CHALMERS

Fault-Tolerant Scheduling

A Model Proposal for Multiple Transient Faults

Master of Science Thesis in the Programme Secure & Dependable Computer Systems

XINGXING LIU

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Göteborg, Sweden, June 2009

Fault-Tolerant Scheduling
A Model Proposal for Multiple Transient Faults

Xingxing Liu

Examiner: Jan Jonsson

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden, June 2009

Abstract

In this thesis, we present a model which provides real-time scheduling with fault-tolerance by time redundancy. By using this model, a feasibility test for the EDF schedule of a task set with multiple transient faults is analyzed for different error detection mechanisms including comparison, timer monitor and HW/SW EDMs. A time redundancy mechanism TEM is employed to mask errors caused by transient faults. Moreover, the success probability of a task set is calculated based on schedulability tests for all possible error patterns. In addition, we give example results from the model simulation.

Keywords: *Real-Time System, Fault-Tolerant, Error Pattern, Recovery Pattern, Success Probability, EDF Scheduling, Error Detection Mechanism, NLFT, TEM*

Contents

1 Introduction

Real-time systems are found in numerous embedded systems [1], ranging from household machines, to car brake-by-wire systems, to aircrafts and spaceships. The real-time systems differ from other general systems. In a hard real-time system, if the output does not meet its deadline, catastrophes or very bad things will happen. In a soft real-time system, although there is no catastrophic result when deadlines are missed, the results will be useless or degraded.

Because transient faults are very common in digital products or systems, the computer systems used in the real-time systems must be fault-tolerant [2]. Power fluctuations, electromagnetic interference, and particle radiation all can cause transient faults [3]. Even in the presence of faults, it is desirable that the computer systems still operate correctly and meet deadlines. These systems are called fault tolerant computer systems, which have been used in many critical systems, such as satellite launchers and spacecrafts[4]. When the time is ample for the recovery procedure of faulty tasks, time redundancy can be used to implement fault tolerance. In this case, the fault tolerance is achieved by combining hardware and software error detection mechanisms (HW/SW EDMs) with temporal error masking (TEM).

Schedulability analysis is usually used to measure a schedule for real-time systems. However from the aspect of fault-tolerant, reliability and availability are the measures of the systems. It will be a more complete description if we combine the two aspects together and measure the success probability of a task set by its reliability based on schedulability analysis.

2 Problem Statement

This thesis project is a part of a larger project about real-time fault-tolerate scheduling with multiple transient faults. In previous sub-projects, other people have studied RM scheduling with one fault [5], EDF scheduling with one fault [6], and RM scheduling with multiple transient faults [8]. However, there is no existed model as far as I know which combines real-time scheduling with fault-tolerance. Therefore in this work, we will try to find a model which provides real-time scheduling with fault-tolerance by time redundancy, and give the success probability of a task set with multiple transient faults based on schedulability test on all possible fault patterns. Firstly, a fault model which fits schedulability analysis is given, then we describe how to perform the schedulability test in the model. Lastly, we give two examples by presenting the results from model simulation.

Previous research in real-time scheduling area has done a lot about various kinds of task sets, including periodic tasks , aperiodic tasks, and sporadic tasks, for multiprocessor systems and for uniprocessor systems. Previous models in the area of fault tolerance have concentrated on the time-redundancy and space-redundancy based on different fault detection techniques and fault-tolerant architectures. The performance measure of a real-time system in those studies usually is schedulability analysis based on response-time analysis(RM scheduling), or utilization based analysis(EDF scheduling). However, for fault-tolerate systems, reliability and availability analysis are usually necessary. The main aim of this work is to connect the analysis for real-time systems with the

analysis for fault-tolerant systems.

In this thesis, we will describe a model, which has the properties of real-time task models and fault-tolerant system models. Based on schedulability analysis, we will give the success probability of a task set in a uniprocessor system. If a task set is schedulable with any possible fault pattern caused by f transient faults, it will complete successfully.

Based on error distribution, we search exhaustively for all the possible combinations of erroneous copies, which are called error patterns, and analyze the schedulability of the original task set with each error pattern, and then give the success probability based on the number of error patterns that have passed EDF scheduling test and on the other parameters from Art68-FT experiment [9].

Previous research projects have already studied a single or multiple faults by RM scheduling for uniprocessor systems, and a single fault by EDF scheduling for uniprocessor systems. Therefore in this work, we choose multiple faults by EDF scheduling for uniprocessor systems. We choose processor demand analysis for feasibility test on each possible recovery pattern.

Statistical analysis, as well as multinomial distribution, binomial coefficient or Poisson distribution, is frequently seen in mathematics and thermal physics, but it is not often used in real-time systems. However in this work we will use statistical analysis frequently to compute the task success probability. Nonetheless this work is only a beginning and there is much room left to improve. Hopefully this work will inspire other research in future to advance our theories and models presented hereby.

The rest part of this thesis is organized as the following: Chapter 3 introduces some real-time and fault-tolerant terminologies and defines the terms used in this thesis; Chapter 4 reviews related work in the area; In chapter 5, we present the system model used to analyze fault occurrence probability and effective error probability and so on; Chapter 6 and chapter 7 are other two important parts. They include schedulability analysis and the algorithms to simulate the system model and to analyze the system shedulability; In chapter 8 we give some example results from model simulations. Chapter 9 suggests some further work that can be done; In the last chapter we draw several conclusions.

3 Concepts of Real-time and Fault-tolerate Systems

In this section, we introduce some necessary terminologies used in real-time systems and fault-tolerate systems. They are essential to understand the task model and the fault model to be described in the following chapters of this thesis.

3.1 Real-Time Terminologies

Some real-time terminologies referred in this thesis will be shortly described in the following sub-sections.

3.1.1 Task, instance and copy

In this thesis, we mainly deal with *periodic* and *critical* tasks. A *critical* task is usually processed by a hard real-time system. If a critical task misses its deadline, it will cause a catastrophe[1]. On the other hand, a *noncritical* task is usually handled by a soft real-time system. If a noncritical task misses its deadline, the result will be useless or less useful. A *periodic* task arrives with a specific pattern repeatedly; in contrast, a *sporadic* task arrives with a time interval equal to or longer than a certain period; and for an *aperiodic* task, there is no minimum time interval between two arrivals[1].

Task, instance and copy are frequently referred in this work. A periodic task includes many subsequent arrivals. Each arrival is called an instance of the task. Each task instance may execute several times. Each execution of the instance is called a copy. Hence, a periodic task is composed of many instances, and an instance in turn may have several copies.

3.1.2 Scheduling

For a certain run-time system and a given task set, a scheduling algorithm generates a schedule, which reserves spacial and temporal resources for the task set. Two classic scheduling algorithms are Rate-Monotonic (RM) scheduling and Earliest-Deadline-First (EDF) scheduling, which were addressed by Liu and Layland[10]. RM is a dynamic scheduling algorithm using static priority, and the priority of a task or a task instance is decided by its frequency. A task having a shorter period is assigned a higher priority. EDF is also a dynamic scheduling algorithm, but it uses dynamic priority, and the priority of a task or a task instance is determined by its absolute deadline. The task that has the earliest deadline receives the highest priority.

Pathan studied tolerating multiple transient faults by using RM scheduling[8]. However in this work, we will use EDF scheduling to handle multiple transient faults.

3.2 Fault Tolerance Terminologies

In this section, some fault tolerance terminologies and the methods used in this thesis will be introduced.

3.2.1 Fault, error and failure

"A system *failure* occurs when the service provided by the system deviates from the specified service. An *error* is a perturbation of the internal state of the system that may lead to a failure"[11]. Usually, a failure occurs when the erroneous state causes the whole system not working correctly. An active *fault* will cause an error, otherwise the fault is said to be dormant.

One way to classify faults is according to their existing durations. A *permanent fault* means that the faulty component needs to be repaired or replaced; An *intermittent fault* usually occurs repeatedly at the same location; A *transient fault* only exists for a short period of time and only occurs at specific time or in a specific situation for various reasons [11].

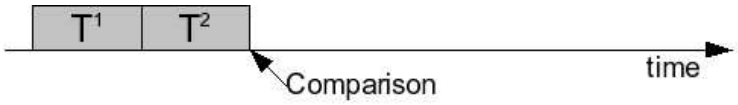

Figure 1: Fault free execution(FF).

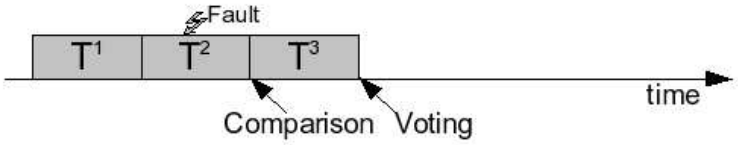

Figure 2: Error detected by the comparison between the two primary copies.

Since transient faults have a much larger occurrence probability than other kinds of faults [12], and only transient faults are most likely to be fault-tolerated by time redundancy, the aim of this work is to tolerate transient faults by time redundancy and to improve the success probability of the whole task set.

3.2.2 Error detection and temporal error masking

Temporal Error Masking(TEM) is a mechanism proposed by Aidemark *et al.*[13] [11] to tolerate faults using time redundancy and the comparison of results at the node level. By TEM, each critical task instance executes twice, and the results are compared to detect errors, or errors are detected by other mechanisms. According to the result of comparison or error detection, an extra execution maybe launched or not. In this way, the success probability of the system is increased. The four different error detection and error masking cases discussed in [13] [11] as examples of TEM are:

1. Fault free execution(FF)
   As mentioned above, in TEM, each task instance executes twice at first. We call these two executions *primary copies*. If the results of the two primary copies match, a third copy, which is also called the *recovery copy*, does not need to execute, and then the next task instance can start immediately. This case is illustrated in Figure 1.

2. Error detected by the comparison between the two primary copies
   After the execution of the two primary copies, their results are compared to detect errors. If their results do not match, a third copy will execute. The results of the three copies are then again checked by a majority voting. If any two results match, they are accepted as the correct results of the task instance. Otherwise, no result will be delivered, and an omission failure occurs. This case is shown by Figure 2.

3. Error detected by hardware(HW)/software(SW) EDMs
   By using hardware and software EDMs, errors can be detected before the two primary copies finish their execution. In this scenario, the defected copy is aborted and a recovery copy starts immediately, as shown in Figure

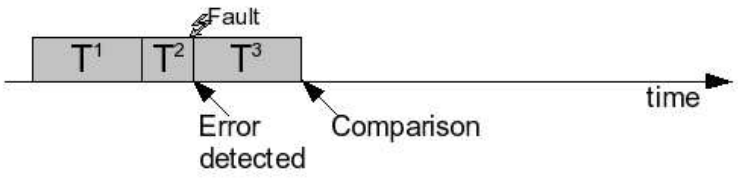

Figure 3: Error detected by HW/SW EDM.

3. Thus, the recovery copy can use the time previously claimed by the faulty copy, and the avaliable slack for recovery procedure increases.

4. Error detected by timer monitor
   A timing error is detected by a timer monitor. Aidemark *et al.* suppose that a timing error happens when the delivery time of a task copy differs from that of a fault free run by more than $250\mu s$ [13] . In this thesis, we instead define the WCET of a task instance as the time-out value for timer monitor. If the delivery time of a task copy exceeds its WCET, the current execution is aborted and it is considered as a fault occurrence, resulting in a similar recovery action as when an error was detected by HW/SW EDMs, while it has the same execution manner as when an error was detected by comparison from the aspect of execution time.

