Resource Augmentation for Fault-Tolerance Feasibility of Real-time Tasks under Error Bursts[*]

Abhilash Thekkilakattil, Radu Dobrin, Sasikumar Punnekkat and Huseyin Aysan
Mälardalen Real-Time Research Center, Mälardalen University, Sweden
{abhilash.thekkilakattil, radu.dobrin, sasikumar.punnekkat, huseyin.aysan}@mdh.se

ABSTRACT

Dependability is a vital system requirement, particularly in safety critical and mission critical real-time systems, due to the potentially catastrophic consequences of failures. In most critical applications different fault tolerance mechanisms using redundancy are employed to prevent possible failures. In the case of real-time systems the system designer must ensure that the task set is feasible even under faults, which we refer to as 'fault tolerance feasibility'. Due to cost considerations, often temporal redundancy has been prevalently used to meet this objective.

In this paper we focus on guaranteeing fault-tolerance feasibility under error bursts on uni-processor systems by the usage of resource augmentation, specifically through processor speed-up. Firstly, we derive a processor demand bound based sufficient condition for a set of real-time tasks to be fault tolerance feasible under an assumption that no more than one error burst occurs during the hyper-period of the task set. Subsequently, we derive the necessary resource augmentation bounds (i.e., the processor speed-up), that guarantees the fault tolerance feasibility, if the sufficient test fails. Finally, we prove that, if the error burst length is no more than half the shortest relative deadline of the task set, the minimum processor speed-up required to guarantee fault tolerance feasibility is upper-bounded by 6.

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special -Purpose and Application -Based Systems—*Real-time and embedded systems*; C.4 [**Computer Systems Organization**]: Performance of Systems— *Fault-tolerance, Reliability, Availability, and Serviceability*

General Terms

Theory, Reliability

[*]This work was partially supported by the Swedish Research Council project CONTESSE (2010-4276).

Keywords

Real-time Scheduling, Fault Tolerance Feasibility, Resource Augmentation, Error Bursts

1. INTRODUCTION

Mission and safety critical real-time systems typically have to perform a number of functionalities of mixed criticality levels, ranging from ultra-critical to non-critical. In addition to the temporal correctness, these systems need to provide for a high degree of reliability, due to the catastrophic consequences a failure may lead to. The reliability of the system is typically achieved by the use of fault tolerance mechanisms that aim to prevent potential system failures while guaranteeing the real-time constraints. Consequently, any reasoning about the correctness of the system needs to take into account an appropriate fault model, as well as the overheads associated with the employed fault-tolerance mechanisms.

In a real-time system, the events occurring in the system are typically mapped to a set of real-time tasks, with the requirement that the task executions must complete by their respective deadlines. Additionally, reliability constraints require the use of an appropriate fault tolerance strategy, most commonly in the form of temporal redundancy, which involves the re-execution of the original task or the execution of an alternate task, before its predefined deadline [2][3][5][22]. In this context, the original task execution is typically referred to as the primary and the re-executions upon faults are referred to as alternates. If the fault occurrence persists during the execution of the recovery attempts, alternates are executed until one successful execution is achieved. In order to reason about the temporal correctness of the system, safe upper-bounds on the task execution times, i.e., the Worst Case Execution Times (WCET), derived using suitable techniques [20], are required. Additionally, the use of temporal redundancy requires that the schedulability analysis techniques consider the transient overloads generated by the execution of alternates in order to guarantee the overall system schedulability under fault occurrences.

Many safety critical and mission critical real-time systems employ a preemptive Fixed Priority Scheduler (FPS) due to its simple scheduling mechanism that enables an easy implementation, even on operating systems that do not provide explicit support for timing constraints [10]. However, in the general case, preemptive FPS may not be able to guarantee schedulability of the task sets if the total task utilization is greater than 69% [21]. Dynamic priority schemes, on the other hand, e.g., Earliest Deadline First scheduling [13] [21], have the ability to utilize the processor more effectively, and are being promoted in the academia [10], as well as in many commercial operating systems [11]. Hence a real-time systems designer can choose from a wide variety of scheduling schemes while designing the system. During the design stage of the system, the choice of a scheduler is influenced by the task attributes. In most cases, the task attributes are derived from the physical characteristics of the environment that the real-time system is controlling and are thus unchangeable. Hence an important question is which scheduling scheme will yield a feasible schedule that can tolerate faults under a specified fault hypothesis. However, even before answering this question it is more appropriate to ask whether it is possible to determine *if there exists* a scheduling scheme that can tolerate faults under the specified fault hypothesis for the given task set.

A real-time fault tolerant scheduler that employs the temporal redundancy approach needs to ensure the execution of either a primary or an alternate, of all critical tasks, before their respective deadlines under the specified fault hypothesis. The existence of a real-time scheduling algorithm that can tolerate faults can be demonstrated by showing that, in any time interval, the total processor demand requested by the task primaries and the alternates that results in a worst case scenario is no greater than the size of the interval [8] [3]. Hence, for a real-time scheduling scheme to be fault tolerant, there must exist sufficient slack in the schedule for the execution of the task primaries and the alternates. EDF is known to be an optimal uniprocessor scheduling algorithm, i.e., if it is possible to schedule the original task executions together with the required alternates without causing a deadline miss, then EDF will also schedule them.

A real-time task set is said to be Fault Tolerance feasible (FT-feasible) if there exists a schedule that is capable of tolerating worst case fault occurrences under a specified fault hypothesis [3]. If the task set is not FT-feasible then there exists no sufficient slack in the schedule which can be utilized by the fault tolerance scheduling algorithm in order to recover from faults. In this case, the use of a faster processor can compensate for the slack deficit, thus enabling feasible recovery from faults. Thus the system designer can select a faster processor that guarantees the fault tolerance feasibility, but at the same time may be interested in choosing the one with the lowest speed among those eligible due to cost factors. However, the system designer has to first know if FT-feasibility can be achieved by speeding up the processor by a practicable and a reasonably low factor. Consequently it demands the knowledge of an upper-bound on the minimum processor speed-up required that can guarantee FT-feasibility. This information is interesting because 1) it provides the system designer with a quick test to check whether a processor of appropriate speed is available in his inventory and 2) it can also provide significant insights into developing a simple utilization based test for FT-feasibility.

