task set consists of 4 tasks, as shown in Table I where columns P, T, C, D, F represent the priority, period, worst case execution time, deadline and the worst case recovery time respectively and the time unit is *milliseconds*. Priorities are ordered from 1 to 4 where 4 is the lowest priority. Note that Task B is a non-critical task, and hence it does not have a worst case recovery time.

Task	P	T	C	D	F
A	1	100	10	100	10
B	2	175	20	117	-
C	3	200	15	200	15
D	4	300	20	300	20

Table I
EXAMPLE TASK SET

1) Derivation of T_{E_i} values: By using the Poisson probability distribution, in [1], Burns et al. show the upper and lower bounds for $Pr(W < T_E)$ by the following theorems:

Theorem 1: If $L/(2T_E)$ is a positive integer then

$$Pr(W < T_E) < 1 + [e^{-\lambda T_E}(1 + \lambda T_E)]^{\frac{L}{T_E}+1}$$
$$-2[e^{-2\lambda T_E}(1 + 2\lambda T_E)]^{\frac{L}{2T_E}}$$

Theorem 2: If $L/(2T_E)$ is a positive integer then

$$Pr(W < T_E) > 1 - [e^{-\lambda T_E}(1 + \lambda T_E)]^{\frac{L}{T_E}}$$

Burns et al. [1] also derived the following two useful approximations for the upper and lower bounds:

Corollary 1: An approximation for the upper bound on $Pr(W < T_E)$ is given by Theorem 1 is $\frac{3}{2}\lambda^2 LT_E$ provided that λT_E, $\lambda^2 LT_E$ are small and $L >> T_E$.

Corollary 2: An approximation for the lower bound on $Pr(W < T_E)$ is given by Theorem 2 is $\frac{1}{2}\lambda^2 LT_E$ provided that λT_E, $\lambda^2 LT_E$ are small.

In this paper, we are going to use the approximation for the upper bound to calculate the T_{E_i} values for each task $\tau_i \in \Gamma_c$ from the reliability requirements.

Let us assume that the λ value for our environment is 10^{-2} and the lifetime L of our system is 1 hour. The probability of any fault happening during L is calculated as approximately 10^{-2} by using the Poisson probability distribution. This would mean that the parts of the system where no fault-tolerance is implemented would fail with a probability of 10^{-2}, i.e., any non-critical task will have a reliability of approximately 0.99. Let us assume that we have the reliability requirements shown in Table II for the critical tasks in our task set (the numbers shown in the table are selected for presentation purposes only).

Then we can use the Corollary 1 and specify $1 - R(i)$ as the upper bound for the error probability of each task to calculate the T_{E_i} values, where $R(i)$ is the reliability requirement of task τ_i. The calculated values are shown in Table III (in *milliseconds*).

Task	Reliability requirement
A	1 - 1 x 10^{-8}
C	1 - 1.25 x 10^{-9}
D	1 - 5.85 x 10^{-9}

Table II
RELIABILITY REQUIREMENTS FOR THE CRITICAL TASKS

Task	T_E
A	240
C	30
D	140

Table III
COMPUTED MINIMUM ERROR INTER-ARRIVAL TIMES FOR CRITICAL TASKS

2) Schedulability test: The worst-case response time R_i for each task τ_i is computed by using the following equation assuming that there are no task failures and no recovery attempts [21]:

$$R_i = C_i + B_i + \sum_{j \in hp(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \qquad (1)$$

where $hp(\tau_i)$ is the set of higher priority tasks than task τ_i, B_i is the maximum blocking time caused by the concurrency protocols used for accessing the shared resources.

The following recurrence relation is used for solving Equation 1:

$$r_i^{n+1} = C_i + B_i + \sum_{j \in hp(\tau_i)} \left\lceil \frac{r_i^n}{T_j} \right\rceil C_j \qquad (2)$$

where r_i^0 is given the initial value of $C_i + B_i$. r^n is a monotonically non-decreasing function of n and when r_i^{n+1} becomes equal to r_i^n then this value is the worst-case response time R_i for task τ_i. If the worst-case response time R_i becomes greater than the deadline D_i, then the task cannot be guaranteed to meet its deadline, and the task set is therefore unschedulable.

Table IV shows the worst case response times that are calculated by Equation 1. For the sake of simplicity, blocking times are assumed to be zero. As all the worst-case response times are less than the corresponding deadlines, this task set is schedulable under no-error scenarios.

