Bounding the Effectiveness of Temporal Redundancy in Fault-tolerant Real-time Scheduling under Error Bursts

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

***Abstract*—Reliability is a key requirement in many distributed real-time systems deployed in safety and mission critical applications, and temporal redundancy is a widely employed strategy towards guaranteeing it. The temporal redundancy approach is typically based on task re-executions in form of entire tasks, task alternates or, check-pointing blocks, and each of the re-execution strategies have different impacts on the Fault Tolerance feasibility (FT-feasibility) of the system, which is traditionally defined as the existence of a schedule that guarantees timeliness of all tasks under a specified fault hypothesis.**

In this paper, we propose the use of resource augmentation to quantify the FT-feasibility of real-time task sets and use it to derive limits on the effectiveness of temporal redundancy in fault-tolerant real-time scheduling under error bursts of bounded lengths. We derive the limits for the general case, and then show that for the specific case when the error burst length is no longer than half the shortest deadline, the lower limit on the effectiveness of temporal redundancy is given by the resource augmentation bound 2, while, the corresponding upper-limit is 6. Our proposed approach empowers a system designer to quantify the effectiveness of a particular implementation of temporal redundancy.

I. Introduction

Distributed real-time systems are increasingly being deployed in environments characterized by high frequencies of error occurrences. A distributed real-time system is typically composed of many processing elements, implementing various functionalities, that communicate over a real-time communication media such as the Controller Area Network (CAN). Consequently, in order to guarantee a failure free operation of the distributed system, the software running on the individual processing elements need to be highly reliable. Various fault tolerance mechanisms are employed to increase reliability, that must ensure timeliness even during error occurrences. The use of fault tolerance mechanisms typically imply extra resources, and hence, to reduce cost, a system designer must optimize resource usage while ensuring the timeliness of the system. In a hard real-time system, the events occurring in the environment are mapped to a set of real-time tasks, with the requirement that the task executions must complete before their respective deadlines. A fault that causes an error in a task, may lead to its failure. This failure of a task can be seen as a fault at a higher level of abstraction in the system. Consequently, the use of terminologies like faults, errors and failures depend on the level of abstraction considered [1]. In this paper, we consider error bursts which are continuous streams of errors occurring on the tasks. For example, in the car *Hyundai Accent* manufactured between 2007-2009, electromagnetic interference causes error bursts and may lead to the illumination of the tire pressure warning lights while passing through airport areas or police stations [2].

Temporal redundancy is a widely used approach to provide reliability guarantees in real-time systems [3] [4] [5]. The use of temporal redundancy implies the entire re-execution of a failed task, the execution of a recovery task, or performing a roll-back to the last saved check-point. The original task executions are typically referred to as *primaries* and the re-executions/recovery tasks are referred to as *alternates*. A real-time fault tolerant scheduler that employs the temporal redundancy approach needs to ensure the successful execution of either the primaries or an alternate of all tasks before their respective deadlines, under the specified fault hypothesis.

Traditionally [3] [5], the Fault-Tolerance feasibility (FT-feasibility) of a set of temporally redundant real-time tasks is defined as the existence of a schedule that guarantees the successful execution of all tasks or their alternates before their deadlines under a specified fault hypothesis. Such a definition of FT-feasibility, however, does not provide any additional information to the system designer to optimize system design e.g., *how far* the real-time task set is from being FT-feasible, in case a feasible schedule does not exist? On the other hand, in case a feasible schedule exists, it is equally interesting to reason about how efficient the redundancy scheme is. Such a quantification of FT-feasibility allows a system designer to compare the efficiency of various temporal redundancy based strategies enabling him to choose the best strategy, consequently optimizing system design and potentially reducing costs.

In this paper we derive the limits on the effectiveness of temporal redundancy in guaranteeing FT-feasibility of uniprocessor feasible real-time tasks under error burst occurrences. Previously [5], we derived the resource augmentation bound that guarantees FT-feasibility of a set of real-time tasks under an error burst of known upper-bounded length, when all the failed tasks have to be re-executed entirely. In this paper, building on this result [5], we use processor speed-up[1] as a metric

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

[1]Remember that in case the task set is already FT-feasible, the amount of slow down affordable before the task set becomes infeasible can be used to measure its FT-feasibility. The analysis is the same for this case as well, however, for easiness of presentation we use the term *speed-up*.

to *measure* the FT-feasibility of a set of real-time tasks under error bursts. The resource augmentation bound that guarantees FT-feasibility when the failed tasks have to be entirely re-executed can be seen as the *upper-limit* on the effectiveness of temporal redundancy approach since it represents the *most* that needs to be done, in terms of the speed-up required, to enable FT-feasibility. Any other temporal redundancy based mechanism having an FT-feasibility measure greater than this value (e.g., when using check-pointing in presence of check-pointing overheads) can be seen as inefficient because by using a processor that is only 6 times faster (i.e., a relatively slower processor), the entire failed task can be re-executed.