In this paper, we examine the FT-feasibility of real-time tasks under at most a single error burst of known length occurring during the hyper-period of the task set-which is the least common multiple of the task periods. We first derive a sufficient condition for the fault tolerance feasibility, leveraging on the optimality of EDF under uni-processor scheduling. We then derive the resource augmentation bounds, specifically the processor speed-up, required to make a real-time task set which is not fault tolerant feasible to be feasible under the error burst. We also show that, if the error burst length is no longer than half the shortest deadline of the task set, the upper-bound on the minimum processor speed-up that guarantees FT-feasibility is 6.

The rest of the section is organized as follows: section 2 discusses the related works and section 3 details the system model. In section 4 we formally define the problem, followed by the fault tolerance feasibility analysis in section 5. We present an example in section 7 followed by our conclusions in 8.

2. RELATED WORK

Avizienis et. al. [2] defines dependability as the ability of a system to deliver a justifiably trusted service. They proposed the use of fault tolerance mechanisms as one of the means to achieve dependability to tackle the threat of faults, that compromise the dependability of the system. The fault tolerance strategy typically involves two stages: error detection and recovery. The recovery process can be classified as error handling and fault handling depending on the process involved in the recovery. The commonly used error handling schemes are rollback, roll forward and compensation using redundancy. The most commonly adopted redundancy technique is the temporal redundancy which involves either the re-execution of the failed software component or the execution of an alternate. In [16], the authors proposed a fault tolerant multi-processor scheduling algorithm for aperiodic tasks. A global optimization method called simulated Annealing [19] was derived from the slow cooling of molten metal to form regular crystalline structure. Attiya and Hamam [1] used Simulated Annealing to allocate tasks in a heterogenous real-time system, maximizing the reliability of the system. Bannister and Trivedi [7] proposed a simple heuristic algorithm that evenly distributes the computational load of the tasks over the nodes. More recently, Islam et.al.[17] proposed a heuristic approach to perform allocation by considering dependability and real-time constraints as well as communication efficiency.

Baruah et. al. [8] derived a sufficient and necessary condition for a set of real-time tasks to be feasible on a uniprocessor. They used the optimality of EDF to derive these conditions i.e., if the task set if EDF schedulable, then it is feasible. Here the feasibility refers to the existence of a real-time scheduling algorithm that can schedule the task set without any deadline misses. In [9], the authors presented an exact schedulability test for fault tolerant real-time task sets for the Fixed Priority Scheduling (FPS) scheme. They considered time redundancy as the fault tolerance strategy while deriving these tests. Aydin [3] considered the uniprocessor fault tolerance feasibility of a real-time task set under a k-fault scenario. They presented an exact feasibility analysis for the real-time task set to be fault tolerant, leveraging on the optimality of EDF. The paper also proposed a dynamic programming technique to calculate the worst case recovery overhead for task sets scheduled using EDF. The k-fault scenario may not be a realistic model; a more realistic model might be to consider fault/error bursts e.g., single event upsets caused due to radiation when an automobile passes through the vicinity of a radiation source, rather than considering a maximum of k faults [4]. Pathan et. al [23] extended this [3] analysis to FPS and derived a necessary and sufficient condition for the fault tolerance feasibility of real-time tasks scheduled using FPS. They assumed no more than k faults every largest relative deadline in the task set. However, as mentioned earlier, the k fault model may not be realistic as the faults normally may occur for a duration. Zhu et. al [26] studied the effects of power management on the reliability of the system and showed that energy management techniques detrimentally affect the reliability of the system. Later, they [25] proposed reliability aware energy management techniques. The technique involves, scheduling a recovery at the maximum processor frequency before executing any task at a lower frequency. Many et. al. [22] considered the FPS schedulability of a set of real-time tasks under a fault burst and derived an equation to find the response times of tasks scheduled under the burst. Additionally they also presented a fault resilience evaluation method. Aysan et.al. [5] derived a sufficient condition to guarantee the schedulability of a task set using FPS under an error burst. They presented a probabilistic burst error model and derived probabilistic schedulability guarantees for the task set. Earlier, they [14] presented a method to maximize the schedulability of mixed criticality real-time tasks using FPS. This was achieved by exploiting the ability of EDF to achieve 100% utilization to embed primary and alternates in the schedule. They achieved this by deriving feasibility windows and then deriving fixed priorities for the tasks and their alternates [15] which was later extended to schedule mixed criticality messages on the Controller Area Network (CAN) [6], as well as to schedule tasks on a distributed real-time system under safety constraints [24].

Resource augmentation [18], is a technique used to understand how much extra resources a scheduler requires such that it can provide a specific guarantee with respect to some constraints. Here, the scheduler under study is given extra resources such as more number of processors or faster processors, such that a certain goal is achieved. Kalyanasundaram et. al [18] first introduced resource augmentation, in which they studied the effectiveness of online scheduling of real-time tasks showing that augmenting the processor with more speed can achieve the same effect as clairvoyance while scheduling tasks online. Davis et. al. [12] used resource augmentation to study the effectiveness of fixed priority schedulers in scheduling all the feasible task sets. They derived resource augmentation bounds on the processor speed-up required for a fixed priority scheduler to schedule all the task sets scheduled by an optimal scheduling algorithm leveraging on the optimality of the Earliest Deadline First (EDF) algorithm.

We leverage on the optimality of EDF to derive a sufficient condition for the FT-feasibility of the real-time tasks under an error burst. Our fault tolerance feasibility analysis is very much similar to [3], with the exception that we consider error bursts affecting the task executions, rather than a bounded number of task execution failures. We then examine the use of processor speed-up to guarantee the FT-feasibility of a task set under the error burst using which we derive resource augmentation bounds for FT-feasibility.

3. SYSTEM MODEL

In this section, we describe the system model and the notations used in this paper.

3.1 Task model

We consider a set of sporadic real-time tasks $\Gamma = \{\tau_1, \tau_2, ...\tau_n\}$, where each τ_i has a minimum inter-arrival time, T_i, a worst case execution time, C_i^S at speed S, and a relative deadline, D_i. We assume that the tasks are ordered according to their increasing deadlines. Each of these tasks generate a potentially infinite sequence of jobs, where the j^{th} job of the i^{th} task is denoted by $\tau_{i,j}$. A job $\tau_{i,j}$ is released at time $(j-1)T_i$ and has to complete its execution no later than $(j-1)T_i + D_i$ in order to meet its deadline. Additionally, let $\{d_1, d_2, ..., d_m\}$ denote the set of absolute deadlines of the task set in the LCM, ordered in the increasing order i.e,. $\forall \tau_i \in \Gamma$, $d_i < d_{i+1}$, where LCM represents the Least Common Multiple of the time periods of the tasks.