If we assume an FT scheduler where the failed tasks are re-executed, then the execution of task τ_i will be affected by errors in task τ_i or any higher priority task. Based on this assumption, the worst-case response times are computed

Task	P	T	C	D	B	R
A	1	100	10	100	0	10
B	2	175	20	175	0	30
C	3	200	15	200	0	45
D	4	300	20	300	0	65

Table IV
RESPONSE TIMES - NO ERROR SCENARIO

[10] by using the following equation:

$$R_i = C_i + B_i + \sum_{j \in hp(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j + \left\lceil \frac{R_i}{T_E} \right\rceil \max_{k \in hep(\tau_i)} (F_k) \tag{3}$$

where F_k is the extra computation time needed by task τ_k, T_E is a known minimum inter-arrival time for errors and $hep(\tau_i)$ is the set of tasks with priority equal to or higher than the priority of task τ_i ($hep(\tau_i) = hp(\tau_i) + \tau i$). The last term calculates the worst-case interference arising from the recovery attempts.

This equation can again be solved by a recurrence relation as in the previous case. If all R_i values are less than or equal to the corresponding D_i values, then the task set is guaranteed to be scheduled under the condition that no two errors occur closer than the T_E value.

Table V shows the response times that are calculated by Equation 3 assuming a minimum interarrival of errors $T_E = 75ms$. The task set is schedulable under this error rate assumption, as all the worst-case response times are less than the corresponding deadlines.

Task	P	T	C	D	F	R ($T_E = 75$)
A	1	100	10	100	10	20
B	2	175	20	175	20	50
C	3	200	15	200	15	65
D	4	300	20	300	20	115

Table V
RESPONSE TIMES - SINGLE CRITICALITY LEVEL

In this paper, we extend the fault-tolerant scheduling paradigm presented [1] by allowing multiple task criticality levels as well as propose a new schedulability test adapted to our extension. In this test, the worst-case response times are computed by using the following equation (which can similarly be solved by forming a recurrence relation):

$$R_i = C_i + B_i + \sum_{j \in hp(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j + I(\tau_i, R_i) \tag{4}$$

In this equation, the term $I(\tau_i, R_i)$ is a function that computes the worst-case interference, arising from the recovery attempts, during the execution of task τ_i for R_i. Algorithm 1 shows the details of this function. Inputs to the algorithm are task τ_i, the response-time R_i in the current iteration of the recurrence relation, and T_{E_k} and F_k values for each $k \in hepc(\tau_i)$, where $hepc(\tau_i)$ is the set of *critical* tasks with priority equal to or higher than the priority of task τ_i. The algorithm first calculates the worst case number of interferences from the following equation:

Algorithm 1: I(τ_i, R_i)

```
input : $\tau_i$, $R_i$, $\forall k \in hepc(\tau_i)$ $T_{E_k}$, $F_k$
output: worst case interference for task $\tau_i$ in response time $R_i$
1  begin
2      $output \leftarrow 0$;
3      $n \leftarrow \left\lceil \frac{R_i}{min(T_{E_k})|k \in hepc(\tau_i)} \right\rceil$ ;
4      foreach $k \in hepc(\tau_i)$ do
5          put $k$ in Array $A$;
6      end
7      $j, k, l \leftarrow 0$;
8      sort $A$ by decreasing $F$;
9      while $j < n$ do
10         $output \leftarrow output + F_{k[A]}$;
11         $j \leftarrow j + 1$;
12         $l \leftarrow l + 1$;
13         if $l \geq \left\lceil \frac{R_i}{T_{E_{A[k]}}} \right\rceil$ then
14             $k \leftarrow k + 1$;
15             $l \leftarrow 0$;
16         end
17     end
18     return $output$;
19 end
```

$$n = \left\lceil \frac{R_i}{min(T_{E_k})|k \in hepc(\tau_i)} \right\rceil \tag{5}$$

Unlike in Equation 3, multiplying the worst case number of interferences with the largest F_k value would generate an unnecessary pessimism, since task τ_k that has the largest F_k may have a larger T_{E_k} than the T_E value used in Equation 5. In that case, the worst case number of interferences n in R_i can be greater than the worst case number of interferences by task τ_k in R_i which is calculated by $\left\lceil \frac{R_i}{T_{E_k}} \right\rceil$. Therefore, after $\left\lceil \frac{R_i}{T_{E_k}} \right\rceil$ additions of F_k, the interferences by task τ_l that has the next largest worst-case recovery time should be added either for $\left\lceil \frac{R_i}{T_{E_l}} \right\rceil$ times or $n - \left\lceil \frac{R_i}{T_{E_k}} \right\rceil$ times, whichever is smaller. This procedure is continued until n interference values are added, and finally, the algorithm outputs the accumulated worst case interference for task τ_i in response time R_i.

In our example, the resulting response times are shown in Table VI. We can see that all worst-case response times are less than the corresponding deadlines, and conclude that the task set is schedulable.

