PROBABILISTIC ANALYSIS OF REAL-TIME SCHEDULING OF SYSTEMS TOLERATING MULTIPLE TRANSIENT FAULTS

Risat Mahmud Pathan
Department of Computer Science and Engineering, BRAC University, Dhaka, Bangladesh.

Abstract

The influence of computer systems in human life is increasing and thereby increases the need for having reliable, robust and real-time services of computer systems. Avoidance of any catastrophic consequences due to faults in such systems is the main objective. This paper addresses the problem of finding a probabilistic measure of schedulability of real-time systems tasks in the presence of multiple transient faults. The main approach used in this paper is employing Temporal Error Masking (TEM) technique to achieve Node Level Fault Tolerance (NLFT) within the least common multiple of periods of a set of pre-emptively scheduled periodic tasks with at most f transient faults. Rate Monotonic (RM) scheduling is used to observe how the probability of system success, that is, the probability of meeting deadlines for all tasks is affected with the worst-case distribution of f faults

Keywords: Fixed-Priority Scheduling, NLFT, Real-Time Fault-Tolerant systems, TEM, Transient faults.

1. INTRODUCTION

Real-time systems that are being increasingly used in several applications are time critical in nature where a failure can pose a threat to human lives. The ability to tolerate faults in such hard *real-time* systems is crucial since a task can potentially miss its deadline when faults occur. The use of fault-tolerant computers in avionics was advanced in 1980s and the early 1990s, when so called fly-by-wire systems were introduced in military aircrafts and commercial airlines, such as the Airbus-320 [1] and Boeing 777 [2]. Real-time systems with high dependability requirements are traditionally being built with massive replication and redundancy. In certain classes of applications, due to space, weight and cost considerations it may not be feasible to provide space redundancy. Such systems need to exploit time redundancy techniques. In this paper, time redundancy is used to achieve fault tolerance in the presence of at most *f* transient faults. Several studies have shown that transient faults occur at much a higher rate than permanent faults [3]. In [4], measurements showed that transient faults are 30 times more frequent than permanent faults. In this paper, time redundant execution of tasks known as Temporal Error Masking (TEM) is used to mask at most *f* transient faults at node level, known as Node Level Fault Tolerance (NLFT). Moreover, in this paper, scheduling with probabilistic guarantee for a hard real-time system is addressed. The term 'probabilistic guarantee' means a scheduling guarantee with an associated probability. Hence, a guarantee of 99.95 % does not mean that 99.95 % of the deadlines of tasks are met. Rather it implies that the probability of all deadlines of all tasks being met during given period of operation is 99.95%.

2. RELATED WORK

One of the first scheduling mechanisms for fault tolerance deals with periodic tasks whose periods must be multiples of one another and the execution time of recovery tasks must be shorter than that of the original execution time was described by Liestman and Campbell [5]. In [6], a framework for light-weight node-level fault tolerance is presented where it is shown that NLFT-nodes may provide 55% higher reliability after one year and 60% higher mean time to failure (MTTF) compared to systems with fail-silent nodes. The fault injection experiment in [7] show that the percentage of correct results increased from 81% to 89% using temporal error masking. In [8] it is shown that, the percentage of detected errors increased from 93.9% to 97.2 %. In [9] Pandya and Malek analyze the schedulability of a set of periodic tasks that are scheduled using Rate Monotonic Scheduling and tolerate a single fault. In [10], the notion of probabilistic guarantee for fault-tolerant hard real-time systems is introduced. Another study, presented in [11], provides an exact schedulability test. In [12], a temporal-redundancy-based recovery technique is proposed that tolerates transient task failures where tasks have timing, resource, and precedence constraints. In [13], an appropriate schedulability analysis based on response-time analysis is proposed where recovery task may be executed at higher priority levels. In [14], a scheme is proposed that guarantees the timely recovery from multiple-faults assuming earliest-deadline-first scheduling (EDF) scheduling for aperiodic pre-emptive tasks. In this paper, similar approach as in [14] is made but for rate monotonic scheduling (RM) algorithm for a set of pre-emptive periodic tasks. Many papers have addressed the problem of tolerating transient faults with some restrictions as follows: only one fault is possible within some operational time, tasks periods are multiple of one another, and recovery task has smaller execution time or higher priority than that of the original task. In this paper, the response time of a task instances is found out considering only the worst-case distribution of *f* faults using an algorithm **RM-FT**. Moreover, a probabilistic measure of system success denoted by $P_{success}$ is determined using resultant data from fault injection experiment with 68340 microprocessor [6-8, 15].