The utilization U_i of a task τ_i executing on a processor at speed S is defined as $U_i^S = \frac{C_i^S}{T_i}$, and the utilization of the entire task set is given by $U^S = \sum_{i=1}^{n} U_i^S$. The demand bound function [8] of a task τ_i, on a processor of speed 1, during an interval t is given by:

$$DBF_i(t) = max\left(0, 1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor\right) C_i^1$$

3.2 Scheduling Model and Fault Tolerance Strategy

It is known that EDF is optimal under a work conserving uniprocessor scheduling scheme, i.e., a work conserving EDF can schedule all task sets which are schedulable by any other work conserving scheduler [13]. Thus, if a valid schedule exists for a particular task set, then EDF can feasibly schedule it. We leverage the optimality of EDF to study the fault tolerance feasibility of real-time tasks on a uni-processor.

Most of the previous works treat an error as a singleton event. In this paper, we consider an error burst which is a series of errors occurring within a specific time interval that makes it impossible to perform any meaningful task executions during that interval. We assume a known upper-bound on the length of the error burst during the LCM denoted by T_{length}. We assume that all the task executions during the error burst fails, and the failure detection happens at the end of the task execution, before its completion. The employed fault tolerance strategy is the re-execution of the failed task or the execution of an alternate task before the original deadline. The fault tolerance strategy assumes that the alternates have the same deadline as the original task (the primary) and they are executed along with the rest of the tasks according to EDF. Consequently, the alternates can also be hit by the error burst, and the alternates are scheduled until one successful execution of the task is achieved. The WCET of the alternates is assumed to be no greater than the WCET of the original task.

3.3 Execution Time Model

In our approach we assume a linear relationship between execution time and processor speed [12] [18]. To ease the readability, and without loss of generality, we assume that the task set is initially executing on a processor of speed $S = 1$. Hence, if C_i^1 is the execution time at speed $S = 1$, for any $S > 1$:

$$C_i^S = \frac{C_i^1}{S}$$

Thus the speed required to obtain an execution time of C_i^S is given by:

$$S = \frac{C_i^1}{C_i^S}$$

This model also allows the use of processor speed-up factors and processor speeds interchangeably. Changing the processor speed from $S = 1$ to $S = a$, is equivalent to speeding up the processor by a factor of 'a'. We also assume that the number of clock ticks required to execute a task τ_i is equal to the execution time of τ_i at speed $S = 1$. Hence, $DBF_i(t)$ denotes the number of clock ticks requested in the time interval t on a processor of speed $S = 1$. Consequently, when a processor of speed 'a' is used, the total time requested by the tasks during the time interval t becomes $\frac{DBF_i(t)}{a}$.

4. PROBLEM DESCRIPTION

In this paper, we address the following questions:

1 How to determine the FT-feasibility of a given set of temporally redundant real-time tasks under an error burst of known upper-bounded length during LCM?

A followup question is:

2 If the real-time task set is not found to be FT-feasible, what is the lowest processor speed-up that guarantees its FT-feasibility under the error burst?

5. FAULT TOLERANCE FEASIBILITY ANALYSIS

In this section, we present the proposed fault tolerance feasibility analysis and derive the processor speed-up required to guarantee the FT-feasibility of a real-time task set, under an error burst.

Due to the error detection mechanism assumed to be performed at the end of the tasks' executions, in the analysis we account for the WCET of the primary and alternate tasks under the error burst. If the error burst starts just before any job of τ_i finishes its execution, the rest of the execution of τ_i is outside the influence of the error burst. We define the execution of τ_i that occurs outside the error burst as the maximum wasted execution time of τ_i.

Definition 1. *The Maximum Wasted Execution Time (MWET) of a task τ_i hit by an error burst, is defined as the execution time of the primary or an alternate of τ_i which lies outside the error burst, that leads to the largest wastage of the processor utilization.*

An example of the maximum wasted execution time of a task is shown in figure 1. In any time interval t, the er-

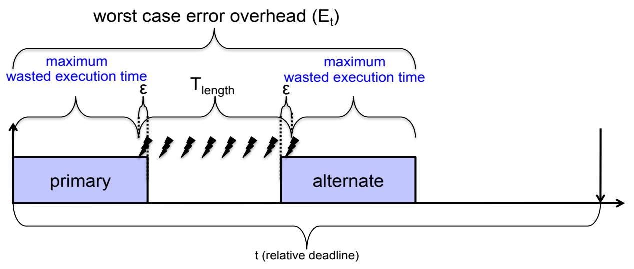
worst case error overhead (E_t)
maximum wasted execution time
ε
T_{length}
ε
maximum wasted execution time
primary
alternate
t (relative deadline)

Figure 1: The worst case error overhead due to error bursts on a single task

ror burst can hit multiple tasks ($\tau_i' s$) leading to many such maximum wasted execution times (MWET) that wastes the processor time. The worst case sum of all the possible maximum wasted execution times of all the tasks until t gives the worst case temporal wastage (WCTW) in the interval t, that leads to the largest overhead outside the error burst. This is formally defined in the following definition.

Definition 2. *The Worst Case Temporal Wastage (WCTW) during a time interval t, denoted by $W_{err}(t)$, is defined as the largest possible temporal overhead which lies outside the error burst, that occurs due to the execution of the MWETs of all the failed primaries and alternates in the interval t, that have their releases and deadlines within t.*

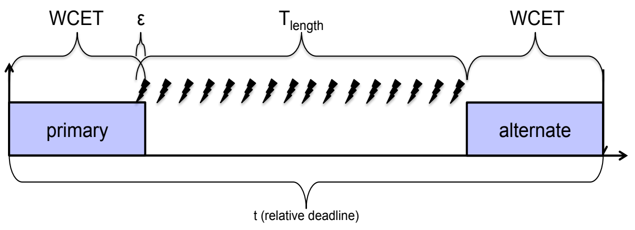

Figure 2: The maximum length of the burst error.

We now identify a necessary condition for FT-feasibility of the set of real-time tasks.