We then use the check-pointing approach [6][7][8] to find the lower-limit on the effectiveness of temporal redundancy under error bursts. We first derive a test to determine the FT-feasibility of a set of check-pointed real-time tasks under error bursts, and then derive the resource augmentation bound that guarantees FT-feasibility. Under the assumption that check-pointing is performed after every unit of execution and check-pointing overheads are negligible, we obtain the maximum achievable improvement in FT-feasibility using temporal redundancy. The quantification of FT-feasibility, and using it to derive the limits on the effectiveness of temporal redundancy, empowers a system designer with the ability to compare and benchmark different temporal redundancy based implementations, enabling him to perform an optimized system design.

While one may argue that, in reality, the speed-up required to guarantee FT-feasibility may differ due to overheads and hence requires a practical study, we would like to remind the reader that our bounds are meant to provide analytical inputs to the system designer while designing the system. The speed-up required to enable FT-feasibility does depend on the actual overheads when using a particular implementation of temporal redundancy, however the *speed-up bound* of that implementation should ideally lie between our derived bounds. We also remind that our FT-feasibility analysis takes the check-pointing overhead into consideration, and also does not assume that the check-pointing operation is resistant to the error burst (more details in Section II.C), potentially facilitating its use in actual systems.

In Section II, we present the system model after which, in Section III, we derive the limits on the effectiveness of temporal redundancy. We conclude in Section V after presenting a short review of the related works in Section IV.

II. System Model

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

A. Task model

We consider a set of n sporadic real-time tasks Γ= $\{\tau_1, \tau_2, ...\tau_n\}$, where each τ_i is characterized by a minimum inter-arrival time, T_i, a worst case execution time, C_i^S at speed S, and a relative deadline, $D_i \leq T_i$. Accordingly, the WCET of τ_i on a processor of speed $S = 1$ is denoted by C_i^1, which for convenience is also denoted by C_i. We assume that the tasks are ordered according to their increasing deadlines in Γ. Each of tasks generates a potentially infinite sequence of jobs, where the j^{th} job of the i^{th} task is denoted by $\tau_{i,j}$.

When these jobs arrive strictly periodically, 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, for the strict periodic case, 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 (also called hyper-period).

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 [9] of a task τ_i, on a processor of speed 1, during an interval $[0, t]$ is given by:

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

B. Fault and Error Model

A fault is a hypothesized cause of an error [1] that may eventually lead to a failure. We consider those faults occurring in the task that causes an error in the task. An error in a task, that leads to its failure, can be seen as a fault in the system, which can be tolerated by a suitable mechanism like temporal redundancy. Hence the use of the term *faults*, *errors*, *failures* depends on the level of abstraction under consideration. Most previous works treat error occurrence as a singleton event. In this paper, we consider an error burst which is a continuous series of *soft* or *intermittent* errors occurring on the tasks, e.g., radar waves in airport areas produce electromagnetic fields that can cause error bursts on the on-board computer of an aircraft [10] [11]. We assume a known upper-bound on the length of the error burst that is denoted by T_{length}. Two consecutive error burst occurrences are assumed to be separated by at least a time interval equal to the LCM of the task periods. This assumption is reasonable because, for example in the case of a typical rotating air search radar, the time between two exposures is a few seconds [10], and the LCM of many typical applications is a few hundred milliseconds [12]. In other cases, e.g., a car passing through airport areas or police stations, the minimum inter-arrival time between two consecutive error bursts will be even greater. We, however, plan to relax this assumption in a future work. The upper-bound on the error burst can be calculated using the characteristics of the error source e.g., the worst case exposure of an aircraft to a rotating air radar will not exceed 100ms [10].

C. Fault Tolerance Strategy

We assume that the tasks are scheduled according to the Earliest Deadline First (EDF) strategy. We assume that every task τ_i is check-pointed, and the maximum size of its *check-pointing block*, as shown in Figure 1, is given by CP_i^S. The check-pointing overheads are assumed to be included in the check-pointing block and the maximum size of the check-pointing block of τ_i on a processor of speed $S = 1$ given

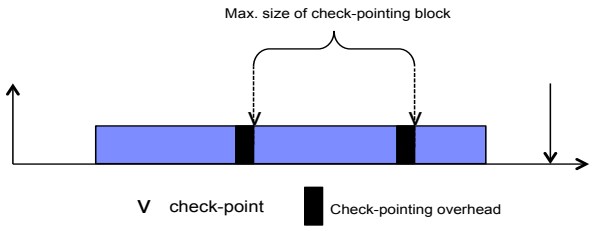

Fig. 1. A check-pointed real-time task

by CP_i^1 is also denoted as CP_i for convenience. We assume that all the task executions during the error burst fail, and the failure detection happens at the end of every check-point before the progress is saved.