In this thesis, TEM is used to tolerate faults during task execution. During the execution of the two primary copies, errors can be detected by comparison, HW/SW EDMs, or timer monitor. If there is no error detected, the task instance completes successfully. Otherwise, a recovery copy runs and tries to tolerate errors. In references [13] [11], at most three copies of a task instance can execute. In the current work, one task instance can execute many copies until its deadline is reached. When an instance has enough time to execute, it is possible that the 3 error-including situations out of the four cases listed above (excluding the fault free one) all can be found in its execution. In our resolution, after the third copy of a task instance finished, a voting or a comparison is taken. If there is no matching result, a fourth copy, which is also called the second recovery copy, is attached. If an error is detected in the fourth copy by HW/SW EDMs or timer monitor, the fifth copy is dispatched, otherwise a comparison or voting is taken after the fourth copy execution. If there is no matching between any two results, an additional copy runs again until reaching its absolute deadline. If there is still no match between any two copies of the task instance, an omission failure occurs.

3.2.3 Fault injection and error detection

The main purpose of using fault injection is to find out the weakness of a fault-tolerate system, as well as to test the fault tolerant function of the system. A number of studies provide various fault injection techniques by using different fault injection tools and methods[14]. Software implemented fault injection(SWIFI) is one way to inject faults.

In this thesis, we use the fault injection results from [13] by Aidemark *et al.*, in which SWIFI with GOOFI tool [15] was used to inject faults to a trap handler and to a trace handler routine located in the system memory. After a fault is injected to the system, Motorola MC68340 micro-controller and a program routine written in programming language Ada95 detect errors. Furthermore, we use both the fault injection and the error detection results from the experiments by Artk68-FT to determine the parameters in calculating the success probability of a task set. The results from Artk68-FT are shown in table 11 in Appendix . If one uses different experiment results, which depend on the chosen computer system and the fault injection mechanism, he may get a different success probability.

Table 1 shows the error detection mechanisms used in the experiment Artk68-FT. Table 2 gives the chosen parameters in the fault injection, error detection and error masking experiments. Those values will be used in our further probability computing.

Selection of Ada run-time constraint checks	
Ada access check	Attempt to follow a null pointer
Ada range check	Attempt to violate a range constraint of scalar value
Ada index check	Attempt to access an index that is not in the range of the array
Selection of motorola 68340 microcontroller hardware checks	
Bus error	Attempt to access non-existing memory
Address error	Attempt to access a word or a long-word at an odd memory address
Illegal instruction	Attempt to execute a non-existing instruction
Line1010	Attempt to execute an unimplemented instruction
Division by zero	Raised if a division instruction gives a divisor value of 0

Table 1: Error detection mechanisms

3.2.4 Node-level fault tolerance (NLFT) techniques

In [11], Node-Level Fault Tolerance (NLFT) techniques were introduced, including hardware and software based techniques. As to software-implemented fault tolerance at the Node-Level, error masking or error recovery techniques may be adopted. In this work, we choose TEM to mask errors. For each task instance, the voting may execute several times to mask errors.

4 Related Work

Here we only briefly mention the works that are closely or directly related to the present work. More literature about fault-tolerant real-time scheduling research can be found in Pathan and Tengdahl [8, 6] and in references therein.

Time redundancy has previously been studied in real time systems. Aidemark *et al.* investigated many projects and experiments about NLFT and time

P_x	Given that a fault occurs, an error is generated
P_{DE}	Given that an error is generated, the error is detected by comparison after double execution (DE)
P_T	Given that an error is generated, the error is detected by a timer monitor (TM)
P_{ED}	Given that an error is generated, the error is detected by a hardware error detection mechanism (EDM)
P_{ND}	Given that an error is generated, the error is not detected
$P_{DE,M}$	Given that an error is detected by DE, the error is masked by TEM
$P_{T,M}$	Given that an error is detected by TM, the error is masked by TEM
$P_{ED,M}$	Given that an error is detected by EDM, the error is masked by TEM

Table 2: Probability parameters

redundancy from fault-tolerant aspect [16, 11, 13, 15, 3, 9]. The goal of NLFT is to tolerate the majority of the transient faults at the computer Node-Level in order to reduce the probability of node-failure. TEM is a time redundancy technique, which achieves fault tolerance through recovery, re-execution and voting. In this work, we use the results from Akr68-FT experiment, in which NLFT and TEM were used to detect and mask errors.

From the viewpoint of real-time fault-tolerant scheduling, many researches focused on multiprocessor systems. In [17, 18, 19], space redundancy is a common idea when multiprocessor computer systems were concerned. All those studies talked about periodic tasks, and the way to achieve fault tolerance by double executions. No error detection and error masking techniques were involved in those studies.

Reference [20] gave a fault-tolerant scheduling scheme for periodic and aperiodic tasks in distributed real-time systems by time redundancy and space redundancy. Double executions were employed to achieve fault tolerance, yet it was only about double execution, without considering error detection and recovery.

Aydin *et al.* proposed an optimal scheduling for imprecise computation tasks in the presence of multiple faults in uniprocessor systems [21]. In their study, a task is divided into two parts, a mandatory part (like critical task) and an optional part (like non-critical task). Algorithm FT_optimal can tolerate up to k faults without missing any deadline in the mandatory part by using the slack from the optional part of the task.

In ref. [7], Pathan presented a fault-tolerant real-time algorithm RM-FT, based on NLFT using TEM technique with RM scheduling to find out the upper bound of a slack time for possible executions with a fixed number of errors.

In another study, multiple transient faults can be tolerated in aperiodic tasks of hard real-time systems with EDF scheduling for uniprocessor systems [22]. In this study, the recovery time of each task is fixed and the model section only concerned the task model. A recovery task is treated as a high priority sporadic task with a fixed pre-known execution time. Only the worst case of

recovery patterns was considered. It was also assumed that faults never occur in recovery procedures. Two algorithms were given. Algorithm "Exact" was used to determine the tasks that can be recovered from a certain fault pattern, whereas algorithm "Sufficient" shows whether an aperiodic task set can recover from the worst case of fault patterns. It is a topic closely related to the second half of this thesis. However, we will include all possible fault patterns instead of a single specific fault pattern or the worst case fault pattern. In other words, we will consider all possible fault patterns for periodic tasks to generate an integrated more precise view.

In ref. [5], TEM was employed by Lou to tolerate a single fault in one planning cycle by RM scheduling in uniprocessor systems. The purpose of the study was to give the success probability of a task set with a single fault. Response time analysis was used in the feasibility test of a task set with each possible fault pattern.

Lou's job was extend by Pathan to tolerate multiple transient faults in uniprocessor systems [8]. The author focused on the scheduling analysis on the worst case of fault patterns. In the RM-FT-Any algorithm, the result of feasibility test was obtained from the worst recovery case of each task instance. It simplified the rather complicated fault pattern analysis by paying the price of wasting a significant fraction of CPU resource. In the Recovery-Min-EDM algorithm, the author prescribed a method to check whether the execution time of a task is reducible based on HW/SW EDMs techniques.

Tengdahl studied tolerating a single transient fault in real-time systems with EDF_scheduling [6]. He also presented the minD_recovery algorithm with a scaling factor λ to calculate the minimum deadline of a recovery task copy. Processor demand analysis was used to test the feasibility of the preemptive EDF schedule for a task set with every possible fault patterns.

5 System Model

In this section, the task and fault models to be used in this thesis for scheduling analysis and for success probability analysis are described. System restrictions and some assumptions are also introduced in this section.

5.1 Task Model

A uniprocessor computer system and a task set $\mathcal{T} = \{\tau_1, ..., \tau_n\}$ including n periodic tasks are considered in this work. Each task $\tau_i (1 \leq i \leq n)$ has a Worst-Case-Execution-Time (WCET) C_i, a period T_i, and a relative deadline D_i. To simplify the problem, $D_i \leq T_i$ is assumed.

Some assumptions are made for the task set:

- All tasks are released at the same time.
- All tasks are independent to each other.
- Tasks are preemptive.
- All the tasks in the task set are periodic.

- Any task relative deadline is equal to or less than the task period.

A planning cycle (PC) is the least common multiple (LCM) of the task periods, that is , $PC = LCM\{T_1, T_2, ...T_n\}$. Since all planning cycles are identical, we only need to consider the first planning cycle.

Without loss of generality, we assume all tasks are released at time 0. For each task $\tau_i(1 <= i <= n)$, $\frac{LCM}{T_i}$ task instances are executed in one planning cycle, thus, there are in total $\sum_{i=1}^{n} \frac{LCM}{T_i}$ instances in one planning cycle. In our further analysis, we define N= $\sum_{i=1}^{n} \frac{LCM}{T_i}$.

For a task set in which each task has $D_i = T_i$, the sufficient and necessary condition for its EDF schedulable is that the total utilization of tasks is less or equal to one, $U_{total} = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$. In this work, we choose a more general case $D_i \leq T_i$, thus the feasibility of a schedule is verified by processor-demand analysis. Furthermore, to achieve fault-tolerance, all critical task instances will execute twice. If an error is detected by comparison, by timer monitor, or by HW/SW EDM, a Temporal Error Masking (TEM) is employed to re-execute the faulty task copy.

By the simulation of preemptive EDF scheduling, an instance set $\mathcal{T}' = \{\tau'_1, \tau'_2, ..., \tau'_N\}(N = \sum_{i=1}^{n} \frac{LCM}{T_i})$ is collected from the first planning cycle. Each instance $\tau'_j(1 <= j <= N)$ is a tuple of d'_j, C'_j, P_j, corresponding to its absolute deadline, its WCET, and the fault occurrence rate for each copy of instance τ'_j. And we use $\tau'_{j,i}$ to label the i-th copy of the task instance τ'_j. The priority of its recovery copy is decided by the absolute deadline of task τ'_j, that is , d'_j.

5.2 Fault Model

In previous research [5, 8, 6], uniform fault distribution and at most f fault occurrences in one LCM period were considered. No fault propagation is assumed, that is , one error is only caused by one fault, and the fault only affects the executing task. Since the same transient fault can trigger many different types of errors randomly, we assume all errors detected are different [8]. With those assumptions, the probability for a fault occurrence in executing task τ_i is [5], ($D_i = T_i$ is assumed)

$$P(U_i) = \frac{2 \times C_i}{T_i} = \frac{2 \times C_i}{D_i}$$

However in this thesis, we consider the situation in which there are *on average* f fault occurrences in one planning cycle, since in nature we usually know *on average* how many faults will happen in a specific period, but do not know how many faults exactly will happen in a period. We consider the fault generation that follows the Poisson process. By Poisson process [23, 24], we assume faults do not occur simultaneously. Because the process is memoryless, the number of faults occurred in the previous instance would not affect the number of fault occurrences in the next instance. This property can facilitate our analysis on the probability for each fault combination to appear.

If there is *on average* one fault in LCM, the expected number of faults in

the lifetime duration of a copy of instance τ'_j is

$$P_j = \frac{C'_j}{LCM}$$

According to the Poisson distribution, the possibility for generating k faults in the time period C'_j is,

$$P_{Poisson}(k) = \frac{P_j^k}{k!} e^{-P_j} \tag{1}$$

For $k = 0$, there is no fault generated, which means the system did not encounter any fault, and of course the system is still reliable. The possibility for this case can be easily obtained by letting $k = 0$ in the Poisson distribution function,

$$P_{Poisson}(0) = \frac{P_j^0}{0!} e^{-P_j} = e^{-P_j} \tag{2}$$

Thus its reliability, the probability of 0 fault occurrence is $e^{-\frac{C'_j}{LCM}} = e^{-P_j}$, where C'_j can be understood as the time duration and $\frac{1}{LCM}$ as the faulty rate.

5.2.1 Multinomial distribution or Poisson distribution

Multinomial distribution is discussed in [26, 25]. For a sequence of N statistically equivalent "trials", which comprises v different outcomes, the probability for the i-th outcome is labeled by P_i. Thus, in N trials, the probability for outcome 1 occurring n_1 times, outcome 2 occurring n_2 times, outcome i occurring n_i times, and so on, (up to $i = v$) is:

$$P_{n_1 n_2 \ldots n_v}(N) = \frac{N!}{n_1! n_2! \ldots n_v!} \times P_1^{n_1} P_2^{n_2} \ldots P_v^{n_v}$$

Where

$$\sum_{i=1}^{v} P_i = 1 \; and \; \sum_{i=1}^{v} n_i = N$$

and also

$$\sum P_{n_1 n_2 \ldots n_v}(N) = (\sum_{i=1}^{v} P_i)^N = 1.$$

Based on the theorem of Multinomial distribution, the problem of distributing k errors to N task instances, $\mathcal{T} = \{\tau'_1, \ldots, \tau'_N\}$, seems equivalent to a sequence of k "trials" including N different outcomes.