Lemma 5.1. *A necessary condition for the FT-feasibility of a task set Γ is,*

$$T_{length} \le \min_{\forall \tau_i \in \Gamma} (D_i - 2C_i) + \epsilon$$

Proof. The proof for this lemma can be easily seen from figure 2. If the burst length is greater than $\min(D_i - 2C_i + \epsilon)$ where $\forall \tau_i \in \Gamma$, for any task τ_j with a deadline t, it is impossible to guarantee a successful execution τ_j before the deadline t. □

Some assumptions : In the rest of the section, we consider any time instant t' when a job $\tau_{i,j}$ is executing on the processor. Unless stated otherwise, we assume that the job $\tau_{i,j}$ is the first job to be hit by the error burst and the error burst starts at the time instant t'. We also consider a time instant t which is the absolute deadline of $\tau_{i,j}$, $t > t'$. The worst case temporal wastage occur when all the jobs arrive in a strictly periodic manner. Our strategy of finding the FT-feasibility is as follows- we assume that even under the error burst, there are no deadline misses in the schedule, and then we derive the sufficient condition for this to be true.

Lemma 5.2. *If no task is released at or after time t', that has an absolute deadline less than or equal to t, the worst case temporal wastage $W_{err}(t)$ at t is given by:*

$$W_{err}(t) = 2(C_i - \epsilon)$$

Proof. According to our assumption, every job of a task released between time t' and t has an absolute deadline greater than the deadline of $\tau_{i,j}$. The job $\tau_{i,j}$ has to finish its execution for any other job to start its execution. Thus, $\tau_{i,j}$ is the only job that is hit by the error burst. This is because, every job present in the ready queue and every job released after time t', has an absolute deadline later than the absolute deadline of $\tau_{i,j}$, and will execute only after $\tau_{i,j}$ completes its execution successfully, since we assume an EDF scheduler. Thus $W_{err}(t)$ is given in the scenario when the error burst starts just before the primary of $\tau_{i,j}$ completes its execution and just after the last failed alternate of $\tau_{i,j}$ starts its execution (see figure 1). Hence, in this case:

$$W_{err}(t) = 2(C_i - \epsilon)$$

The WCTW at time t is thus equal to twice the MWET of τ_i. □

Observation 5.1. *Every task that is released at or after time t', having an absolute deadline less than or equal to t, will have a relative deadline less than or equal to D_i.*

This is quite straight forward as $\tau_{i,j}$ has been released at a time instant less than or equal to t'. Thus every task that is released after t' having an absolute deadline less than or equal to t must have a relative deadline less than the relative deadline of τ_i.

All the jobs that are released in the interval $[t', t]$, having a later deadline than t will not be hit by the error burst. This is proved in the following lemma.

Lemma 5.3. *No job $\tau_{a,b}$ released in the interval $[t', t]$, having an absolute deadline $bT_a + D_a > t$, can be hit by the error burst.*

Proof. The job $\tau_{a,b}$ will be scheduled only after $\tau_{i,j}$ has completed one successful execution since $\tau_{i,j}$ has the earliest deadline. According to our assumption, the task set is schedulable even under the error burst. Thus, the error burst would have ended before $\tau_{a,b}$ started its execution, since $\tau_{i,j}$ completed one successful execution. □

We now show that the WCTW at the absolute deadlines of jobs released in the interval $[t,' t]$, having a later deadline than t, is equal to the WCTW at time instant t.

Lemma 5.4. *The $W_{err}(d_l)$ for any job $\tau_{a,b}$ that is released in the interval $[t', t]$, having an absolute deadline denoted by $d_l = bT_a + D_a > t$, is given by:*

$$W_{err}(d_l) = W_{err}(d_{l-1})$$

Proof. When $\tau_{a,b}$ starts its execution, the value of $W_{err}(d_l)$ is equal to the value of $W_{err}(t)$, since no job with a deadline greater than t is hit by the error burst (consequently no 'new' alternates are executed). Thus, in general we can say that $W_{err}(d_l) = W_{err}(d_{l-1})$ for such a job $\tau_{a,b}$, as the same argument holds for every such job having an earlier absolute deadline than d_l. □

In the next lemma, we bound the contribution of $\tau_{i,j}$ to the WCTW at t when more than one task is hit by the error burst in the interval $[t', t]$.

Lemma 5.5. *If the error burst hits more than one task in the interval $[t', t]$, the contribution of $\tau_{i,j}$ to $W_{err}(t)$ at time t is $2(C_i - \epsilon)$.*

Proof. In this case, the primary of $\tau_{i,j}$ is hit ϵ units before it completes its execution, and one of its failed alternates is preempted immediately (ϵ time units) after it starts its execution. Assume that $\tau_{a,b}$ is the task preempting $\tau_{i,j}$, which means that $\tau_{a,b}$ has an earlier absolute deadline than $\tau_{i,j}$. Using the similar argument from lemma 5.3,

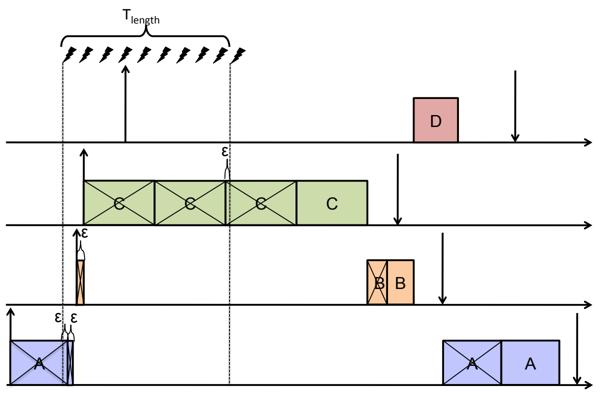

Figure 3: Error burst hitting multiple jobs

the error burst will end before $\tau_{a,b}$ completes one successful execution. When $\tau_{i,j}$ resumes its execution, its remaining execution, i.e., $C_i - \epsilon$ is wasted as it was hit by the error burst just before it was preempted. The alternate of job $\tau_{i,j}$ then executes successfully as the error burst has already ended. Thus the contribution of $\tau_{i,j}$ to to $W_{err}(t)$ at time t is $2(C_i - \epsilon)$. □

Only either the primary or one of the alternates of the jobs hit by the error burst in the interval $[t', t]$ contributes to the WCTW at time t. While a proof of it under FPS has been presented in [4], in this paper, we extend it to EDF in the following lemma since the assumptions under FPS are no longer valid under EDF.

Lemma 5.6. *If the error burst hits more than one task in the interval $[t', t]$, only either the primary, or exactly one alternate of each task, other than τ_i, that is hit by the error burst will contribute to $W_{err}(t)$ at time t.*

Proof. We consider only the jobs that are executing in the interval $[t', t]$. This is because only these jobs have to finish their execution before their corresponding absolute deadlines, so that $\tau_{i,j}$ can finish its execution no later than time instant t. Any job having an absolute deadline greater than t will not affect the execution of $\tau_{i,j}$.