A task execution is defined to be successful if all the check-pointing blocks are executed successfully without errors. If an error occurs during the execution of a check-pointing block, that check-pointing block has to be re-executed. We refer to the re-execution of a check-pointing block as an *alternate* of the task for convenience. The alternates can also be hit by the error burst, and the alternates are scheduled until one successful execution is achieved. Remember that we assume that the check-pointing overheads, specifically the overheads for failure detection and the actual check-pointing operation, are included in the check-pointing block. Hence even if the check-pointing operation fails due to the burst, our analysis holds since we consider a worst case scenario where the entire block has to be re-executed, and hence there exists sufficient slack to re-execute *just* the check-pointing operation. We assume an error resistant failure detection operation as commonly assumed in the literature [3][8][7][13]– though this may seem to be restrictive, we point out that without this assumption it is impossible to reason about the correctness of the system.

We give an example of how check-pointing improves the FT-feasibility of real-time tasks under temporal redundancy when there are error bursts in the system. Consider a set of

Task	C_i	D_i	T_i
A	2	7	8
B	3	12	12
C	5	24	24

TABLE I
EXAMPLE TASK SET

real-time tasks as shown in table I. As shown in figure 2, in the presence of an error burst of length $T_{length} = 4$, task A will miss its deadline. In order to avoid the deadline miss on task A, check-pointing can be used as shown in figure 3. Assuming a negligible check-pointing overhead, the FT feasibility of a given task set is maximized when placing check-points after every time unit of execution.

D. Execution Time Model

In our approach we assume a linear relationship between execution time and processor speed [14], which can be easily

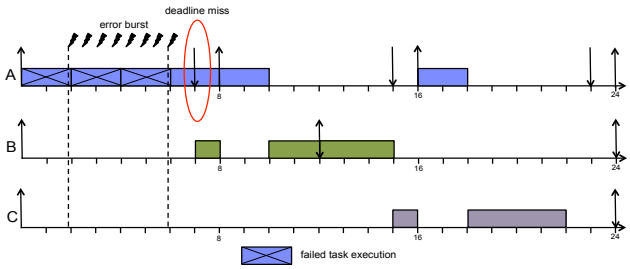

Fig. 2. FT-EDF schedule under faults without check-pointing

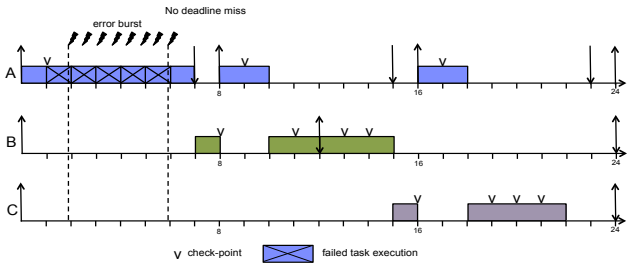

Fig. 3. FT-EDF schedule under faults with check-pointing

relaxed. 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 $S > 1$:

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

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'. Consequently, when a processor of speed 'a' is used, the total time requested by the tasks during the time interval t becomes $\frac{DBF_i(t)}{a}$.

III. The Limits of Temporal Redundancy in Real-time Scheduling

As stated in the problem formulation, the main goal of this paper is to determine the limits on the effectiveness of temporal redundancy based fault tolerant real-time scheduling approaches. We use processor speed-up as a metric to *quantify* the FT-feasibility of real-time tasks under temporal redundancy. In the following, we derive the upper-bound on the lowest processor speed-up that guarantees FT-feasibility when 1) the failed tasks are entirely re-executed and 2) the failed tasks are re-executed from the last saved check-point. The derived speed-up factors respectively represent the upper- and lower-limits of temporal redundancy based fault tolerance schemes for real-time systems. The closer the *measured* FT-feasibility of a task set employing a particular implementation of temporal redundancy to the lower limit, the better that particular implementation is.