However, if we also consider the contribution from recovery copies, the errors can also be in the recovery copies, and a recovery copy is only needed when an error is detected. Thus, distributing k errors to N task instances is no longer exactly a simple multinomial distribution problem, since the number of recovery copies depends on the results of previous execution, and at most one error can be detected in one task copy.

Moreover, if we know that there are *exactly* f faults in the planning cycle LCM, we must observe this constraint when we derive the probabilities for each fault combination to appear. It is a dependable process because the total number of faults in the LCM cycle is *exactly* f, including the faults happened

in the execution time and the faults happened in the idle time if there is any. Analyzing the probabilities for fault combinations to appear in this case seems to be cumbersome and less significant, while it can lead to a separate project.

In this work, the property of Poisson process is applied to obtain the weights of all possible error patterns, or the probabilities for those patterns to appear. Then their schedulabilities are checked to see if they are schedulable or how much is the chance to be scheduable. If an error pattern is scheduable, it means that the erroneous copies in the error pattern can be recovered before the deadlines and the pattern contributes to the success probability.

In statistics and probability theory, Poisson distribution states the probability for a given number of events to occurre in a fixed period of time. These events have a known *average* happening rate and they occur independently along the time [27]. Based on Poisson distribution, we assume on *average* f faults in one planning cycle. For each copy, the error free probability can be obtained by considering the zero error probability in its execution time.

5.2.2 Effective errors and non-effective errors

Effective errors and non-effective errors are discussed in [3]. Effective errors are the errors detected by various error detection mechanisms or escaped from the mechanisms, whereas non-effective errors are the ones when faults are not activated. Non-effective errors do not affect the system behavior. Non-effective errors are around 80% of the injected faults for each different kernel function [3]. Since non-effective errors are not real errors, in the following sections when we talk about errors, we mean effective errors.

To facilitate our discussion, here we introduce a few terminologies. The faults caused effective errors are called activated faults, and the faults caused non-effective errors are called inactivated faults.

If a fault occurs during the execution of a task copy, the recovery will be triggered if and only if an effective error is generated by the fault. Suppose that a task copy catches 2 transient faults, and the 1st fault leads to a non-effective error, at most one error will be detected in this copy. Thus, a faulty task copy can have more than one fault, while only the last fault generates an effective error.

Table 3 includes all the situations can happen when one task instance gets two transient faults. Here c stands for a copy to be correct; f means the copy has an activated fault; f' represents the copy has an inactivated fault, and NE means the copy non-existed. The first row through the 3rd row include the three different ways to distribute two activated faults to instance copies. From the 4th row to the 11th row there are 8 different ways to distribute one activated fault and one inactivated fault to the instance. The last three rows show the three different methods to distribute two inactivated faults to the instance. The first, the 4th, and the 5th rows are the same fault copy but they result in different error copies. On the other hand, the 4th, the 6th, and the 11th rows represent different fault copies but they result in the same error copy. It seems hard to analyze our fault model by starting from different fault copy and error copy combinations.

On the other hand, only erroneous copies need to be recovered. How many recovery copies required depends on the number of erroneous copies, but not on the number of faulty copies. So we can only analyze erroneous copies instead of

considering all the faulty copies.

No.	$\tau'_{1,1}$	$\tau'_{1,2}$	$\tau'_{1,3}$	$\tau'_{1,4}$	fault copies	error copies
1	f	f	c	c	$\tau'_{1,1}, \tau'_{1,2}$	$\tau'_{1,1}, \tau'_{1,2}$
2	f	c	f	c	$\tau'_{1,1}, \tau'_{1,3}$	$\tau'_{1,1}, \tau'_{1,3}$
3	c	f	f	c	$\tau'_{1,3}, \tau'_{1,2}$	$\tau'_{1,3}, \tau'_{1,2}$
4	f	f'	c	NE	$\tau'_{1,1}, \tau'_{1,2}$	$\tau'_{1,1}$
5	f'	f	c	NE	$\tau'_{1,1}, \tau'_{1,2}$	$\tau'_{1,2}$
6	f	c	f'	NE	$\tau'_{1,1}, \tau'_{1,3}$	$\tau'_{1,1}$
7	f'	c	f	NE	$\tau'_{1,1}, \tau'_{1,3}$	$\tau'_{1,3}$
8	c	f	f'	NE	$\tau'_{1,3}, \tau'_{1,2}$	$\tau'_{1,2}$
9	c	f'	f	NE	$\tau'_{1,3}, \tau'_{1,2}$	$\tau'_{1,3}$
10	c	f',f	c	NE	$\tau'_{1,2}$	$\tau'_{1,2}$
11	f',f	c	c	NE	$\tau'_{1,1}$	$\tau'_{1,1}$
12	f'	f'	NE	NE	$\tau'_{1,1}, \tau'_{1,2}$	
13	f'f'	c	NE	NE	$\tau'_{1,1}$	
14	c	f'f'	NE	NE	$\tau'_{1,2}$	

Table 3: One task instance with two transient faults . (c-correct, f-fault causing an effective error, f'- fault causing a non-effective error, NE-Not Existed.)

As talked previously, if there is *on average* one fault in LCM, each copy of the instance τ'_j has probability $e^{-\frac{C'_j}{LCM}} = e^{-P_j}$ to be fault free. Then, if there are on average f faults in one planning cycle, each copy of instance τ'_j has the fault free probability $e^{-\frac{fC'_j}{LCM}} = e^{-fP_j}$.

Moreover, if each fault has a possibility P_x to generate an effective error based on Aidemark's Akr68-FT experiment [9], the average number of errors is $P_x f$. Thus, each copy of instance τ'_j has probability $e^{-\frac{P_x f C'_j}{LCM}} = e^{-P_x f P_j}$ to be error free, which we call error free execution probability in this work. On the other hand, each copy of instance τ'_j has probability $1-e^{-P_x f P_j}$ to be erroneous, which we call error execution probability.

5.2.3 Error patterns and recovery patterns

An *error pattern* is a set that includes all erroneous task copies. For example, the error pattern $\{\tau'_{1,1}, \tau'_{2,1}, \tau'_{2,3}\}$ represents that the first copy of task instance τ'_1, the first copy of task instance τ'_2, and the third copy of task instance τ'_2 are erroneous in a specific planning cycle. Each error pattern corresponds to a *recovery pattern* that describes which task instances would run extra copies besides primary copies. For example, the recovery pattern for the error pattern $\{\tau'_{1,1}, \tau'_{2,1}, \tau'_{2,3}\}$ is $\{\tau'_1, \tau'_2, \tau'_2\}$. From a recovery pattern, it is easy to find out which task instance needs recovery copies, and how many recovery copies of that instance will run. An error pattern addresses the exact erroneous copies of task instances, whereas a recovery pattern only includes the task instances referred by the erroneous copies in the error pattern. Hence, several different error patterns may have the same recovery pattern.

5.2.4 The ways to count error patterns and recovery patterns

We limit our consideration to that one fault can cause at most one error. Here we show that one task copy can only have at most one detected error. Suppose a task copy got two faults. If an error was generated when the first fault occurred, the current execution will abort, thus there is no chance for the second fault occurrence. Otherwise, if the first fault did not generate an error, it cannot be detected. If a fault does not generate an error, the fault is said to be inactivated. Although the non-generated error is called non-effective error, it actually does not exist, and of course cannot be detected. In both cases, the number of detected errors is not more than one.

In our fault model, if there are up to two errors in a task instance, at most four copies of that instance can run, two primary copies and two recovery copies. Let the names of each copy to be $\tau'_{j,1}$,$\tau'_{j,2}$,$\tau'_{j,3}$, $\tau'_{j,4}$. An error pattern can be $\{\tau'_{j,1}$,$\tau'_{j,2}$ $\}$, but cannot be $\{\tau'_{j,2}$,$\tau'_{j,4}\}$ because there are already two correct copies, $\tau'_{j,1}$ and $\tau'_{j,3}$, prior to $\tau'_{j,4}$, therefore $\tau'_{j,4}$ would not run.

If three errors in one planning cycle is assumed, at most three recovery copies could run. All possible situations are shown in Table 4. All the instance copies detected to be wrong are included in the error patterns. Recovery patterns are shown in the rightmost column. Here, c represents a correct task copy, e stands for an erroneous task copy, and NE for the task copy not existed according to our error detection mechanisms. For this example, there are 9 error patterns but only 3 recovery patterns.

$\tau'_{1,1}$	$\tau'_{1,2}$	$\tau'_{1,3}$	$\tau'_{1,4}$	$\tau'_{1,5}$	possible copies in error pattern	possible instances in recovery pattern
e	e	e	c	c	$\tau'_{1,1}$,$\tau'_{1,2}$,$\tau'_{1,3}$	τ'_1,τ'_1,τ'_1
e	e	c	e	c	$\tau'_{1,1}$,$\tau'_{1,2}$,$\tau'_{1,4}$	τ'_1,τ'_1,τ'_1
e	c	e	e	c	$\tau'_{1,1}$,$\tau'_{1,3}$,$\tau'_{1,4}$	τ'_1,τ'_1,τ'_1
c	e	e	e	c	$\tau'_{1,2}$,$\tau'_{1,3}$,$\tau'_{1,4}$	τ'_1,τ'_1,τ'_1
e	e	c	c	NE	$\tau'_{1,1}$,$\tau'_{1,2}$	τ'_1,τ'_1
c	e	e	c	NE	$\tau'_{1,2}$,$\tau'_{1,3}$	τ'_1,τ'_1
e	c	e	c	NE	$\tau'_{1,1}$,$\tau'_{1,3}$	τ'_1,τ'_1
e	c	c	NE	NE	$\tau'_{1,1}$	τ'_1
c	e	c	NE	NE	$\tau'_{1,2}$	τ'_1
c	c	NE	NE	NE		

Table 4: One task instance with at most three errors. c-correct, e-erroneous, NE-Not Existed.

Table 5 gives all the error patterns and the corresponding recovery patterns that can be generated by three task instances with two errors. From the table, We can see that there are 21 error patterns, but only 6 different recovery patterns.