Task	P	T	T_E	C	D	F	R
A	1	100	240	10	100	10	20
B	2	175	-	20	175	-	40
C	3	200	30	15	200	15	90
D	4	300	140	20	300	20	175

Table VI
RESPONSE TIMES - MULTIPLE CRITICALITY LEVELS

In figures 2 to 5, we show the worst-case interference for the individual tasks and describe how they have been derived. In the worst case, Task A has one interference in a computational window of $R_A = 20$, and it is the interference by Task A itself, since n in Equation 5 is 1 ($\lceil\frac{20}{240}\rceil$) and Task A is the only task in $hepc(TaskA)$. Figure 2 shows the worst-case interference scenario for Task A.

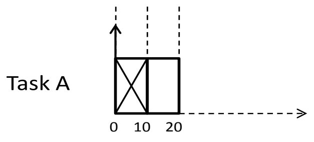

Figure 2. Worst-case interference for Task A

Task B has also one interference in a computational window of $R_B = 40$ in the worst case, and it is also the interference by Task A, since n in Equation 5 is 1 ($\lceil\frac{40}{240}\rceil$) and Task A is again the only task in $hepc(TaskB)$. Figure 3 shows the worst-case interference scenario for Task B.

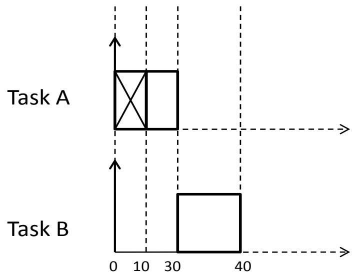

Figure 3. Worst-case interference for Task B

Task C can be interfered for maximum three times in a computational window of $R_C = 90$ ($\lceil\frac{90}{30}\rceil$). As the maximum of the F values for the tasks in $hepc(C)$ is Task C's worst case recovery time $F_C = 15$, the algorithm returns three times this value ($I(C, 90) = 45$). Figure 4 shows the worst-case interference scenario for Task C.

Task D has a worst case of 6 interferences in a computational window of $R_D = 175$ ($\lceil\frac{175}{30}\rceil$). The maximum of the F values for the tasks in $hepc(D)$ is Task D's worst

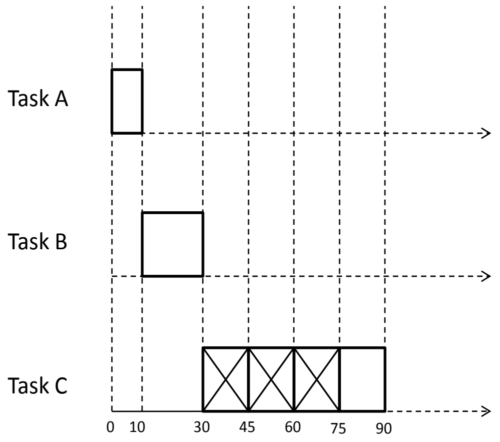

Figure 4. Worst-case interference for Task C

case recovery time $F_D = 20$, however, the recovery of Task D can interfere Task D's execution maximum 2 times, as $T_{E_D} = 140$ and $\lceil\frac{175}{140}\rceil = 2$. The other 4 interferences can come from Task C which has the next largest F value. The algorithm returns $I(D, 175) = 100$. Figure 5 shows the worst-case interference scenario for Task D.

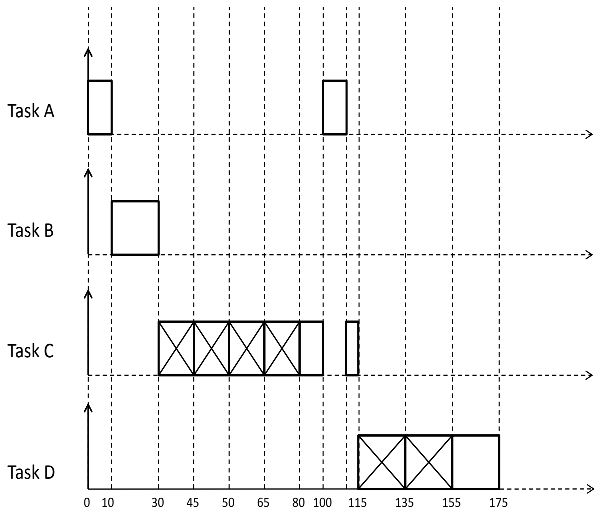

Figure 5. Worst-case interference for Task D

V. CONCLUSIONS

In this paper, we have presented a method to allow system designers to specify task-level reliability requirements and provide a priori probabilistic scheduling guarantees for real-time tasks with mixed-criticality levels in preemptive fixed-priority scheduling. We have focused on redundancy in the temporal domain for achieving FT as it is the often preferred method in many dependable embedded applications to recover from the most common transient and intermittent errors. The details of the method includes the translation of the designer specifications into low level decisions such as how many recovery attempts can feasibly be performed for each task in the task set, and the schedulability test to provide guarantees in meeting the specified requirements. We have illustrated the method on a running example.