A. The upper-limit

In our previous work [5], we derived the resource augmentation bound that enables FT-feasibility of a set of temporally redundant real-time tasks under error bursts of known upper-bounded lengths. We derived the resource augmentation bound under the assumption that the failed tasks have to be entirely re-executed. Consequently, the resource augmentation bound in this case denotes the *upper-bound* of the effectiveness of temporal redundancy approach in guaranteeing the FT-feasibility of real-time tasks under error bursts. We recall the following two theorems from our previous paper [5]:

Theorem III.1. *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}, when every failed task is entirely re-executed, is upper-bounded by:*

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

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

Theorem III.2. *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}, when every failed task is entirely re-executed, 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:*

$$S_b \leq 6$$

Thus the upper-limit on the effectiveness of temporal redundancy under an error burst of length $T_{length} \leq \frac{D_{min}}{2}$ is given by 6 (for other cases it is given by Theorem III.1).

B. The lower-limit

In this section we first present the fault tolerance feasibility analysis of real-time tasks that employs check-pointing strategy under error burst occurrences. We then derive the lower limit on the effectiveness of temporal redundancy. This is done by deriving the resource augmentation bound that guarantees FT-feasibility for the case when every task is check-pointed after every single unit of execution and the check-pointing overhead is negligible.

If the error burst starts ϵ time unit before a particular check-point of a job of τ_i, the execution of the entire check-pointing block that ends at the check-point is *wasted* i.e., no meaningful computations are performed, leading to wasted resources e.g., the processor time.

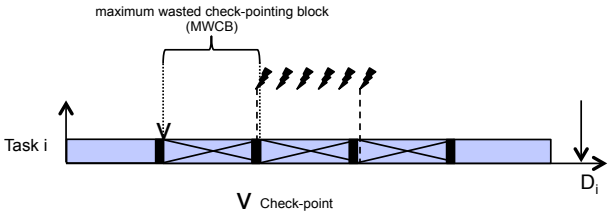

Fig. 4. The Maximum Wasted Check-pointing Block

Definition 1. *The Maximum Wasted Check-pointing Block (MWCB) of a task τ_i hit by an error burst, is defined as the execution time of the largest check-pointing block of τ_i (which is decided by the system designer) that is wasted due to the error burst.*

$$MWCB(\tau_i) = CP_i^S - \epsilon$$

An example of the maximum wasted check-pointing block for a task τ_i is given in figure 4. When the error burst occurs during an arbitrary time interval $[0, t]$ hitting more than one task, each of these tasks may *waste* processor time in executing the failed check-pointing blocks. We define the sum of all the MWCBs of the failed tasks in an interval $[0, t]$ due to the error burst as the Worst Case Temporal Wastage (WCTW).

Definition 2. *The Worst Case Temporal Wastage (WCTW) during a time interval $[0, t]$, denoted by $W_{err}(t)$, is defined as the largest possible processor time wasted due to the execution of MWCBs of all failed check-pointing blocks of the tasks completely scheduled in the interval $[0, t]$.*

We derive the condition for FT-feasibility by first calculating all WCTWs and then accounting for them in the EDF feasibility condition. In the following we describe the method to calculate the WCTW during any arbitrary time interval $[0, t]$. We first present a *necessary condition*, for completeness, and then derive the *sufficient condition* for feasibility.

a) Necessary Condition for FT-Feasibility: The necessary condition is obtained from the length of the error burst.

Lemma III.1. *A necessary condition for the FT-feasibility of a task set $\Gamma = \{\tau_1, \tau_2, ...\tau_n\}$ under an error burst of length T_{length} is,*

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

Proof: Consider a task τ_i that is hit by the burst. To guarantee that τ_i does not miss its deadline, it must be possible to allocate C_i time units to τ_i outside the error burst to enable one full re-execution. Consequently, the error burst length may not exceed $D_i - C_i$.

Hence the largest tolerable length of the error burst is obtained when the error burst starts just before one of the check-pointing blocks finishes (i.e., before the check-pointing operation) and ends when the last (failed) re-execution completes its execution. Under check-pointing, the progress of the task execution is saved at most after every CP_i time units. Hence, the maximum time wasted is $(CP_i - \epsilon)$.

To ensure feasibility of τ_i, $CP_i - \epsilon + T_{length} \leq D_i - C_i$.

$$\Rightarrow T_{length} \leq (D_i - C_i - CP_i) + \epsilon$$

Consequently, when considering the entire task set, the maximum error burst length is given by the maximum T_{length} among all the tasks. ■

b) Sufficient Condition for FT-Feasibility: Our strategy of finding the sufficient condition for the feasibility of check-pointed real-time task sets is as follows. We consider a time instant t' when a job $\tau_{i,j}$ having an absolute deadline at time

t, is executing. We assume that even under the error burst, there are no deadline misses on the jobs in the interval $[t', t]$, and then we derive the sufficient condition for this to be true.