According to our assumption, $\tau_{i,j}$ is the first job to be hit by the error burst. Let the job $\tau_{a,b}$ that has a release time and deadline in the interval $[t', t]$, be the next task to be hit by the error burst. For the task set to be schedulable, $\tau_{a,b}$ needs to recover before its absolute deadline i.e., it must have one successful execution before its absolute deadline. Additionally, there should not be any deadline misses in the rest of the schedule until the LCM. The execution of the job $\tau_{a,b}$ is under the error burst and its contribution to $W_{err}(t)$ is maximum when either:

1. The primary or one of the failed alternates of job $\tau_{a,b}$ is immediately preempted by a higher priority job $\tau_{e,f}$ as soon as it starts execution.

   In this case, the error burst will end before the job $\tau_{e,f}$ completes one successful execution, after which the remaining executions of the failed primary or alternate of $\tau_{a,b}$, which was preempted, execute to completion. Thus the maximum processor time wasted by $\tau_{a,b}$ is

$(C_a - \epsilon)$, before it can successfully execute, according to the definition 1.

2. The error burst ends just before the last failed alternate of $\tau_{a,b}$ starts executing.

   This is the case when $\tau_{a,b}$ is the only job other than $\tau_{i,j}$ that is hit by the error burst. In this case, the contribution of $\tau_{a,b}$ to $W_{err}(t)$ is maximum when the error burst ends just after the start of an alternate of $\tau_{a,b}$. Hence, according to definition 1, the maximum processor time wasted is $(C_a - \epsilon)$, before $\tau_{a,b}$ successfully executes.

In both cases, maximum execution of $\tau_{a,b}$ that can lie outside the region of the error burst is $(C_a - \epsilon)$. The above argument can be repeated for all higher priority jobs $\tau_{e,f}$ that are released between the release time of $\tau_{a,b}$ and its absolute deadline.

Thus we can see that, if the error burst hits more than one job, either only the primary or exactly one alternate of each task other than $\tau_{i,j}$, that is hit by the error burst, will contribute to $W_{err}(t)$ at time t. □

We have thus bounded the contributions of the jobs scheduled in the interval $[t', t]$ to the $W_{err}(t)$ at t, when the error burst hits more than one job. An example, when the error burst hits multiple jobs, is given in figure 3. We now derive the $W_{err}(t)$ when the error burst hits more than one job in the interval $[t', t]$, in the general case.

LEMMA 5.7. *If the error burst hits more than one job in the interval $[t', t]$, the worst case temporal wastage $W_{err}(t)$ is given by:*

$$W_{err}(t) = 2(C_i - \epsilon) + \sum_{\forall \tau_k \in \Gamma : D_k \leq D_i} (C_k - \epsilon)$$

PROOF. We know from lemma 5.6 that only either the primary or one of the alternate of the failed tasks that have release times and deadlines in $[t', t]$ will contribute to the worst case temporal wastage at t. Thus the total contribution to $W_{err}(t)$ is the maximum when every job $\tau_{a,b}$ released in the interval $[t', t]$, that has a deadline no later than t, is hit by an error burst and leaves $C_a - \epsilon$ time units of execution outside the error burst. This scenario occurs when the tasks released in the interval $[t', t]$ preempt each other in a nested manner, with every preemption occurring ϵ units after the start of the execution of the preempted task.

The tasks that may be potentially released in the interval $[t', t]$ are the tasks that have relative deadlines less than or equal to D_i (observation 5.1). Thus following the reasoning in lemma 5.6, only either the primary or exactly one alternate of each of the failed tasks other than $\tau_{i,j}$ will contribute to the worst case temporal wastage. The worst case contribution of $\tau_{i,j}$ is $2(C_i - \epsilon)$ according to lemma 5.5.

Thus, the worst case temporal wastage at time t is equal to,

$$W_{err}(t) = 2(C_i - \epsilon) + \sum_{\forall \tau_k \in \Gamma : D_k \leq D_i} (C_k - \epsilon)$$

We thus obtain the WCTW at t, when the error burst hits multiple jobs. □

Let us now consider the case when the error burst hits only a single job and $\tau_{i,j}$ is not necessarily the job to be hit. This means that any task in the interval $[t', t]$ could be hit by the error burst and the WCTW at t is given by the following lemma.

LEMMA 5.8. *If the error burst hits only a single job, not necessarily $\tau_{i,j}$, in the time interval $[t', t]$, the worst case temporal wastage at time t is given by:*

$$W_{err}(t) = \max_{\forall \tau_k \in \Gamma : D_k \leq D_i} \{2(C_k - \epsilon), 2(C_i - \epsilon)\}$$

PROOF. According to the observation 5.1, all the jobs that are completely scheduled in the interval $[t', t]$ are the jobs of the tasks with a relative deadline less than or equal to the relative deadline of τ_i. Thus, if only one job is hit by the error burst in the interval $[t', t]$, the maximum contribution to the worst case temporal wastage at t is twice the maximum of the worst case execution time wastage (w_k) of τ_k, if a job of τ_k is scheduled in the interval $[t', t]$. This is the case when the error burst starts just before the primary of the task hit by the burst finishes its execution and ends just after the last failed alternate has started its execution, as shown in lemma 5.2. Here, τ_k can be either τ_i or, according to observation 5.1 and using lemma 5.3, any τ_k such that $D_k \leq D_i$. Hence,

$$W_{err}(t) = \max_{\forall \tau_k \in \Gamma : D_k \leq D_i} \{2(C_k - \epsilon), 2(C_i - \epsilon)\}$$

We have thus derived the worst case temporal wastage for the final scenario. □

We now propose one of our main theorems which bounds the WCTW at t, which we later use to reason about the FT-feasibility.