5.2.5 The number of error patterns and binomial coefficients

The number of τ'_i included in a recovery pattern equals to the number of errors detected in τ'_i. Hence, if the number of erroneous copies of τ'_i is k_i, the number

$\tau'_{1,1}$	$\tau'_{1,2}$	$\tau'_{1,3}$	$\tau'_{1,4}$	$\tau'_{2,1}$	$\tau'_{2,2}$	$\tau'_{2,3}$	$\tau'_{2,4}$	$\tau'_{3,1}$	$\tau'_{3,2}$	$\tau'_{3,3}$	$\tau'_{3,4}$	error pattern	recovery pattern
e	e	c	c	c	c	NE	NE	c	c	NE	NE	$\tau'_{1,1},\tau'_{1,2}$	τ'_1,τ'_1
e	c	e	c	c	c	NE	NE	c	c	NE	NE	$\tau'_{1,1},\tau'_{1,3}$	τ'_1,τ'_1
c	e	e	c	c	c	NE	NE	c	c	NE	NE	$\tau'_{1,2},\tau'_{1,3}$	τ'_1,τ'_1
c	c	NE	NE	e	e	c	c	c	c	NE	NE	$\tau'_{2,1},\tau'_{2,2}$	τ'_2,τ'_2
c	c	NE	NE	e	c	e	c	c	c	NE	NE	$\tau'_{2,1},\tau'_{2,3}$	τ'_2,τ'_2
c	c	NE	NE	c	e	e	c	c	c	NE	NE	$\tau'_{2,2},\tau'_{2,3}$	τ'_2,τ'_2
c	c	NE	NE	c	c	NE	NE	e	e	c	c	$\tau'_{3,1},\tau'_{3,2}$	τ'_3,τ'_3
c	c	NE	NE	c	c	NE	NE	e	c	e	c	$\tau'_{3,1},\tau'_{3,3}$	τ'_3,τ'_3
c	c	NE	NE	c	c	NE	NE	c	e	e	c	$\tau'_{3,2},\tau'_{3,3}$	τ'_3,τ'_3
e	c	c	NE	e	c	c	NE	c	c	NE	NE	$\tau'_{1,1},\tau'_{2,1}$	τ'_1,τ'_2
e	c	c	NE	c	e	c	NE	c	c	NE	NE	$\tau'_{1,1},\tau'_{2,2}$	τ'_1,τ'_2
c	e	c	NE	e	c	c	NE	c	c	NE	NE	$\tau'_{1,2},\tau'_{2,1}$	τ'_1,τ'_2
c	e	c	NE	c	e	c	NE	c	c	NE	NE	$\tau'_{1,2},\tau'_{2,2}$	τ'_1,τ'_2
e	c	c	NE	c	c	NE	NE	e	c	c	NE	$\tau'_{1,1},\tau'_{3,1}$	τ'_1,τ'_3
e	c	c	NE	c	c	NE	NE	c	e	c	NE	$\tau'_{1,1},\tau'_{3,2}$	τ'_1,τ'_3
c	e	c	NE	c	c	NE	NE	e	c	c	NE	$\tau'_{1,2},\tau'_{3,1}$	τ'_1,τ'_3
c	e	c	NE	c	c	NE	NE	c	e	c	NE	$\tau'_{1,2},\tau'_{3,2}$	τ'_1,τ'_3
c	c	NE	NE	e	c	c	NE	e	c	c	NE	$\tau'_{2,1},\tau'_{3,1}$	τ'_2,τ'_3
c	c	NE	NE	e	c	c	NE	c	e	c	NE	$\tau'_{2,1},\tau'_{3,2}$	τ'_2,τ'_3
c	c	NE	NE	c	e	c	NE	e	c	c	NE	$\tau'_{2,2},\tau'_{3,1}$	τ'_2,τ'_3
c	c	NE	NE	c	e	c	NE	c	e	c	NE	$\tau'_{2,2},\tau'_{3,2}$	τ'_2,τ'_3

Table 5: All the possible ways for distributing two errors in three task instances, and the corresponding error patterns and recovery patterns.(c-Correct, e-erroneous, NE-Not existed)

of the task instance $\tau_i'(1 \leq i \leq N)$ appeared in the corresponding recovery pattern is also k_i.

Let $\tau_{i,last}'$ to be the last copy of the task instance τ_i'. If two primary copies are both correct, $\tau_{i,last}'$ is $\tau_{i,2}'$. If only one error was detected in τ_i', $\tau_{i,last}'$ will be $\tau_{i,3}'$. Since two correct copies are enough, $\tau_{i,last}'$ is always the second correct copy of the task instance τ_i'.

Thus, if k_i errors are detected in τ_i', in total of $k_i + 2$ copies of τ_i' would run. Since $\tau_{i,last}'$, which is τ_{i,k_i+2}', should be correct, all k_i faults must distribute in other $k_i + 1$ copies. The number of ways for distributing k_i faults in $k_i + 1$ task copies is a binomial coefficient problem.

Binomial coefficient from mathworld [28], " in combinatorics, $\binom{n}{k}$is interpreted as the number of k-element subsets (the k-combinations) of an n-element set. It is the number of ways choosing k elements from a set of n elements.

$$\binom{n}{k} = \frac{n!}{k! \times (n-k)!} \tag{3}$$

From equation (3) , the ways of distributing k_i errors in $k_i + 1$ task copies of a single task instance τ_i' is:

$$\binom{k_i+1}{k_i} = \frac{(k_i+1)!}{k_i! \times (k_i+1-k_i)!} = k_i + 1$$

Next we consider two task instances, τ_1' and τ_2'. If k_1 errors are detected in τ_1', and k_2 errors are detected in τ_2', the number of possible ways to distribute k_1 faults in τ_1' and k_2 faults in τ_2' is $\binom{k_1+1}{k_1} \times \binom{k_2+1}{k_2} = (k_1+1) \times (k_2+1)$.

Similarly, in the model of this work, there are N task instances $(\tau_1', \tau_2', \ldots, \tau_N')$. Assuming the corresponding error occurrences are $k_1^h, k_2^h, \ldots, k_N^h$ $(k_j^h \geq 0)$ in a specific recovery pattern, recovery(h), the number of total ways of error distribution is:

$$parameter(h) = (k_1^h + 1) \times (k_2^h + 1) \times \ldots \times (k_N^h + 1) = \prod_{j=1}^{N} (k_j^h + 1) \tag{4}$$

Furthermore, for a total of k errors distributing in N task instances, we need all possible values of $k_1^h, k_2^h, \ldots, k_N^h$, thus the number of different error patterns is,

$$F(k, N) = \sum_{k_1^h, k_2^h, \ldots, k_N^h} (k_1^h + 1)(k_2^h + 1) \ldots (k_N^h + 1) \tag{5}$$

The summation is over all the possible $k_i^h \geq 0$ under the constraint $k_1^h + k_2^h + \ldots + k_N^h = k$.

For 2 errors distributing in 3 instances, table 5 gives the exhaustive searching result of error patterns. The number of error patterns is 21. This number can be checked by equation (5). Let us compute the number of error patterns for the example using equation (5) as shown in table 6. That is: $F(2,3) = \sum_{k_1,k_2,k_3} (k_1+1)(k_2+1)(k_3+1) = 21$.

k_1	k_2	k_3	number of error patterns
2	0	0	$(k_1+1)(k_2+1)(k_3+1)=3$
0	2	0	$(k_1+1)(k_2+1)(k_3+1)=3$
0	0	2	$(k_1+1)(k_2+1)(k_3+1)=3$
1	1	0	$(k_1+1)(k_2+1)(k_3+1)=4$
1	0	1	$(k_1+1)(k_2+1)(k_3+1)=4$
0	1	1	$(k_1+1)(k_2+1)(k_3+1)=4$
			$total: 21$

Table 6: Computing the number of error patterns for 2 errors distributing into 3 task instances

5.2.6 The number of recovery patterns

The number of recovery patterns depends on the number of combinations of k_i, for which $k_i \geq 0$ and $(k_1 + k_2 + \ldots + k_N = k)$. Let R($k$,N) to be the number of ways for distributing k errors to N task instances, thus R(k,N) is also the number of recovery patterns for k error occurrences in N task instances. It is clear that R(k,1)=1. When N=2, suppose the first task gets $m(0 \leq m \leq k)$ errors, then the second task gets $(k-m)$ errors, thus R(k,2)=k +1. For further analysis, we write

$$R(k,2) = k+1 = \frac{(k+1)!}{k!1!}.$$

And also, the number of ways for distributing k errors to 2 task instances, is the sum of all the possible ways, from distributing 0 error to 1 instance to distributing k errors to 1 instance, that is:

$$R(k,2) = R(0,1) + R(1,1) + \ldots + R(k,1) = k+1 = \frac{(k+1)!}{k!1!} \tag{6}$$

Similarly, the ways of distributing k errors to 3 task instance, R(k,3) = R(0,2) + R(1,2) + ... + R(k,2), that is

$$R(k,3) = \sum_{m=0}^{k} R(m,2) = \sum_{m=0}^{k} \frac{(m+1)!}{m!1!}$$

And by the relationship of binomial coefficients $\sum_{m=0}^{k} \frac{(m+a)!}{m!a!} = \frac{(k+a+1)!}{k!(a+1)!}$, thus we have

$$R(k,3) = \sum_{m=0}^{k} \frac{(m+1)!}{m!1!} = \frac{(k+2)!}{k!2!} \tag{7}$$

From equation (6) and (7), we get the ways for distributing k errors to N task instances,

$$R(k,N) = \frac{(k+N-1)!}{k!(N-1)!} \tag{8}$$

Table 5 shows that the number of recovery patterns for distributing 2 errors into 3 instances is 6. Now we check it with equation (8).

$R(2,3) = \frac{(2+3-1)!}{2!(3-1)!} = 6$, which is exactly the same as the counting result from the recovery patterns in table 5.

5.3 Error Free Execution Analysis

An error-free execution can be either that all transient faults occurred when no primary copy was executing, or some faults occurred when the primary copies were executing but no error was detected. If there is no error detected in one planning cycle, it can be one of the following four different situations.

1. All faults occurred in idle time.

2. No fault generates an effective error.

3. Effective errors generated but not detected.

4. Part of faults occurred in idle time, but the others did not generate errors, or errors were not detected.

The error free execution for an instance set can be considered as corresponding to an empty error pattern. For f average faults in one planning cycle, we already know that a copy of instance τ_j' has a probability of $e^{-P_x f P_j}$ to be correct, so the probability for executing twice both correctly is $e^{-2P_x f P_j}$. Since each instance needs to execute correctly twice, the error free execution probability is simply the probability that all primary copies execute correctly, which is:

$$P_{EF} = e^{-2P_x f P_1} \times e^{-2P_x f P_2} \times \ldots \times e^{-2P_x f P_N} = \prod_{j=1}^{N} e^{-2P_x f P_j} \tag{9}$$

5.4 Error Execution Probability Analysis

Suppose there are *on average* f faults in one planning cycle which obeys the Poisson distribution, for instance τ_j' which has two correct copies, the probability for these two correct copies is $e^{-2P_x f P_j}$ as discussed in the section of error free execution. Now we consider erroneous copies. Each erroneous copy has a probability $(1 - e^{-P_x f P_j})$ to appear. If the number of erroneous copies of τ_j' is k_j^h , then the probability for τ_j' having two correct copies and k_j^h erroneous copies is: $e^{-2P_x f P_j}(1 - e^{-P_x f P_j})^{k_j^h}$.

And furthermore the probability for an error pattern corresponding to a recovery pattern, recovery(h), is:

$$P_{recovery}(h) = \prod_{j=1}^{N} \left(e^{-2P_x f P_j} \left(1 - e^{-P_x f P_j}\right)^{k_j^h} \right) = P_{EF} \prod_{j=1}^{N} \left(1 - e^{-P_x f P_j}\right)^{k_j^h}.$$

In fact in each error pattern, any instance also needs to execute correctly twice, so P_{EF} is a prefactor for all the error patterns.

As talked previously, we have a number of parameter(h) $= \prod_{j=1}^{N}(k_j^h + 1)$ error patterns resulting in the same recovery(h). So the probability for all those

error patterns is:

$$parameter(h) \times P_{recovery}(h) = P_{EF} \prod_{j=1}^{N}(k_j^h + 1) \times \prod_{j=1}^{N} \left(1 - e^{-P_x f P_j}\right)^{k_j^h}.$$

Therefore, for all the $R(k, N)$ recovery patterns which have k errors in the pattern, the probability is:

$$P(f, k, N) = \sum_{h=1}^{R(k,N)} parameter(h) \times P_{recovery}(h) \tag{10}$$

$$= P_{EF} \sum_{h=1}^{R(k,N)} \prod_{j=1}^{N}(k_j^h + 1) \times \prod_{j=1}^{N} \left(1 - e^{-P_x f P_j}\right)^{k_j^h}. \tag{11}$$

where $k_1^h + k_2^h + \ldots + k_N^h = k$. Obviously, $P(f, 0, N) = P_{EF}$ is the probability for 0 error occurrence, or error free execution probability.