The WCTW at t occurs when all jobs arrive in a strictly periodic manner, because such a scenario maximizes the number of jobs that can be hit by the burst. Assuming that the error burst occurs before the absolute deadline t, we derive the WCTW at t when the error burst hits:

Case 1: No job scheduled in the interval $[t', t]$.
Case 2: Only one job scheduled in the interval $[t', t]$.
Case 3: More than one job scheduled in the interval $[t', t]$.

In cases 2 and 3, *unless stated otherwise*, we assume that $\tau_{i,j}$ is the first job to be hit by the error burst and the error burst starts at the time instant t'.

Observation III.1. *(from [5]) Every job 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.*

Lemma III.2. *(from [5]) No job $\tau_{a,b}$ released in the interval $[t', t]$, having an absolute deadline $> t$, can be hit by the error burst.*

Lemma III.3. *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 $W_{err}(t)$ during the time interval $[0, t]$ is given by:*

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

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 $\tau_{i,j}$. We also assume that no job is released in $[t', t]$ with an absolute deadline less than or equal to t, and hence according to lemma III.2, $\tau_{i,j}$ is the only job that is hit by the error burst. Consequently, $W_{err}(t)$ occurs when the error burst starts just before the largest check-pointing block completes its execution, and ends just after the last failed check-pointing block starts its execution. The proof follows. ■

Lemma III.4. *(from [5]) 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:*

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

where d_{l-1} is the latest absolute deadline (of any other job) that is less than d_l.

Lemma III.5. *If the error burst hits more than one task in the interval $[t', t]$, the contribution of $\tau_{i,j}$ to $W_{err}(t)$ during a time interval $[0, t]$ is $2(CP_i - \epsilon)$.*

Proof: This scenario occurs when the largest check-pointing block 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. Suppose $\tau_{a,b}$ is the task preempting $\tau_{i,j}$. Hence $\tau_{a,b}$ has an earlier absolute deadline than $\tau_{i,j}$. Consequently, under the assumption of an EDF scheduler, the error burst will end before $\tau_{a,b}$ completes one successful execution. When the preempted alternate of $\tau_{i,j}$ resumes its execution, its remaining execution, i.e., $CP_i - \epsilon$ is wasted as it was hit by the error burst just before the preemption. The job $\tau_{i,j}$ then executes successfully as the error burst has already ended. Thus the contribution of $\tau_{i,j}$ to $W_{err}(t)$ at time t is $2(CP_i - \epsilon)$. ■

Lemma III.6. *If the error burst hits more than one task in the interval $[t', t]$, only one check-pointing block belonging to the failed tasks other than τ_i, that is hit by the error burst will contribute to $W_{err}(t)$ during a time interval $[0, t]$.*

Proof: We consider the jobs that are executing in the interval $[t', t]$ since only these jobs need to finish executing before their corresponding absolute deadlines, so that $\tau_{i,j}$ can finish executing by t. Any job with an absolute deadline greater than t will not affect the execution of $\tau_{i,j}$ since we assume EDF.

We have assumed that $\tau_{i,j}$ is the first job to be hit by the error burst. Let $\tau_{a,b}$ that has a release time and deadline in the interval $[t', t]$, be the next job 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 achieve one successful execution before its absolute deadline. The contribution of job $\tau_{a,b}$, since it is hit by the burst, to $W_{err}(t)$ is maximum when either:

1) One of the failed check-pointing blocks of job $\tau_{a,b}$ is immediately preempted by a higher priority job $\tau_{e,f}$ as soon as it starts executing.
   In this case, the error burst will end before $\tau_{e,f}$ completes one successful execution. After $\tau_{e,f}$ completes the remaining executions of the failed check-pointing block of $\tau_{a,b}$, which was preempted, execute to completion. Hence the maximum processor time wasted by $\tau_{a,b}$ is $(CP_a - \epsilon)$, before it can successfully execute (Definition 1).
2) The error burst ends just before the last failed check-pointing block 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 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 a failed check-pointing block of $\tau_{a,b}$. Hence, according to Definition 1, the maximum wasted processor time is $(CP_a - \epsilon)$.

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

Hence only one check-pointing block of tasks other than $\tau_{i,j}$ that is hit by the error burst will contribute to $W_{err}(t)$ at time t, when more than one tasks are hit by the burst. ■

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 III.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:

$$W_{err}(t) = \max_{\forall \tau_m \in \Gamma : D_m = D_i} \{2(CP_m - \epsilon)\} + \sum_{\forall \tau_k \in \Gamma : D_k < D_i} (CP_k - \epsilon)$$

Proof: We showed in lemma III.6 that only one check-pointing block of the failed tasks that have release times and deadlines in $[t', t]$ will contribute to the $W_{err}(t)$. Hence 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 $CP_a - \epsilon$ 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 largest check-pointing block of the preempted task.