THEOREM 5.1. *The worst case temporal wastage $W_{err}(t)$ at any time instant t, where $t = d_l = jT_i + D_i$ for any job $\tau_{i,j}$, is given by:*

$$W_{err}(t) = \max(x, y, W_{err}(d_{l-1}))$$

Here,

$$x = \max_{\forall \tau_k \in \Gamma : D_k \leq D_i} \{2(C_k - \epsilon)\}$$

$$y = 2(C_i - \epsilon) + \sum_{\forall \tau_k \in \Gamma : D_k \leq D_i} (C_k - \epsilon)$$

PROOF. The proof follows from lemma 5.4, 5.7, 5.8. At deadline d_l, according to lemma 5.7, if the error burst hits more tasks in addition to $\tau_{i,j}$,

$$W_{err}(t) = 2(C_i - \epsilon) + \sum_{\forall \tau_k \in \Gamma : D_k \leq D_i} (C_k - \epsilon)$$

According to lemma 5.8, at deadline d_l, if the error burst hits only one task, the $W_{err}(t)$ is given by the maximum of the MWETs of the tasks scheduled until d_l, thus,

$$W_{err}(t) = \max_{\forall \tau_k \in \Gamma : D_k \leq D_i} \{2(C_k - \epsilon), 2(C_i - \epsilon)\}$$

Finally, according to lemma 5.4, if $\tau_{i,j}$ is a job that has an absolute deadline greater than the deadline of the job first hit by the error burst, then,

$$W_{err}(d_l) = W_{err}(d_{l-1})$$

Hence, $W_{err}(t)$, where $t = jT_i + D_i$ for any $\tau_{i,j}$ is given by the maximum of the $W_{err}(t)$ given by lemmas 5.4, 5.7, 5.8. □

We now define the worst case error overhead at any time t which is the worst case overhead involved in tolerating faults.

Definition 3. *The worst case error overhead E_t, in any time interval t, is defined as the sum of the error burst length T_{length} and the worst case temporal overhead in the time interval t.*

$$E_t = T_{length} + W_{err}(t)$$

An example of the worst case error overhead at a time instant t i.e., E_t is shown in figure 1. We now build on the demand bound analysis proposed by Baruah et. al [8] and define the sufficient condition for FT-feasibility under an error burst.

Theorem 5.2. *A real-time task set Γ is FT-feasible under an error burst of length T_{length} if, $\forall t = kT_j + D_j, \forall \tau_j \in \Gamma$ and $t \leq LCM$,*

$$E_t + \sum_{i=1}^{n} DBF_i(t) \leq t$$

Proof Sketch. When the above condition is satisfied, there is sufficient slack in the schedule, during any time interval t, for the execution of the real-time tasks and the alternates of the failed tasks, outside the region of the error burst.

Suppose that the above condition is not satisfied for some t, i.e.,

$$W_{err}(t) + \sum_{i=1}^{n} DBF_i(t) > t - T_{length}$$

This means that during some time interval, the total execution demanded by the task set exceeds the size of that interval and hence the task set is not feasible. The formal proof is similar to the proof presented in [3]. □

However, depending on the real-time schedule, the actual maximum temporal wastage at t may or may not be equal to the worst case. Hence if the lemma is not satisfied, no guarantees can be given about the FT-feasibility using the above feasibility test. Thus the above theorem is only a sufficient test for FT-feasibility.

6. RESOURCE AUGMENTATION FOR FT-FEASIBILITY

In this section, we examine the resource augmentation bounds that guarantees the FT-feasibility of a set of real-time tasks under a known error burst length. We first, in the following theorem, derive the exact processor speed-up that guarantees the FT-feasibility of the real-time task set.

Theorem 6.1. *The minimum processor speed-up required to guarantee the FT-feasibility of a real-time task set Γ under a burst error of length T_{length} is given by:*

$$S = \max_{\forall t} \left\{ \frac{W_{err}(t) + \sum_{i=1}^{n} DBF_i(t)}{t - T_{length}} \right\}$$

Proof. If any given task set Γ is not FT-feasible on a processor of speed $S = 1$, there exists a time instant t such that,

$$T_{length} + W_{err}(t) + \sum_{i=1}^{n} DBF_i(t) > t$$

Suppose that speeding up the processor by a factor of S will ensure its FT-feasibility. We get,

$$T_{length} + \frac{W_{err}(t)}{S} + \frac{\sum_{i=1}^{n} DBF_i(t)}{S} \leq t$$

Thus, $\forall t$,

$$\frac{W_{err}(t) + \sum_{i=1}^{n} DBF_i(t)}{S} \leq t - T_{length}$$

Solving for S we get $\forall t$,

$$S \geq \frac{W_{err}(t) + \sum_{i=1}^{n} DBF_i(t)}{t - T_{length}}$$

Hence,

$$S = \max_{\forall t \in aT_j + D_j, t \leq LCM} \left\{ \frac{W_{err}(t) + \sum_{i=1}^{n} DBF_i(t)}{t - T_{length}} \right\}$$

We thus obtain the minimum processor speed-up required to guarantee FT-feasibility. □

In order to derive upper-bounds on the processor speed-up that guarantees FT-feasibility, we bound the $W_{err}(t)$ at any time instant t.

Lemma 6.1. *The worst case temporal wastage $W_{err}(t)$, $t \in \{d_1, d_2, ..., d_m\}$, is upper-bounded by:*

$$W_{err}(t) \leq 2 \sum_{i=1}^{n} DBF_i(t)$$

Proof. At deadline d_1, which is the shortest relative deadline D_1, the worst case temporal wastage $W_{err}(t)$, according to theorem 5.1, is given by:

$$W_{err}(t) = max(x, y)$$

$$x = \max_{\forall \tau_k \in \Gamma : D_k \leq D_1} \{2(C_k - \epsilon)\}$$

$$y = 2(C_1 - \epsilon) + \sum_{\forall \tau_k \in \Gamma : D_k \leq D_1} (C_k - \epsilon) = 2(C_1 - \epsilon)$$

Here, clearly x or y can be upper-bounded by:

$$x \leq 2 \sum_{i=1}^{n} DBF_i(D_1) \; and \; y \leq 2 \sum_{i=1}^{n} DBF_i(D_1)$$

Consider any absolute deadline d_l of any task τ_i, $d_l = jT_i + D_i$. The $W_{err}(t)$ is given by:

$$W_{err}(t) \leq max(x, y, d_{l-1})$$

Here again, we can see that x and y can be bounded by:

$$x \leq 2 \sum_{i=1}^{n} DBF_i(d_l) \; and \; y \leq 2 \sum_{i=1}^{n} DBF_i(d_l)$$

Hence for any t,

$$W_{err}(t) \leq 2 \sum_{i=1}^{n} DBF_i(t)$$

This gives an upper-bound on the worst case temporal wastage in any time interval t. □

Using the above bounds on the $W_{err}(t)$, we derive an upper-bound on the processor speed-up that guarantees FT-feasibility in the following theorem.