To obtain the total probability of error execution , we need to include all possible k values. Recall that f is only the average number of faults, but the real number of faults in one planning cycle can be any integer from one to infinity for an error execution. Moreover, for considering the number of generated errors by f faults, it is possible to be any number from 0 to the number of faults. Therefore, the error execution probability for a task set with f average transient faults in one planning cycle is:

$$P(f, N) = \sum_{k=1}^{\infty} P(f, k, N) \tag{12}$$

$$= \sum_{k=1}^{\infty} \sum_{h=1}^{R(k,N)} parameter(h) \times P_{recovery}(h) \tag{13}$$

$$= P_{EF} \sum_{k=1}^{\infty} \sum_{h=1}^{R(k,N)} \prod_{j=1}^{N}(k_j^h + 1)(1 - e^{-P_x f P_j})^{k_j^h} \tag{14}$$

where $k_1^h, k_2^h, \ldots, k_N^h \geq 0$, and $k_1^h + k_2^h + \ldots + k_N^h = k$.

The summation does not include $k = 0$ because $k = 0$ stands for error free execution, in which $k_1^h = k_2^h = \ldots = k_N^h = 0$. The summation runs to infinity because the actual faults can be any number although *on average* it is f.

The sum of error free execution and those including 1 or more errors is unity, $P_{EF} + P(f, N) = 1$. It is verified in the output of our simulations. Of course we cannot sum up infinity terms in our code. Summing up a large number of terms is good enough. By checking the output, we always get $P_{EF} + P(f, N) \geq 0.999994$ in our example runs. Obviously, one can increase the accuracy by summing up more terms.

5.5 Limitations in This Fault Model

One limitation in our fault model is that when the number of task instances and the number of faults in one planning cycle increase, the number of error

patterns and recovery patterns will increase dramatically. For example, if the number of task instances is 100 and the number of faults is 10 in one planning cycle, the number of recovery patterns will be around 6×10^{10}, and this number exceeds the range of long integer in my machine, which is at the order of 10^9, and it will cause problems in computing the success probability.

6 Schedulability and Success Probability Analysis

6.1 Processor-Demand Analysis

After a schedule is generated, we need to check whether it is feasible. If a schedule for a task set can fulfill all the requirements for the task set, then the schedule is feasible. For RM, worst-case response-time analysis is usually applied to check the feasibility of a schedule. For EDF, processor-demand analysis is normally used to determine whether a task set is EDF_schedulable[29].

In processor-demand analysis, the amount of demanded processor time is calculated at each absolute deadline. If the demanded time is equal to or less than the available processor time at any point, then the schedule is feasible. In other words, a task is EDF_schedulable for a uniprocessor computer system and a given periodic task set $\mathcal{T} = \{\tau_1, ..., \tau_n\}$, if and only if the following equation holds:

$$\forall L : \sum_{i=1}^{n} \left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) \times C_i \leq L$$

for any checking point $L \in \{D_i^k = kT_i + D_i, D_i^k \leq LCM\{T_1, ..., T_n\}, 1 \leq i \leq n, k \geq 0\}$. Here C_i , T_i, and D_i are the Worst-Case-Execution-Time (WCET), the period , and the relative deadline of each task, respectively. In this work, the success probability of a periodic task set with f transient faults is calculated from the first planning cycle of the EDF_schedule.

6.2 An Extension of Processor-Demand Analysis

For each recovery pattern, processor demand analysis is employed to check the feasibility of schedule. When each instance executes twice, and $D_i = T_i$ is assumed, the formula of processor-demand analysis becomes the following equation given by M. Tengdahl [6]:

$$\forall L : \sum_{i=1}^{n} \left(\left\lfloor \frac{L - D_i}{D_i} \right\rfloor + 1 \right) \times 2C_i \leq L$$

The function was modified by M. Tengdahl again to address the sufficient and necessary condition for a task set with exactly one transient fault in one planning cycle [6].

$$\delta(L - d_{e,rec}) = \begin{cases} 0 & \text{if } L < d_{e,rec} \\ 1 & \text{if } L \geq d_{e,rec} \end{cases}$$

$$\forall L : \sum_{i=1}^{n} \left(\left\lfloor \frac{L - D_i}{D_i} \right\rfloor + 1 \right) \times 2C_i + \delta(L - d_{e,rec}) \times C_e \leq L \qquad (15)$$

Here we use $d_{e,rec}$ and C_e to denote the absolute deadline and the execution time of a recovery copy, and checking points $L \in \{D_i^k = kT_i + D_i, D_i^k \leq LCM\{T_1, ..., T_n\}, 1 \leq i \leq n, k \geq 0\}$. The step function $\delta(L - d_{e,rec})$ shows if the absolute deadline of a recovery copy is larger than the value of checking point L or not.

Follow the rules of processor-demand analysis, C_e can be a part of processor demand only when the absolute deadline of the faulty task copy equals to or is earlier than the given control point L, hence, $\delta(L - d_{e,rec}) \times C_e$ is used to prevent overestimation in computing the demanded time.

In this work, based on equation (15) and different error detection methods, schedulability analysis is separated into two parts. One part is for the errors detected by comparison or by timer monitor, and the other part is for the errors detected by HW/SW EDMs, to be discussed in the following two subsections, respectively.

6.2.1 Schedulability test if errors were detected by comparison or timer monitor

Now we consider a specific recovery pattern, recovery(h), and let d'_j , C'_j and k_j^h to be the absolute deadline, WCET, and the number of faulty copies in recovery(h) of task instance τ'_j , then we have:

$$\delta(L - d'_j) = \begin{cases} 0 & \text{if } L < d'_j \\ 1 & \text{if } L \geq d'_j \end{cases}$$

If there are errors detected by comparison or timer monitor, the execution time of a faulty task copy is its WCET, then we consider the following condition for the schedulability test:

$$\forall L : primary\ part + recovery\ part \leq L$$

The primary part is the processor time demand for the primary copies, which is $\sum_{i=1}^{n} \left(\left\lfloor \frac{L-D_i}{T_i} \right\rfloor + 1 \right) \times 2C_i$. The recovery part includes all the WCETs of faulty copies, $\sum_{\tau'_j \in recovery(h)} (\delta(L - d'_j) \times C'_j \times k_j^h)$. Thus, a sufficient and necessary condition for the schedulability of a task set $\mathcal{T} = \{\tau_1, ..., \tau_n\}$ with a certain recovery pattern, recovery(h), is :

$$\forall L : \sum_{i=1}^{n} \left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) \times 2C_i + \sum_{\tau'_j \in recovery(h)} (\delta(L - d'_j) \times C'_j \times k_j^h) \leq L \quad (16)$$

6.2.2 Schedulability test if errors were detected by HW/SW EDMs

If errors were detected by EDM, the execution time of a faulty task copy can be less than its WCET. We consider the following condition for the schedulability test:

$$\forall L : correct\ part + error\ part \leq L$$

Since each task instance should execute correctly twice, the correct part is the processor time demand for the two correct copies of each task instance, that is: $\sum_{i=1}^{n} \left(\left\lfloor \frac{L-D_i}{T_i} \right\rfloor + 1 \right) \times 2C_i$. Let $C_{err,j}^h$ to be the allowed maximum execution

time for recovery copies. Recall that k_j^h is the number of erroneous copies of τ_j' in recovery(h). Thus the error part becomes $\sum_{\tau_j' \in recovery(h)} \delta(L - d_j')C_{err,j}^h k_j^h$, where $\delta(L - d_j')$ is the same as in the case that errors detected by comparison or timer monitor.

Hence, a sufficient and necessary condition for the schedulability of a task set $\mathcal{T} = \{\tau_1, ..., \tau_n\}$ with a certain recovery pattern, recovery(h), is :

$$\forall L : \sum_{i=1}^{n} \left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) \times 2C_i + \sum_{\tau_j' \in recovery(h)} (\delta(L - d_j')C_{err,j}^h k_j^h) \leq L \quad (17)$$

From equation (17), we can see that the biggest $C_{err,j}^h$ makes the largest contribution to the faulty part, that is, the longer the execution time of faulty copies, the smaller the chance to pass the schedulability test. Since the maximum execution time $C_{err,j}^h$ is not known beforehand, we can use equation (17) to find out its allowed maximun value. If several instances or copies have the same absolute deadline, and a task set with a certain recovery pattern, recovery(h), is EDF_schedulable when the errors are detected by comparison or timer monitor, we let the execution time of each faulty copy to be its WCET, $(C_{err,j}^h = C_j')$, otherwise we will compute $C_{err,j}^h$ for instance τ_j' based on available processor demand and its WCET, C_j'. Details on assigning $C_{err,j}^h$ is given in the algorithm Chapter.

6.3 Success Probability Analysis

For feasibility test, we define a parameter ζ_h to represent the schedulability of a task set with a certain recovery pattern.

$$\zeta_h = \begin{cases} 0 & \text{Schedule is not feasible based on recovery(h)} \\ 1 & \text{Schedule is feasible based on recovery(h)} \end{cases}$$

Recall that $P_{recovery}(h)$ is the error execution probability for a certain recovery patter, recovery(h) (see section 5.4). Moreover, let $P^{\zeta}(h) = \zeta_h P_{recovery}(h)$ to be the error execution probability when the task set is EDF_schedulable with a certain recovery pattern. Thus, we have

$$P^{\zeta}(h) = \begin{cases} 0 & \text{Schedule is not feasible based on recovery(h)} \\ P_{recovery}(h) & \text{Schedule is feasible based on recovery(h)} \end{cases}$$

Also recall that parameter(h) is the number of error patterns corresponding to the same recovery pattern, recovery(h). Thus, $parameter(h)P^{\zeta}(h)$ is the probability of errors being recovered for all the error patterns corresponding to the recovery pattern, recovery(h), based on EDF_schedulability test.

From equation (13), and all the possible error patterns which still keep the task set EDF schedulable, we obtain the error execution recovery probability for the task set includeing N instances when errors were induced by f average transient faults in a planning cycle based on EDF_schedulability test,

$$E(f, N, \zeta) = \sum_{k=1}^{\infty} (\sum_{h=1}^{R(k,N)} parameter(h)P^{\zeta}(h)) \quad (18)$$

where $k_1^h, k_2^h, \ldots, k_N^h \geq 0$, $k_1^h + k_2^h + \ldots + k_N^h = k$, and $1 \leq k$.

6.3.1 Formula to compute success probability

The success probability of a task set, $\mathcal{T} = \{\tau_1, ..., \tau_n\}$, includes two parts. The first part is the probability of error free execution, P_{EF}, as discussed in the fault model section. The second part is the probability of detecting and masking effective errors, P_{error}. Thus total success probability is:

$$P_{success} = P_{EF} + P_{error}$$

In which, $P_{error} = P_{DET} + P_{EDM}$.

P_{DET} is the part of the success probability for recovery patterns passed schedulability test, in which errors were detected by double executions or timer monitor. The execution time of recovery copies are the WCET of faulty instances. P_{DET} can be derived from equation (18),

$$P_{DET} = E(f, N, \zeta)(P_{DE} \times P_{DE,M} + P_T \times P_{T,M}) \tag{19}$$

As shown in table 11, P_{DE} is the probability for errors being detected by comparison; $P_{DE,M}$ is the probability for errors detected by comparison being masked by TEM; P_T is the probability for errors being detected by timer monitor; $P_{T,M}$ is the probability for errors detected by timer monitor being masked by TEM.

P_{EDM} is the part of the success probability of a task set from recovering errors detected by HW/SW EDMs and masked by TEM. Since by HW/SW EDMs, an error can be detected earlier than the faulty copy finishes its execution, some recovery patterns which would not schedulable when errors were detected by comparison, perhaps can be schedulable now.