The tasks that are potentially released in the interval $[t', t]$ are the tasks having relative deadlines less than or equal to D_i (observation III.1). Hence, following the reasoning in lemma III.6, only one check-pointing block of each of the failed tasks other than $\tau_{i,j}$ contributes to the worst case temporal wastage. The worst case contribution of $\tau_{i,j}$ is $2(CP_i - \epsilon)$ according to lemma III.5. However, since there can be more than one task τ_m with deadline $D_m = D_i$, we take the maximum execution time of all such tasks.. ∎

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 during a time interval $[0, t]$ is given by the following lemma.

Lemma III.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 during a time interval $[0, t]$ is given by:*

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

Proof: According to the observation III.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 τ_k scheduled in the interval $[t', t]$ is hit by the error burst, the maximum contribution to the worst case temporal wastage at t is $2(CP_k - \epsilon)$. Here, τ_k can be either τ_i or, according to observation III.1 and using lemma III.2, any τ_k such that $D_k \leq D_i$. The proof follows. ∎

Theorem III.3. *The worst case temporal wastage $W_{err}(t)$ during a time interval $[0, t]$, 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,

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

$$y = \max_{\forall \tau_m \in \Gamma : D_m = D_i} \{2(CP_m - \epsilon)\} + \sum_{\forall \tau_k \in \Gamma : D_k < D_i} (CP_k - \epsilon)$$

Proof: The proof follows from lemma III.4, III.7, III.8. At deadline d_l, according to lemma III.7, if the error burst hits more tasks in addition to $\tau_{i,j}$,

$$W_{err}(t) = \max_{\forall \tau_m \in \Gamma : D_m = D_i} \{2(CP_m - \epsilon)\} + \sum_{\forall \tau_k \in \Gamma : D_k < D_i} (CP_k - \epsilon)$$

According to lemma III.8, at deadline d_l, if the error burst hits only one task, the $W_{err}(t)$ is given by

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

Finally, according to lemma III.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 III.4, III.7 and III.8. ∎

Definition 3. *The worst case error overhead E_t, during a time interval $[0, t]$, is defined as the sum of the error burst length T_{length} and the worst case temporal overhead in the time interval $[0, t]$.*

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

We build on the demand bound analysis proposed by Baruah et. al [9] and define the sufficient condition for feasibility under an error burst.

Theorem III.4. *(from [5]) A real-time task set Γ is FT-feasible under an error burst of length T_{length} if, for any time interval $[0, t] \forall t = kT_j + D_j, \forall \tau_j \in \Gamma$ and $t \leq LCM$,*

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

Theorem III.5. *(from [5]) 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:*

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

In order to derive the lower-limits on temporal redundancy for FT-feasibility, we bound the $W_{err}(t)$ during a time interval $[0, t]$ when check-pointing is used. Remember that to derive the lower-limits on temporal redundancy we assume that all the tasks are check-pointed after every unit of execution and the check-pointing overheads are zero. Hence:

$$CP_i = 1 \quad \forall \tau_i \in \Gamma$$

Remember that CP_i cannot be less than 1 since it is the smallest atomic unit of execution that can be check-pointed.

Lemma III.9. *The worst case temporal wastage $W_{err}(t)$ during a time interval $[0, t]$, $t \in \{d_1, d_2, ..., d_m\}$ when every task is check-pointed after every unit of execution and assuming negligible check-pointing overhead, is bounded by:*

$$W_{err}(t) \leq (n + 1)(1 - \epsilon)$$

Proof: At deadline d_1, which is the shortest relative deadline D_1, the worst case temporal wastage $W_{err}(t)$, according to theorem III.3, when every task is check-pointed after every unit of execution and the overheads are zero, is given by:

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

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

Therefore,

$$x = 2(1 - \epsilon)$$

Also,

$$y = \max_{\forall \tau_m \in \Gamma : D_m = D_i} \{2(CP_m - \epsilon)\} + \sum_{\forall \tau_k \in \Gamma : D_k < D_i} (CP_k - \epsilon)$$

Since we consider d_1, there are no tasks with relative deadline less than D_1, therefore,

$$y = \max_{\forall \tau_m \in \Gamma : D_m = D_1} \{2(CP_m - \epsilon)\} = 2(1 - \epsilon)$$

Consider any absolute deadline d_l of any task τ_i, $d_l = jT_i + D_i$. The $W_{err}(d_l)$ is given by $W_{err}(d_l) \leq max(x, y, d_{l-1})$. Here again, we can see that x can be bounded by:

$$x = \max_{\forall \tau_k \in \Gamma : D_k \leq D_i} \{2(CP_k - \epsilon)\} = 2(1 - \epsilon)$$