THEOREM 6.2. *The minimum processor speed-up S_b that guarantees the FT-feasibility of a set of real-time tasks Γ under an error burst of length T_{length} is upper-bounded by:*

$$S_b \leq \frac{3y}{y-1}$$

where $y = \frac{t}{T_{length}}$, $t \in \{d_1, d_2, ..., d_m\}$.

PROOF. According to lemma 6.1, the upper-bound on the $W_{err}(t)$, $t \in \{d_1, d_2, ..., d_m\}$ is given by:

$$W_{err}(t) \leq 2 \sum_{i=1}^{n} DBF_i(t)$$

According to theorem 6.1,

$$S = \max_{\forall t} \left\{ \frac{W_{err}(t) + \sum_{i=1}^{n} DBF_i(t)}{t - T_{length}} \right\}$$

Substituting the upper-bounds on $W_{err}(t)$, we get, $\forall t \in \{d_1, d_2, ..., d_m\}$,

$$S \leq \frac{3 \sum_{i=1}^{n} DBF_i(t)}{t - T_{length}}$$

Since we assume that the original task set is schedulable, $\forall t \in \{d_1, d_2, ..., d_m\}$,

$$\sum_{i=1}^{n} DBF_i(t) \leq t$$

Substituting for $\sum_{i=1}^{n} DBF_i(t)$, $\forall t \in \{d_1, d_2, ..., d_m\}$, we get the upper-bound on the required speed-up denoted by S_b,

$$S_b \leq \frac{3t}{t - T_{length}}$$

Thus,

$$S_b \leq \frac{3y}{y-1}$$

where $y = \frac{t}{T_{length}}$, $t \in \{d_1, d_2, ..., d_m\}$.

The largest value of S_b is obtained at $d_1 = D_1$, the shortest relative deadline of the task set. $\square$

We now derive the resource augmentation bounds for the case when the error burst length is no longer than half the shortest deadline.

THEOREM 6.3. *The upper-bound on the minimum processor speed-up S_b that guarantees the FT-feasibility of a set of real-time tasks Γ under an error burst of length T_{length} such that for any time interval $t \in \{d_1, d_2, ..., d_m\}$, $y \geq 2$, $y = \frac{t}{T_{length}}$, is given by:*

$$S_b \leq 6$$

PROOF. This is straight away obtained from theorem 6.2, by evaluating the limits at $y = 2$ and $y = \infty$. $\square$

We have thus presented a sufficient condition for the fault tolerance feasibility of a task set under an error burst, and derived upper-bounds on the processor speed-up required to guarantee the fault tolerance feasibility, if the sufficient condition fails for some task set. We have proved that if the error burst length is no longer than half the shortest deadline of the task set, the resource augmentation bound that guarantees the FT-feasibility is 6.

Task	C_i	D_i	T_i
A	1	5	6
B	1	9	9
C	2	18	18

Table 1: Example task set

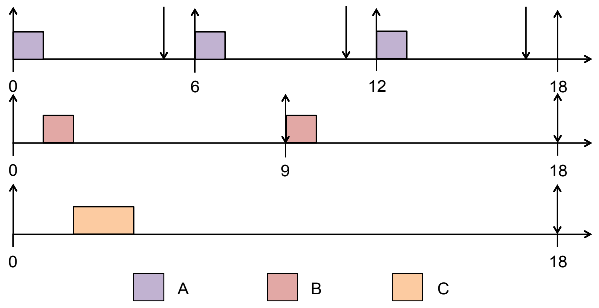

Figure 4: EDF schedule

7. EXAMPLE

We illustrate our feasibility analysis and resource augmentation bounds using a simple example. Consider a real-time task set as shown in table 1 with 3 tasks. To illustrate the use of processor speed-up to enable FT-feasibility, let us assume that the error burst length $T_{length} = 4$. The demand bound until the first absolute deadline 5 (demanded by task A) is equal to:

$$\sum_{i=A}^{C} DBF_i(5) = 1$$

Suppose the primary of task A is hit by the error burst, the maximum time is *wasted* when the burst hits the primary just before it finishes its execution. At time instant $t = 1$, the alternate of task A starts its execution and this is again hit by the burst. At $t = 2$, the alternate is again executed, which is again hit by the error burst. Alternates continue to execute and at time instant $t = 4.9$, during the execution of one of the alternates, the error burst ends. It can be easily seen that task A does not have sufficient slack outside the error burst to complete one successful execution since it has a deadline at $t = 5$. One of the fault scenarios where $T_{length} = 4$ is illustrated in figure 5, and there is a deadline miss on task A. Formally,

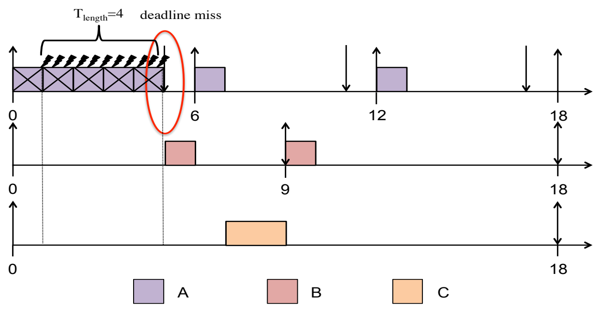

Figure 5: EDF schedule under faults with $T_{length} = 4$

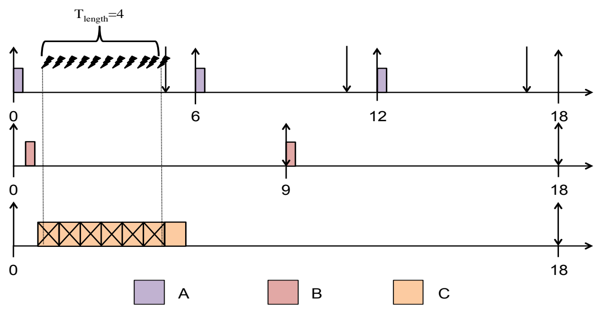

Figure 6: EDF schedule under faults after a speed-up of 2.8

$$E_5 + \sum_{i=A}^{C} DBF_i(5) = 5.8 + 1 = 6.8 > 5$$

Similarly at deadline $t = 9$, the demand bound $= 2$. Here the worst possible overheads due to the error burst can be bounded by $E_9 = 6.7$. The worst case temporal wastage (WCTW) occurs when the primary of task B is hit leading to a scenario as in the previous deadline. Additionally, we add one failed alternate from the higher priority task to account for the cases where higher priority tasks preempt the primary or one of the alternates of the task B under consideration. Hence,

$$E_9 + \sum_{i=A}^{C} DBF_i(9) = 6.7 + 2 = 8.7 < 9$$

Similarly, we calculate the processor demand bounds at all the absolute deadlines.