The maximum execution time of the faulty copy of instance τ'_j in recovery(h), $C^h_{err,j}$, can be separated into two parts, the maximum normal execution time of a faulty copy, C^h_j, and the latency between error generation and being detected, t_{lat}. We use the value calculated by Lou in [5], $t_{lat} = 0.45ms$, then $C^h_j = C^h_{err,j} - t_{lat}$. If a recovery pattern satisfies schedulability test by comparison, it definitely satisfies the schedulability test by EDM, and the maximum execution time of a task copy in this recovery pattern is its WCET,$C^h_{err,j} = C'_j$. The probability for the error being detected in an erroneous copy of instance τ'_j in recovery(h) is:

$$\frac{C^h_j}{C'_j} = \frac{C^h_{err,j} - t_{lat}}{C'_j}$$

Similarly, the probability for all the errors being detected in the erroneous copies in recovery(h) for N instances in one planning cycle is :

$$\Delta_h = \begin{cases} 0 & \exists j : C^h_{err,j} - t_{lat} \leq 0 \\ \prod_{j=1}^{N} (\frac{C^h_{err,j} - t_{lat}}{C'_j})^{k^h_j} & \forall j : C^h_{err,j} - t_{lat} > 0 \end{cases}$$

Where $C^h_{err,j} \leq C'_j$ and $1 \leq j \leq N$.

Moreover, since $P^{\zeta}(h)$ is the error execution recovery probability when the task set is EDF_schedulable for recovery(h), together with Δ_h, $P^{\zeta}(h) \times \Delta_h$ is the probability for recovery(h) to be EDF_schedulable when errors were detected by EDM. We will use this value instead of $P^{\zeta}(h)$ in equation (18) to calculate the success probability from recovery(h) for a task set when errors are detected by

HW/SW EDMs and masked by TEM. Thus, we modify equation (18) to adapt the error detection property of HW/SW EDMs as following:

$$E(f,N,\Delta) = \sum_{k=1}^{\infty}(\sum_{h=1}^{R(k,N)} parameter(h) P^{\zeta}(h) \Delta_h) \tag{20}$$

where $k_1^h, k_2^h, \ldots, k_N^h \geq 0$, $k_1^h + k_2^h + \ldots + k_N^h = k$, and $1 \leq k$.

Now we have the success probability for a task set when errors were detected by HW/SW EDMs and masked by TEM,

$$P_{EDM} = E(f,N,\Delta) \times P_{ED} \times P_{ED,M} \tag{21}$$

Where P_{ED} is the probability of errors being detected by HW/SW EDMs, and $P_{ED,M}$ is the probability of errors detected by HW/SW EDMs being masked by TEM.

7 Algorithms

7.1 Feasibility Test

Algorithm 1 feasi_test($\mathcal{T}, \tau_i$, recovery(h))

Require: EDF-feasible task set $\mathcal{T}$ with $U_{total}(\mathcal{T}) < 1$, recovery pattern: recovery(h)

Ensure: Whether task set schedulable with this certain recovery pattern

1: $PDL = REC\_LOAD := 0$;
2: $primary[numk]$ $\{primary[k] := \sum_{i=1}^{n} \left(\left\lfloor \frac{L-D_i}{T_i} \right\rfloor + 1 \right) \times 2C_i$, precomputed$\}$
3: $L[numk] := \{d_k \in \mathcal{D} : d_k \leq PC\}$ {abosulute deadline set,precomputed}
4: $d[N], for\ 1 \leq i \leq N, d[i] := d_i$ {abosulute deadline set for each task instances,precomputed}
5: **for** $k = 1$ *to* $numk$ **do**
6: **while** $i \leq N$ and $d[i] \leq L[k]$ **do**
7: $REC\_LOAD := REC\_LOAD + k_i^h \times C[i]$
8: $i++$
9: **end while**
10: $pdL := primary[k] + REC\_LOAD$ {primary part + recovery part}
11: **if** $pdL > L[k]$ **then**
12: return 0 {If task set with current recovery pattern is not EDF schedulable}
13: **end if**
14: **end for**
15: return 1 {task set with current recovery pattern is EDF schedulable}

This algorithm does the EDF schedulability test for a recovery pattern, recovery(h), of a task set. When errors were detected by comparison or timer monitor and the pattern is schedulable, 1 is returned. Otherwise, 0 is returned.

Some pre-known or precomputed arguments for algorithm 1 : L[numk] is an array of checking points; d[N] is an array of deadlines for task instances;

primary[numk] , $primary[k] := \sum_{i=1}^{n} \left(\left\lfloor \frac{L-D_i}{T_i} \right\rfloor + 1 \right) \times 2C_i$, is an array of pre-computed workloads for each L[k]. Recovery(h) is a recovery pattern, and k_i^h is the number of τ_i' in recovery(h).

Primary[k], d[i] and L[k] are in ascending orders. For each checking point L[k], we calculate the processor time demanded by the recovery part (REC_LOAD, as shown in line 7). Adding it to the primary part (in line 10) we obtain the total demanded time at the current checking point L[k]. If the task set with a certain recovery pattern is not EDF schedulable, 0 is returned, otherwise 1 is returned.

7.2 Computing the Δ of a Recovery Pattern to Keep a Schedule Feasible

If a task set is not EDF schedulable when the errors are detected by comparison or timer monitor, we will use this algorithm to calculate the probability for a recovery pattern to be schedulable if errors in the faulty copies in recovery(h) are detected by HW/SW EDMs. Processor demand analysis is applied to compute the execution time of each faulty copy. If the task set with recovery(h) can be possibly EDF_schedulable when errors are detected by HW/SW EDMs, the Δ_h is computed and returned, otherwise, 0 is returned.

Parameter ERR_LOAD is the total processor time demanded by the faulty part. $ERR\_LOAD[k]$ is the total available processor time can be used by the error part from the previous checking point, L[K-1], to the current checking point, L[k]. The execution time of each erroneous copy is assumed to be its WCET. When the processor time demanded at the current checking point exceeds L[k], based on the remaining available processor demand, we either use $C_{err,j} + t_{lat}$ as the execution time of an erroneous copy of instance τ_j', or re-assign the value of $C_{err,j}$. Finally, Δ_h is calculated for each recovery pattern based on $C_{err,j}$.

Algorithm 2 maxi_exe($\mathcal{T}, \tau_i$, recovery(h))

Require: EDF-feasible task set $\mathcal{T}$ with $U_{total}(\mathcal{T}) < 1$, recovery pattern: recovery(h)

Ensure: Assign execution time to faulty copies and get the possible value of Δ

1: $PDL = ERR\_LOAD1 = ERR\_LOAD2 := 0$;
2: $\Delta = i := 1; num = EXTRA\_LOAD := 0; t_{lat} = 0.45$
3: $primary[maxk]$ $\{primary[k] := \sum_{i=1}^{n} \left(\left\lfloor \frac{L-D_i}{T_i} \right\rfloor + 1 \right) \times 2C_i$, precomputed$\}$
4: $L[maxk] := \{d_k \in \mathcal{D} : d_k \leq PC\}$ {abosulute deadline set,precomputed}
5: $d[N], for 1 \leq i \leq N, d[i] := d_i$ {abosulute deadline set for each task instances,precomputed}
6: **for** $k = 1$ *to* $maxk$ **do**
7: $ERR\_LOAD[k] := 0$
8: $j := i$
9: **while** $i \leq N$ and $d[i] \leq L[k]$ **do**
10: **if** $k_i^h > 0$ **then**
11: $C_{err,i} := C[i] - t_{lat}$
12: $ERR\_LOAD[k] := ERR\_LOAD[k] + k_i^h \times C[i]$
13: **end if**
14: $i++$
15: **end while**
16: $ERR\_LOAD := ERR\_LOAD + ERR\_LOAD[k]$
17: $PDL := primary[k] + ERR\_LOAD$ {correct part + faulty part}
18: **if** $PDL > L[k]$ **then**
19: {If needed processor demand is larger than check point $L[k]$}
20: **if** $(L[k] - PDL - ERR\_LOAD[k]) \leq (num[k] \times t_{lat})$ **then**
21: {If the remaining available processor demand is less than the value of needed latency demand.}
22: **for** $m = 1$ *to* $i - 1$ **do**
23: re-assign $C_{err,m}$ for all erroneous copies of which deadlines are equal to or earlier than the check point L[k];
24: and renew ERR_LOAD
25: **end for**
26: **else**
27: **for** $m = j$ to $i - 1$ **do**
28: assign $C_{err,m}$ for new coming erroneous copies based on remaining available processor demand
29: and renew ERR_LOAD
30: **end for**
31: **end if**
32: **end if**
33: **end for**
34: $\Delta := \prod_{i=1}^{N} (\frac{C_{err,i}}{C[i]})^{k_i^h}$
35: RETURN Δ { EDF schedulable}

7.2.1 Computing the execution time of erroneous copies

There are different ways to estimate the execution time of erroneous copies. One approach is based on the utilization of faulty copies. However, we will only discuss the method we used for this algorithm. It may not be an optimal way to compute the largest execution time of erroneous copies. This algorithm can possibly be improved in future work.

In our solution, suppose that instances τ_1' and τ_2' have the same deadline, and τ_1' has 1 faulty copy whereas τ_2' has 2 faulty copies, and the available processor demand is L_{RPD}. The latency t_{lat} is the time elapse between error generation and error detection. Firstly, we will assign latency to each erroneous copy to make sure that each copy has a chance to be detected erroneous. The left available processor time is $(L_{RPD} - 3 \times t_{lat})$. Secondly, we will assign the remaining processor time to each erroneous copy proportional to their WCETs. In this example, $C^h_{err,1} = (L_{RPD} - 3 \times t_{lat}) \times \frac{C_1' - t_{lat}}{C_1' + 2C_2'} + t_{lat}$, and $C^h_{err,2} = (L_{RPD} - 3 \times t_{lat}) \times \frac{C_2' - t_{lat}}{C_1' + 2C_2'} + t_{lat}$.

$C^h_{err,j}$ can be changed in the further analysis at later checking points. For example when the next checking point is reached, 2 more erroneous copies of τ_3' came up, and the remaining processor time can even not be enough for latencies, $(L_{RPD} - 2 \times t_{lat} \leq 0)$, then the previously allocated processor time for erroneous copies will be re-collected and re-assigned to each erroneous copy. In our example, if the current checking point is L[k], and the processor time demanded by primary copies is primary[k], the total available processor demand for erroneous copies will be L[k]-primary[k]. Then, $C^h_{err,j}$ is re-assigned based on C_j' and L[k]-primary[k], that is $C^h_{err,j} = (L[k] - primary[k] - 5 \times t_{lat}) \times \frac{C_j' - t_{lat}}{C_1' + 2C_2' + 2C_3'} + t_{lat}, (1 \leq j \leq 3)$.

7.3 Calculating the Success Probability

This algorithm computes the success probability of a task set with f average transient faults in one planning cycle.

The index of the outer loop, k, is the number of errors in a recovery pattern. The index of the inner loop, h, points to the $h-th$ recovery pattern including k effective errors. For each recovery(h), algorithm 1 is called for feasibility test. If the returned value from algorithm 1 is 1, the corresponding error execution recovery probability and other parameters are prepared to computer P_{DET}, and P_{DEM} . Otherwise, if the returned value of algorithm 1 is 0, algorithm 2 is called to compute the Δ of recovery(h) for P_{DEM}. Finally, the probability of error free execution P_{EF} and the probability of error detection and masking part, P_{error} are computed, and the success probability of a task set with f average transient faults in one planning cycle is returned.

Algorithm 3 proba_cal($\mathcal{T}, \tau_i$, recoverytxt[R(k,N)][N])

Require: EDF-feasible task set $\mathcal{T}$ with $U_{total}(\mathcal{T}) < 1$, recovery pattern matrix: recoverytxt[R(k,N)][N]

Ensure: Success probability is computed based on feasibility test or feasibility test with the execution time of faulty task copies.