Our ongoing research includes extending this approach to deal with burst error models, as well as simulation studies to evaluate the effectiveness of the proposed methodology.

Acknowledgment

This work was partially supported by the Swedish Foundation for Strategic Research via the strategic research centre PROGRESS. The authors would like to thank Antonio Bondavalli for fruitful discussions on *"dependability requirements specification"* that formed the basis of this work.

References

[1] A. Burns, S. Punnekkat, L. Strigini, and D. Wright, "Probabilistic scheduling guarantees for fault-tolerant real-time systems," *Dependable Computing for Critical Applications 7, 1999*, pp. 361–378, Nov 1999.

[2] M. Pandya and M. Malek, "Minimum achievable utilization for fault-tolerant processing of periodic tasks," *IEEE Transactions on Computers*, vol. 47, no. 10, 1998.

[3] H. Aysan, R. Dobrin, and S. Punnekkat, "Fault tolerant scheduling on control area network (can)," *IEEE International Workshop on Object/component/service-oriented Real-time Networked Ultra-dependable Systems*, 2010.

[4] A. L. Liestman and R. H. Campbell, "A Fault-Tolerant Scheduling Problem," *IEEE Transactions on Software Engineering*, vol. 12, no. 11, pp. 1089–95, November 1986.

[5] C.-C. Han, K. G. Shin, and J. Wu, "A fault-tolerant scheduling algorithm for real-time periodic tasks with possible software faults." *IEEE Trans. Computers*, vol. 52, no. 3, pp. 362–372, 2003.

[6] C. Krishna and K. Shin, "On scheduling tasks with a quick recovery from failure," *IEEE Transactions on Computers*, vol. 35, no. 5, pp. 448–455, May 1986.

[7] S. Ramos-Thuel and J. Strosnider, "The transient server approach to scheduling time-critical recovery operations," in *IEEE Real-Time Systems Symposium*, December 4-6 1991, pp. 286–295.

[8] S. Ghosh, R. Melhem, and D. Mosse, "Enhancing real-time schedules to tolerate transient faults," *IEEE Real-Time Systems Symposium*, 1995.

[9] F. Liberato, R. G. Melhem, and D. Mosse, "Tolerance to multiple transient faults for aperiodic tasks in hard real-time systems," *IEEE Transactions on Computers*, vol. 49, no. 9, pp. 906–914, 2000.

[10] A. Burns, R. I. Davis, and S. Punnekkat, "Feasibility analysis of fault-tolerant real-time task sets," *Euromicro Real-Time Systems Workshop*, 1996.

[11] S. Punnekkat, A. Burns, and R. I. Davis, "Analysis of checkpointing for real-time systems." *Real-Time Systems*, vol. 20, no. 1, pp. 83–102, 2001.

[12] G. Lima and A. Burns, "An optimal fixed-priority assignment algorithm for supporting fault-tolerant hard real-time systems," *IEEE Transactions on Computers*, vol. 52, no. 10, pp. 1332–1346, October 2003.

[13] G. Lima and A.Burns, "Scheduling fixed-priority hard real-time tasks in the presence of faults," *Lecture Notes in Computer Science*, pp. 154–173, 2005.

[14] R. M. Santos, J. Santos, and J. D. Orozco, "A least upper bound on the fault tolerance of real-time systems," *J. Syst. Softw.*, vol. 78, no. 1, pp. 47–55, 2005.

[15] R. Dobrin, H. Aysan, and S. Punnekkat, "Maximizing the fault tolerance capability of fixed priority schedules," *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2008.

[16] M. Pizza, L. Strigini, A. Bondavalli, and F. D. Giandomenico, "Optimal discrimination between transient and permanent faults," *3rd IEEE International Symposium on High-Assurance Systems Engineering*, pp. 214–223, 1998.

[17] J.-C. Laprie, "Dependable computing and fault-tolerance: Concepts and terminology," *International Symposium on Fault-Tolerant Computing, ' Highlights from Twenty-Five Years'.*, 1995.

[18] A. Avizienis, J. Laprie, and B. Randell, "Fundamental concepts of dependability," *Research Report N01145, LAAS-CNRS*, 2001.

[19] J. Ferreira, "An experiment to assess bit error rate in can," *3rd International Workshop of Real-Time Networks*, pp. 15–18, 2004.

[20] J. Rufino, P. Verissimo, G. Arroz, C. Almeida, and L. Rodrigues, "Fault-tolerant broadcasts in can," *Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing, 1998. Digest of Papers.*, pp. 150–159, 1998.

[21] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal - British Computer Society*, vol. 29, no. 5, pp. 390–395, October 1986.