$$E_{11} + \sum_{i=A}^{C} DBF_i(11) = 6.7 + 3 = 9.7 < 12$$

$$E_{17} + \sum_{i=A}^{C} DBF_i(17) = 6.7 + 4 = 10.7 < 17$$

$$E_{18} + \sum_{i=A}^{C} DBF_i(18) = 9.6 + 6 = 15.6 < 18$$

Thus, the only possibility of a deadline miss due to the error burst is at time $t = 5$. The speed-up required to guarantee FT-feasibility is,

$$S = max\left(\frac{2.8}{1}, \frac{4.7}{5}, \frac{5.7}{8}, \frac{6.7}{13}, \frac{11.7}{14}\right) = \frac{2.8}{1} = 2.8$$

When we increase the processor speed to 2.8, during the time interval $[0, 5]$, the total value of $W_{err}(t) + \sum_{i=1}^{n} DBF_A^C(t) = \frac{1.8+1}{2.8} = 1$. Hence,

$$E_5 + \sum_{i=1}^{n} DBF_i(5) = 1 + 4 = 5$$

The same scenario in figure 5 on a processor that is 2.8 times faster is given in figure 6. Observe that there is no deadline miss on task A in the schedule in figure 6 under the error burst, after the speed-up. Thus, we can prevent a deadline miss at $t = 5$ by using a processor that is 2.8 times faster.

8. CONCLUSIONS

In this paper, we have examined the use of resource augmentation to guarantee the fault tolerance feasibility of a set of real-time tasks under an error burst. In this context, we derive a sufficient condition under the assumption of no more than a single error burst occurrence during the hyper-period of the tasks. For the cases where the sufficient condition fails, we also derive the necessary speed-up that guarantees the fault tolerance feasibility under the burst. We show that, if the length of the error burst is no more than half the shortest deadline of the task set, the minimum processor speed-up that guarantees the fault tolerance feasibility is upper-bounded by 6.

The proposed method adds a new capability in the system designer's repertoire for analyzing the fault tolerance feasibility of a given set of real-time tasks under an error burst and derive essential resource augmentation requirements. We intend to extend the proposed approach to more severe error scenarios as well as to distributed systems in future.

9. REFERENCES

[1] G. Attiya and Y. Hamam. Task allocation for maximizing reliability of distributed systems: A simulated annealing approach. *Journal of Parallel and Distributed Computing*, October 2006.

[2] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable Secure Computing*, January 2004.

[3] H. Aydin. Exact fault-sensitive feasibility analysis of real-time tasks. *IEEE Transactions on Computers*, October 2007.

[4] H. Aysan. Fault-tolerance strategies and probabilistic guarantees for real-time systems. In *PhD thesis, Malardalen University*, June 2012.

[5] H. Aysan, R. Dobrin, S. Punnekkat, and R. Johansson. Probabilistic schedulability guarantees for dependable real-time systems under error bursts. In *The 8th IEEE International Conference on Embedded Software and Systems*, November 2011.

[6] H. Aysan, A. Thekkilakattil, R. Dobrin, and S. Punnekkat. Efficient fault tolerant scheduling on controller area network (CAN). In *15th International Conference on Emerging Technologies and Factory Automation*, September 2010.

[7] J. A. Bannister and K. S. Trivedi. Task Allocation in Fault-Tolerant Distributed Systems. *Acta Informatica, Springer-Verlag*, 1983.

[8] S. K. Baruah, L. E. Rosier, and R. R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, October 1990.

[9] A. Burns, R. Davis, and S. Punnekkat. Feasibility analysis of fault-tolerant real-time task sets. In *The 8th Euromicro Workshop on Real-Time Systems*, June 1996.

[10] G. C. Buttazzo. Rate monotonic vs. EDF: judgment day. *Real-Time Systems Journal*, January 2005.

[11] M. Cirinei, A. Mancina, D. Cantini, P. Gai, and L. Palopoli. An educational open source real-time kernel for small embedded control systems. In

Computer and Information Sciences. Springer Berlin / Heidelberg, 2004.

[12] R. Davis, T. RothvoSS, S. Baruah, and A. Burns. Exact quantification of the sub-optimality of uniprocessor fixed priority pre-emptive scheduling. *Real-Time Systems*, July 2009.

[13] M. L. Dertouzos. Control robotics: The procedural control of physical processes. In *IFIP Congress*, 1974.

[14] R. Dobrin, H. Aysan, and S. Punnekkat. Maximizing the fault tolerance capability of fixed priority schedules. In *The 14th IEEE Internationl Conference on Embedded and Real-Time Computing Systems and Applications*, August 2008.

[15] R. Dobrin, G. Fohler, and P. Puschner. Translating off-line schedules into task attributes for fixed priority scheduling. In *Real-Time Systems Symposium*, December 2001.

[16] S. Ghosh, R. Melhem, and D. Mosse. Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems. *IEEE Transactions on Prarallel and Distributed Systems*, March 1997.

[17] S. Islam, R. Lindstrom, and N. Suri. Dependability driven integration of mixed criticality SW components. *Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, April 2006.

[18] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of ACM*, July 2000.

[19] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, May 1983.

[20] B. Lisper. Trends in timing analysis. In *From Model-Driven Design to Resource Management for Distributed Embedded Systems*. Springer Boston, 2006.

[21] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *The Journal of ACM*, January 1973.

[22] F. Many and D. Doose. Scheduling analysis under fault bursts. In *The 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, April 2011.

[23] R. Pathan and J. Jonsson. Exact fault-tolerant feasibility analysis of fixed-priority real-time tasks. In *The16th International Conference on Embedded and Real-Time Computing Systems and Applications*, April 2010.

[24] A. Thekkilakattil, H. Aysan, and S. Punnekkat. Towards a contract-based fault-tolerant scheduling framework for distributed real-time systems. In *The 1st International Workshop on Dependable and Secure Industrial and Embedded Systems*, June 2011.

[25] D. Zhu. Reliability-aware dynamic energy management in dependable embedded real-time systems. In *In Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, April 2006.

[26] D. Zhu, R. Melhem, and D. Mosse. The effects of energy management on reliability in real-time embedded systems. In *Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, November 2004.