1: $t_{lat} := 0.49; P_x := 0.17$, Other parameters check the table
2: $P_{error} := 0; P_{EF} := 1$; {initial success probability and fault probability}
3: **for** $k = 1$ *to* $max(20, f)$ **do**
4: $P_{DET} = P_{DEM} := 0$; {initial P_{DET} and P_{DEM} }
5: **for** $h = 1$ *to* $R(k, N)$ **do**
6: read line h from recoverytxt, from recoverytxt[h][1] to recoverytxt[h][N], and copy to array recovery(h)
7: $PU := \prod_{j=1}^{N}(e^{-2P_x f P_j}(1 - e^{-P_x f P_j})^{K_j^h}(K_j^h + 1))$ {collect the error execution probability of all error patterns corresponding to recovery(h)}
8: **if** $feasi\_test(recovery(h)) == 1$ **then**
9: $P_{DET} := P_{DET} + PU$ {collect the total fault occurrence probability to compute P_{DET}}
10: $\Delta := \prod_{i=1}^{N}(\frac{C[i]-t_{lat}}{C[i]})^{K_i^h}$ {probability of error execution caused the shorter execution time in τ_i'}
11: $P_{DEM} := P_{DEM} + \Delta \times PU$ {collect the total error execution probability to compute P_{DEM}}
12: **else**
13: $\Delta := maxi\_exe(recovery(h))$
14: **if** $\Delta > 0$ **then**
15: $P_{DEM} := P_{DEM} + \Delta \times PU$
16: **end if**
17: **end if**
18: **end for**
19: $P_{error} := P_{error} + P_{DET} \times (P_{DE} \times P_{DE,M} + P_T \times P_{T,M}) + P_{DEM} \times P_{ED} \times P_{ED,M}$
20: **end for**
21: $P_{EF} = \prod_{j=1}^{N} e^{-2P_x f P_j}$
22: RETURN $(P_{error}+P_{EF})$

8 Examples

Our program written in C language is based on the system model described in chapter 5 , the schedulability and success probability analysis presented in chapter 6, and the algorithms in chapter 7. In this chapter, we give two examples. The success probabilities for the two examples were calculated. We also discuss the meanings of our results.

8.1 Example 1

In Table 7, there are three tasks, τ_1, τ_2 and τ_3. The time unit for period T_i, WCET C_i, and relative deadline D_i is millisecond(ms). The latency between error generation and error detection is $t_{lat} = 0.45ms$. This example is taken from Lou [5]. We use it to show that our system model and program simulation give almost the same success probability when one fault in one planning cycle is assumed. A slight difference between our results and those from [5] arises from different assumptions on the fault generation processes. For comparison, we quote the success probability from Lou's version for one fault is 0.9311.

Task	$T_i(ms)$	$C_i(ms)$	$D_i(ms)$
τ_1	100	20	100
τ_2	200	40	200
τ_3	400	25	400

Table 7: The task set for example 1. T_i, C_i, and D_i are the period, WCET, and relative deadline, respectively.

From the simulation of preemptive EDF scheduling, an instance set $\mathcal{T}' = \{\tau'_1, \tau'_2, \ldots, \tau'_7\}$ is collected from the first planning cycle as shown in Table 8. Each instance $\tau'_j (1 <= j <= 7)$ includes its absolute deadline d'_j, WCET C'_j , and fault occurrence probability $P_j = \frac{C'_j}{LCM}$. $L[k]$ are the checking points which include all possible absolute deadlines of the instance set. And $primary[k]$ is the processor demand for primary copies (or correct copies when errors are detected by HW/SW EDMs), $primary[k] = \sum_{i=1}^{7} \left(\left\lfloor \frac{L[k]-D_i}{D_i} \right\rfloor + 1 \right) \times 2C_i$, where the values for C_i and D_i are from Table 7.

The results from simulation are shown in Table 9. P_{error} is the probability for errors detected by error detection mechanisms and masked by TEM. P_{EF} is the probability of error free execution. $P_{success} = P_{error} + P_{EF}$ shows the success probability of a task set with average f transient faults in one planning cycle.

The data in Table 9 are plotted in Figure 4. From Figure 4 and Table 9, we can see that when the average number of faults increases from $f = 1$ to $f = 7$, P_{error} also increases. However from $f = 8$ to $f = 40$, when the average number of faults increases, P_{error} decreases. P_{EF} and P_{sucess} monotonically decrease in the whole range.

The error free execution gives a larger contribution to the successful execution up to $f = 9$ than P_{error} does. But P_{EF} drops sharply with the increasing number of faults. This is mainly due to the large utilization of the instance set.

Instance	C'_j	d'_j	P_j	$L[k]$	$primary[k]$
τ'_1	20	100	0.05	100	40
τ'_2	20	200	0.05	200	160
τ'_3	40	200	0.1	300	200
τ'_4	20	300	0.05	400	270
τ'_5	20	400	0.05		
τ'_6	40	400	0.1		
τ'_7	25	400	0.0625		

Table 8: The instance set in example 1. C'_j, d'_j, and P_j are the WCET, the absolute deadline, and the fault occurrence probability for task instance τ'_j, respectively. $L[k]$ are the checking points, and $primary[k]$ the processor demands for primary copies. P_j is dimensionless. The unit for other quantities is ms.

The utilization for executing each instance twice correctly is $2\sum_{j=1}^{7} P_j = 0.925$, so there is only $7.5\%LCM$ left for possible errors to happen without causing faulty copies. When there are more faults in the planning cycle, they are very likely to cause faulty copies and the chance of error free execution is small.

When there are only a few erroneous copies, the double execution mechanism can correct most of them, so P_{error} initially increases with f until $f = 7$. But when there are too many faulty copies, the mechanism is unable to cope with them before the deadlines and the correction rate decreases. However, the decreasing rate is smaller than the rate of P_{EF}, and the correction to the faulty copies gives a significant contribution to $P_{success}$. For $f \geq 10$, its contribution surpasses the contribution from P_{EF}.

8.2 Example 2

In Table 10, there are two tasks, τ_1 and τ_2. Millisecond(ms) is the time unit as before. The latency $t_{lat} = 0.45ms$. Comparing to the first example, this task set has a lower utilization for primary (or correct) copies, $2\sum_{i=1}^{2} \frac{C_i}{T_i} = 0.16$, and the relative deadlines are earlier than corresponding task periods.

From the simulation of preemptive EDF scheduling, an instance set $\mathcal{T}' = \{\tau'_1, \tau'_2, \tau'_3\}$ is collected from the first planning cycle as shown in Table 11. It gives the absolute deadline d'_j, WCET C'_j , and fault occurrence probability P_j for each task instance, as well as the checking point $L[k]$. Notations are the same as in example 1.

Simulation results are presented in Table 12 and plotted in Figure 5. Again we see the monotonic decrease of P_{EF} and P_{sucess} with the increasing average number of faults f. However, the decreasing rates are smaller than in example 1 because the utilization of correct copies in example 2 is smaller. A large fraction of errors happened in the idle time and those errors do not cause faulty instance copies.

In this example with increasing f from 1 to 40, P_{error} almost also linearly increase. This shows that the faulty copies of instances can almost all be corrected by the TEM especially when errors are detected by HW/SW EDMs. The system is more resilient than the one in example 1. Because the primary copies in task set 2 only take a small fraction of the planning cycle, there is ample time

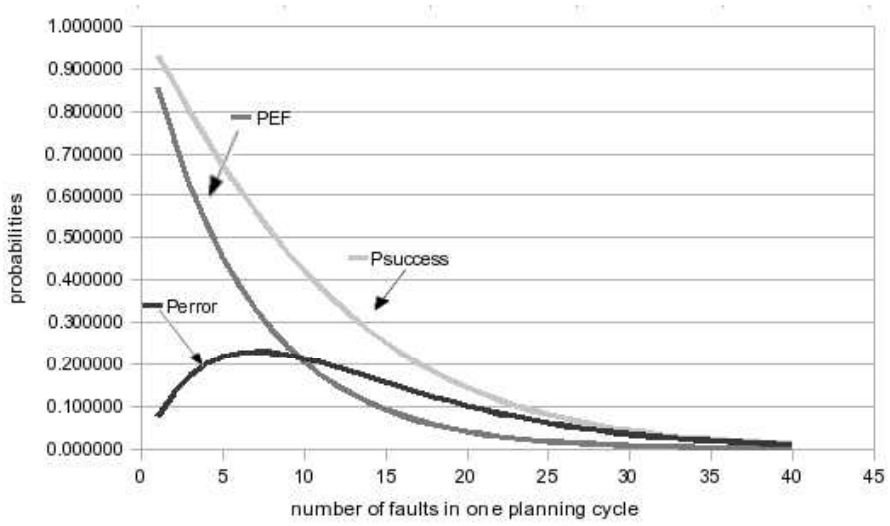

Figure 4: Plot for Example 1. See text and the caption of Table 9 for the meanings of labels.

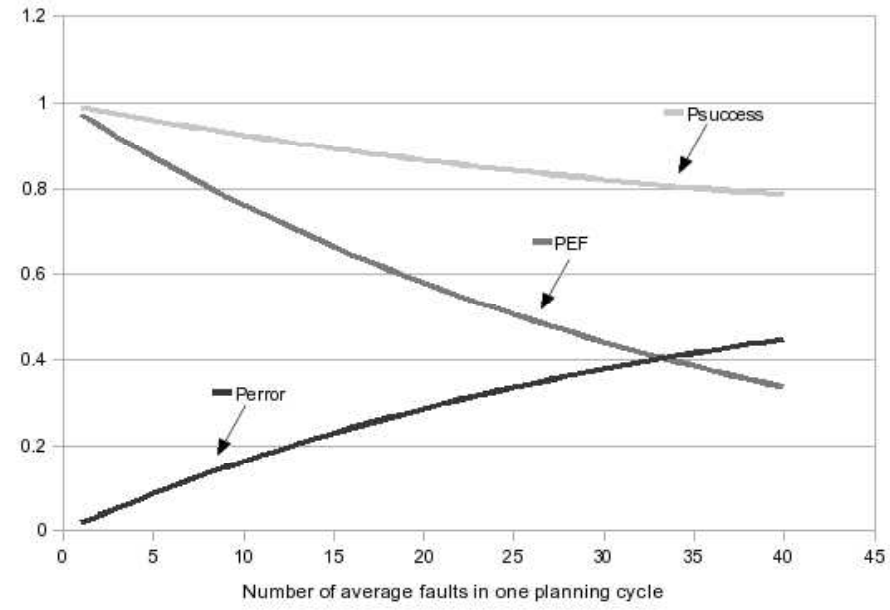

Figure 5: Plot for Example 2. See text and the caption of Table 12 for the meanings of labels.

f	P_{error}	P_{EF}	$P_{success}$	f	P_{error}	P_{EF}	$P_{success}$
1	0.076833	0.854490	0.931324	21	0.094137	0.036801	0.130938
2	0.133577	0.730154	0.863731	22	0.085406	0.031446	0.116852
3	0.174129	0.623909	0.798038	23	0.077314	0.026870	0.104184
4	0.201721	0.533125	0.734846	24	0.069847	0.022960	0.092807
5	0.219030	0.455551	0.674581	25	0.062983	0.019619	0.082602
6	0.228259	0.389269	0.617528	26	0.056696	0.016764	0.073460
7	0.231219	0.332625	0.563845	27	0.050954	0.014325	0.065279
8	0.229390	0.284224	0.513614	28	0.045725	0.012241	0.057966
9	0.223973	0.242866	0.466840	29	0.040976	0.010460	0.051436
10	0.215942	0.207530	0.423472	30	0.036672	0.008938	0.045610
11	0.206077	0.177331	0.383408	31	0.032780	0.007637	0.040417
12	0.195001	0.151527	0.346529	32	0.029268	0.006526	0.035793
13	0.183206	0.129478	0.312684	33	0.026103	0.005576	0.031679
14	0.171076	0.110638	0.281714	34	0.023257	0.004765	0.028021
15	0.158907	0.094539	0.253445	35	0.020701	0.004072	0.024772
16	0.146922	0.080782	0.227705	36	0.018409	0.003479	0.021888
17	0.135290	0.069028	0.204318	37	0.016356	0.002973	0.019329
18	0.124127	0.058984	0.183111	38	0.014520	0.002540	0.017061
19	0.113517	0.050401	0.163918	39	0.012880	0.002171	0.015051
20	0.103511	0.043067	0.146578	40	0.011417	0.001855	0.013272

Table 9: Simulation results for example 1. f is the average number of transient faults in one planning cycle. P_{error} is the probability of errors detected and masked by TEM. P_{EF} error free execution. $P_{success} = P_{error} + P_{EF}$ is the success probability.

left to re-execute erroneous copies and correct errors.