We need to find a bound on y. Remember that

$$y = \max_{\forall \tau_m \in \Gamma : D_m = D_i} \{2(CP_m - \epsilon)\} + \sum_{\forall \tau_k \in \Gamma : D_k < D_i} (CP_k - \epsilon)$$

The value of y is maximum at D_n, which is the largest relative deadline. This happens in the case when the τ_n is hit by the error burst immediately before its largest check-pointing block finishes its execution, and one of the failed check-pointing block is preempted immediately by τ_{n-1}, which is again immediately preempted by τ_{n-2} and so on. In this situation, τ_n leaves $2(1 - \epsilon)$ units of execution outside the burst while every other task leaves approximately $1 - \epsilon$ unit outside the burst. Hence the value of y is:

$$y = 2(1 - \epsilon) + (n - 1)(1 - \epsilon) = (n + 1)(1 - \epsilon)$$

Hence for any t, since $n \geq 1$, $W_{err}(t) \leq (n + 1)(1 - \epsilon)$ ■

Lemma III.10. *The upper-bound on the minimum processor speed-up S_b that guarantees the FT-feasibility of Γ under an error burst of length T_{length} when every task is check-pointed after every unit of execution and assuming negligible check-pointing overhead is*

$$S_b \leq \frac{D_{min}}{D_{min} - T_{length}} + \frac{(n + 1)(1 - \epsilon)}{D_{min} - T_{length}}$$

Proof: According to theorem III.5, 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:

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

The maximum value of $\sum_{i=1}^{n} DBF_i(t)$ is t since we assume uniprocessor feasible task set. Substituting $\sum_{i=1}^{n} DBF_i(t) = t$, we obtain the bound on the speed-up required to guarantee FT-feasibility:

$$S_b = \max_{\forall t} \left\{ \frac{t + W_{err}(t)}{t - T_{length}} \right\}$$

According to lemma III.9,

$$W_{err}(t) \leq (n + 1)(1 - \epsilon)$$

$$\Rightarrow S_b \leq \max_{\forall t} \left\{ \frac{t}{t - T_{length}} + \frac{(n + 1)(1 - \epsilon)}{t - T_{length}} \right\}$$

Since T_{length}, n and ϵ are constants, the largest value of S_b is given by the smallest value of t, which is $t = D_{min}$. Hence,

$$S_b \leq \frac{D_{min}}{D_{min} - T_{length}} + \frac{(n + 1)(1 - \epsilon)}{D_{min} - T_{length}}$$

■

Theorem III.6. *The upper-bound on the minimum processor speed-up S_b that guarantees the FT-feasibility of Γ under an error burst of length $T_{length} \leq \frac{D_{min}}{2}$, when every task is check-pointed after every unit of execution and assuming negligible check-pointing overhead, is given by:*

$$S_b \leq 2 + \frac{2(n + 1)(1 - \epsilon)}{D_{min}}$$

Proof: According to theorem III.10, the bound on the minimum processor speed-up S_b that guarantees the FT-feasibility of Γ under an error burst of length T_{length} when every task is check-pointed after every unit of execution and assuming negligible check-pointing overhead is

$$S_b \leq \frac{D_{min}}{D_{min} - T_{length}} + \frac{(n + 1)(1 - \epsilon)}{D_{min} - T_{length}}$$

Substituting $T_{length} \leq \frac{D_{min}}{2}$ we get

$$\frac{D_{min}}{D_{min} - T_{length}} \leq 2 \tag{1}$$

Similarly when $T_{length} \leq \frac{D_{min}}{2}$ we get

$$\frac{(n + 1)(1 - \epsilon)}{D_{min} - T_{length}} \leq \frac{2(n + 1)(1 - \epsilon)}{D_{min}} \tag{2}$$

The proof follows. ■

Theorem III.7. *The bound on the minimum processor speed-up S_b that maximizes the FT-feasibility of a set of real-time tasks Γ under an error burst of length $T_{length} \leq \frac{D_{min}}{2}$, when every task is check-pointed after every unit of execution and assuming negligible check-pointing overhead, is given by*

$$S_b \leq 2$$

Proof: The speed-up required is minimized if the ratio of a single unit of task execution to its relative deadline is minimized for all tasks. The ratio of a single unit of execution to its relative deadline is the maximum for τ_i which has the shortest relative deadline $D_{min} = D_1$. Hence, the speed-up required is minimized if $\frac{1}{D_{min}}$ is as small as possible.

Consequently, substituting $\frac{1}{D_{min}} \to 0$, in Theorem III.6 we get:

$$S_b \leq 2 + \lim_{\frac{1}{D_{min}} \to 0} \frac{2(n+1)(1-\epsilon)}{D_{min}} = 2$$

■

Thus the lower-limit on the effectiveness of temporal redundancy under an error burst of length $T_{length} \leq \frac{D_{min}}{2}$ is given by 2. This means that, in order to guarantee FT-feasibility of a task set under an error burst of known upper-bounded length $T_{length} \leq \frac{D_{min}}{2}$, the processor speed needs to be increased by a factor of at least 2. The closer the value of FT-feasibility of real-time tasks using a particular implementation of temporal redundancy to 2, the better its efficiency.

IV. Related Work

Dependability is defined as the ability of a system to deliver a justifiably trusted service [1] and is achieved by using appropriate fault tolerance mechanisms. The fault tolerance mechanisms typically involves two stages: error detection and error handling. The commonly used error handling schemes are rollback, roll forward and compensation using redundancy. The most commonly adopted redundancy technique is the temporal redundancy approach [3][5][15][13], which are complemented using check-pointing schemes [6] [8]. Baruah et. al. [9] derived a sufficient and necessary condition for uniprocessor feasibility of real-time tasks. A feasibility analysis of fault tolerant systems under a k-fault scenario was presented in [3] and [13]. Many et. al. [10] considered the Fixed Priority Schedulability of a set of real-time tasks under a fault burst and derived a response time analysis for tasks scheduled under the burst. Aysan et.al. [16] presented a probabilistic schedulability analysis for real-time tasks scheduled using FPS under an error burst. Earlier, they [17] presented a method to maximize the schedulability of mixed criticality real-time tasks and messages using FPS, which was extended to the Controller Area Network (CAN) [15]. Resource augmentation [14] was proposed to quantify the cost for non-clairvoyance in online real-time scheduling. We [19] used resource augmentation to quantify the sub-optimality of non-preemptive scheduling with respect to a uniprocessor optimal scheduling algorithm.

V. Conclusions

In this paper we address the FT feasibility of temporally redundant real-time tasks under error bursts. We use resource augmentation, in particular processor speed-up, to measure the FT-feasibility of temporally redundant real-time tasks, and show that the limits on the effectiveness of temporal redundancy in guaranteeing the FT-feasibility are given by 6 and 2 respectively when the error burst length is no greater than half the shortest deadline. We also derive a sufficient condition for determining the feasibility of check-pointed real-time tasks under an error burst of known upper-bounded length. Our analysis does not assume that the check-pointing operation is resistant to the error burst.

Our approach enables a system designer to assess the efficiency of a particular implementation of temporal redundancy in order to maximize the system's reliability while reducing costs. Future work will include tightening the resource augmentation bounds and extensions to spatial redundancy, and ultimately derive the limits on redundancy based fault tolerant real-time scheduling.

References

[1] 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.

[2] "Used hyundai accent reviews- consumer guide road test evaluation," *http://www.consumerguide.com/hyundai/accent/used/ (last accessed: 14 Jun 2013)*.

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

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

[5] A. Thekkilakattil, R. Dobrin, S. Punnekkat, and H. Aysan, "Resource augmentation for fault-tolerance feasibility of real-time tasks under error bursts," in *The 20th International Conference on Real-Time and Network Systems*, November 2012.

[6] K. Shin, T.-H. Lin, and Y.-H. Lee, "Optimal checkpointing of real-time tasks," *IEEE Transactions on Computers*, 1987.

[7] V. Izosimov, P. Pop, P. Eles, and Z. Peng, "Synthesis of faulttolerant embedded systems with checkpointing and replication," in *In International Workshop on Electronic Design, Test and Applications*, 2006.

[8] S. Punnekkat, A. Burns, and R. Davis, "Analysis of checkpointing for real-time systems," *Real-Time Systems*, vol. 20, 2001.

[9] 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.

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

[11] ARP5583A, "Guide to certification of aircraft in a high-intensity radiated field (HIRF) environment," in *Ae-4 Electromagnetic Environmental Effects (E3) Committee, SAE International, Product Code: ARP5583A, Revision Number: A*, June 2010.

[12] H. Hansson, C. Norstrom, and S. Punnekkat, "A simulation based approach for estimating the reliability of distributed real-time systems," in *The 8th IEEE International Conference on Emerging Technologies and Factory Automation*, 2001.

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] A. Thekkilakattil, R. Dobrin, and S. Punnekkat, "Quantifying the sub-optimality of non-preemptive real-time scheduling," in *The 25th Euromicro Conference on Real-Time Systems*, July 2013.