9 Future Work

Improving our current fault model is an interesting and meaningful extension to this work. One may consider different statistical distributions for errors and analyze the system success probabilities. The comparison between the results from different fault distributions is interesting.

In this work, the recovery copies of an instance have the same absolute deadline as that of the instance. Since EDF is considered, in a run time system, keeping on running recovery copies may hinder other primary copies to execute, then the success probability of the rest instances can be decreased. It might be better if we give higher priority to primary copies than to recovery copies, and treat recovery copies as sporadic tasks, or change the deadlines of recovery copies as Tengdahl did in [6]. Those probably are better strategies for the success probability of some instances in a run time system.

Using different ways to compute the execution time of each erroneous copy in a recovery pattern can also be a worthy try. In this study, when a task set with a certain recovery pattern is non EDF schedulable if errors are detected by comparison or timer monitor, we assume that the errors can only possibly be

Task	$T_i(ms)$	$C_i(ms)$	$D_i(ms)$
τ_1	250	10	200
τ_2	500	20	450

Table 10: The task set for example 2. T_i, C_i, and D_i are the period, WCET, and relative deadline, respectively.

Instance	C'_j	d'_j	P_j	$L[k]$	$primary[k]$
τ'_1	10	200	0.02	200	20
τ'_2	10	450	0.02	450	80
τ'_3	20	450	0.04		

Table 11: The instance set in example 2. C'_j, d'_j, and P_j are the WCET, the absolute deadline, and the fault occurrence probability for task instance τ'_j, respectively. $L[k]$ are the checking points, and $primary[k]$ the processor demands for primary copies. P_j is dimensionless. The unit for other quantities is ms.

detected by HW/SW EDMs, and did not consider the probability of errors being detected by other mechanisms. Considering the probability of all error detection mechanisms for each copy may result in a more accurate success probability.

RM schedulability analysis for our fault model can be another attractive study as well. One can calculate the RM success probability of a task set and compare it with those obtained from EDF scheduling.

Finally, extending our current model to multiprocessor systems is certainly a challenging and meaningful work.

10 Conclusion

In this thesis, we presented a model which connects the concepts in real time scheduling and those in fault tolerant reliability. Statistical analysis, including binomial coefficients, binomial distribution, Poisson distribution, is employed to analyze the numbers of error patterns and of recovery patterns.

We analyzed the schedulability of a task set with on average f transient faults in one planning cycle based on all possible recovery patterns. In our analysis, error detection mechanisms, such as comparison, timer monitor, and HW/SW EDMs, are treated separately according to their different properties.

The success probability of a task set relies on the number of error patterns that can pass the EDF schedulability test, and parameters from the experimental results of Artk68-FT [9]. The partial success probabilities for errors detected by comparison and timer monitor and for errors detected by HW/SW EDMs are given in the thesis.

At last, we simulated schedulability analysis and computed success probabilities for task sets with f average transient faults using the current model. Some examples are given and results are interpreted.

f	P_{error}	P_{EF}	$P_{success}$	f	P_{error}	P_{EF}	$P_{success}$
1	0.018479	0.973167	0.991646	21	0.297011	0.564847	0.861858
2	0.036448	0.947053	0.983501	22	0.307203	0.549690	0.856893
3	0.053919	0.921641	0.975560	23	0.317106	0.534940	0.852046
4	0.070908	0.896910	0.967817	24	0.326728	0.520586	0.847314
5	0.087425	0.872843	0.960268	25	0.336075	0.506617	0.842692
6	0.103485	0.849421	0.952906	26	0.345156	0.493023	0.838179
7	0.119099	0.826628	0.945727	27	0.353978	0.479793	0.833771
8	0.134279	0.804447	0.938726	28	0.362547	0.466919	0.829466
9	0.149036	0.782861	0.931897	29	0.370870	0.454390	0.825260
10	0.163383	0.761854	0.925237	30	0.378954	0.442197	0.821150
11	0.177330	0.741411	0.918741	31	0.386804	0.430331	0.817135
12	0.190887	0.721517	0.912404	32	0.394428	0.418784	0.813212
13	0.204066	0.702156	0.906221	33	0.401831	0.407547	0.809378
14	0.216875	0.683315	0.900190	34	0.409019	0.396611	0.805630
15	0.229326	0.664979	0.894305	35	0.415998	0.385968	0.801967
16	0.241427	0.647135	0.888563	36	0.422774	0.375611	0.798385
17	0.253189	0.629770	0.882959	37	0.429351	0.365533	0.794884
18	0.264619	0.612871	0.877490	38	0.435735	0.355724	0.791459
19	0.275727	0.596426	0.872153	39	0.441932	0.346179	0.788111
20	0.286521	0.580422	0.866943	40	0.447946	0.336890	0.784835

Table 12: Simulation results for example 2. f is the average number of transient faults in one planning cycle. P_{error} is the probability of errors detected and masked by TEM. P_{EF} error free execution. $P_{success} = P_{error} + P_{EF}$ is the success probability.

References

[1] C. M. Krishna and K. G. Shin. *Real-Time Systems.* McGraw-Hill International Editions, 1997.

[2] N. Storey. Safety-Critical Computer Systems. *Prentice Hall.*,ISBN 0-201-42787-7, 1996.

[3] J. Aidemark, J. Vinter, P. Folkesson, and J. Karlsson. Experimental evaluation of time-redundant execution for a brake-by-wire application. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 210–215, June 2002.

[4] A. Avizienis. Design of fault-tolerant computers. *AFIPS conference proceedings.*,vol.31.pp.733-743, 1967.

[5] Q. Lou. Real-time scheduling analysis of system employing tem. Master's thesis, Chalmers University of Technology, 2005.

[6] M. Tengdahl. Tolerating transient faults in Real-time systems with Dynamic Task Priorities . Master's thesis, Chalmers University of Technology, Gothenburg University, 2008.

[7] R. M. Pathan. Fault-tolerant real-time scheduling algorithm for tolerating multiple transient faults. In *4th International Conference on Electrical and Computer Engineering*, pages 577–580, December 2006.

[8] R. M. Pathan. Real-time scheduling analysis of systems tolerating multiple transient faults. Master's thesis, Chalmers University of Technology, 2005.

[9] J. Aidemark, J. Vinter, P. Folkesson, and J. Karlsson. Experimental dependability evelutation of the artk68-ft real-time kernel. In *Proceedings of the International Conference on Real-Time and Embedded Comuting Systems and Applications*, August 2004.

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

[11] J. Aidemark. *Node-Level Fault Tolerance for Embedded Real-Time Systems.* PhD thesis, Chalmers University of Technology, 2004.

[12] X. Castillo, S. R.McConnel,and D. P.Siewiorek. Derivation and Calibration of a Transient Error Reliability Model. *IEEE Transactions on Computers*, 4(3):214–237, Aug. 1986.

[13] J. Aidemark, P. Folkesson, and J. Karlsson. A framework for node-level fault tolerance in distributed real-time systems. Technical Report 04-06, Department of Computer Engineering, Chalmers University of Technology, 2004.

[14] H. Mei-Chen, T.K. Tsai, P.K. Iyer. Fault Injection Techniques and Tools. *Computer*, Volume:30, Issue:4(1997).

[15] J. Aidemark, P. Folkesson, and J. Karlsson. GOOFI: Generic Object-Oriented Fault Injection tool. Proceedings of the International Conference on Dependable Systems and Networks, Gothenburg, 2001.

[16] J. Abawajy. Fault-tolerant dynamic job scheduling policy. In *6th international conference on algorithms and architectures for parallel processing*, pages 165–173, October 2005.

[17] G. Manimaran, C.S.R. Murthy. A Fault-tolerant Scheduling Algorithm for Multiprocessor Real-time Systems and its Analysis. *IEEE Transactions on Parallel and Distributed Systems.*,1045-9219, Pages: 1137 - 1152, 1998.

[18] Y. Oh, S.H. Son. An Algorithm for Real-Time Fault-Tolerant Scheduling in Multiprocessor Systems. *2008 Third International Conference on Convergence and Hybrid Information Technology* .,978-0-7695-3407-7, Pages: 816-821, 2008.

[19] F. Liberato, S. Lauzac, R. Melhem, D. Mossé. Fault Tolerant Real-Time Global Scheduling on Multiprocessors. *Real-Time Systems, 1999.*,0-7695-0240-7, Pages: 252-259, 1999.

[20] Y.S. Hong, H.W. Goo. A Fault-tolerant Scheduling Scheme for Hybrid Tasks in Distributed Real-time Systems. *Third IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems(SEUS'05),2005.*, 0-7695-2357-9,2005.

[21] H. Aydin, R. Melhem, D. Mossé. Optimal Scheduling of Imprecise Computation Tasks in the Presence of Multiple Faults. *Real-Time Computing Systems and Applications, 2000.*, ISBN: 0-7695-0930-4,page(s): 289-296,2002.

[22] F. Liberato, R. Melhem, D. Mossé. Tolerance to Multiple Transient Faults for Aperiodic Tasks in Hard Real-Time Systems in Hard Real-time Systems. *IEEE Transactions on Parallel and Distributed Systems.*,1045-9219, Pages: 272 - 284, 1997.

[23] E. W. Weisstein, other contributors. Wolfram MathWorld. http://mathworld.wolfram.com/PoissonDistribution.html

[24] Nist, Sematech. e-Handbook of Statistical Methods. http://www.itl.nist.gov/div898/handbook/pmc/section3/pmc331.htm , 25 October 2006

[25] E. Merran, N. Hastings, B. Peacock. Statistical Distributions. *New York: Wiley.*, 3rd ed.,ISBN 0-471-37124-6,2000.

[26] P. M. Morse. Thermal physics. *W. A. Benjamin, inc*, 1964.

[27] unknown contributors. Wikipedia. http://en.wikipedia.org/wiki/Poisson_distribution , 01 June 2009

[28] E. W. Weisstein, other contributors. Wolfram MathWorld. http://mathworld.wolfram.com/BinomialCoefficient.html .

[29] S. K. Baruah, L. E. Rosier, and R. R. Howell. Algorithms and complexity concerning the preemptively scheduling of periodic, real-time tasks on one processor. *Real-Time Systems Journal*, 2(4):301–324, November 1990.

11 APPENDIX - Fault Injection Results

P_X	Given that a fault occurs, an error is generated	373 (of 2076)	17%
P_{DE}	Given that an error is generated, the error is detected by comparison after double execution (DE)	68 (of 373)	18%
P_T	Given that an error is generated, the error is detected by a timer monitor (TM)	19 (of 373)	5%
P_{ED}	Given that an error is generated, the error is detected by a hardware error detection mechanism (EDM)	286 (of 373)	77%
P_{ND}	Given that an error is generated, the error is not detected	0 (of 373)	0%
$P_{DE,M}$	Given that an error is detected by DE, the error is masked by TEM	68 (of 68)	100%
$P_{T,M}$	Given that an error is detected by TM, the error is masked by TEM	18 (of 286)	6%
$P_{ED,M}$	Given that an error is detected by EDM, the error is masked by TEM	194 (of 286)	68%

Table 13: Results from fault injection into a 68340 microprocessor