3. TASK MODEL

The task set consists of $\boldsymbol{n}$ periodic tasks, $\Gamma=\{\tau_1, \tau_2, \ldots \tau_n\}$. Each task $\boldsymbol{\tau_i}$ has a period $\boldsymbol{T_i}$, and a relative deadline

D_i which is equal to its period. Each task τ_i has the worst-case execution time C_i and has a priority P_i. The highest priority task has the lowest period. The length of the planning cycle (PC) within which the tasks repeat themselves iteratively is the least common multiple of all task periods is defined as: **PC=lcm(P_1, P_2….P_n)**. Within one PC, one or more instances of task τ_i will execute. Each task instance is denoted by τ_{ij} where j is the **j**th instance of task τ_i. Γ_{all} is defined as the set of all task instances within PC. That is, $\Gamma_{all}= \{\tau_{ij} \mid i=1,2,\ldots n$ and $j=1,2,\ldots.\lceil \frac{PC}{Ti} \rceil\}$.

4. TEMPORAL ERROR MASKING (TEM)

The technique for TEM is, first two copies (primary copies) of a task instance is run, and if error is detected either by comparison of results of the two copies, or by timer monitor or by Error Detection Mechanism (EDM) a third copy of the task instance is run and then by majority voting of the three results, the result is either accepted or *omission failure* is occurred. In [15], three different cases in Fig. 1 were considered. According to [15], whenever two primary copies are faulty, assuming

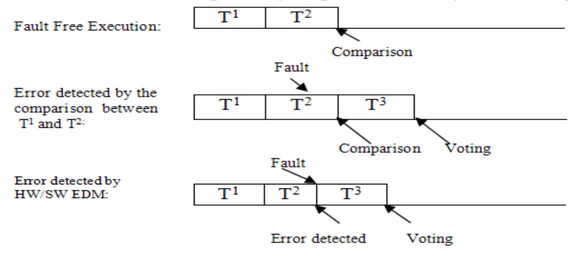
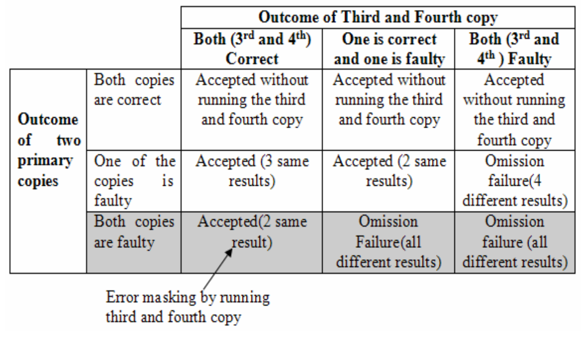


Fig. 1 Error detection and error recovery using TEM

two faults will not lead to the same error, running the third copy always leads to omission failure, hence the error is not masked. The approach in this paper to mask ***f*** errors, assuming multiple faults will not lead to same error, is first to run two primary copies of the same task instance. If an error is detected, run ***f*** more extra copies of the task instance. For example, in case ***f***=2, run two more extra copies (third and fourth) when error is detected. Then the four results are compared. If there are *at least* two matching results, the result is accepted. If we have all four different results, it will lead to omission failure as shown in Table I.

Table I Masking two errors by running two extra copies of a task instance.



		Outcome of Third and Fourth copy		
		Both (3rd and 4th) Correct	One is correct and one is faulty	Both (3rd and 4th) Faulty
Outcome of two primary copies	Both copies are correct	Accepted without running the third and fourth copy	Accepted without running the third and fourth copy	Accepted without running the third and fourth copy
	One of the copies is faulty	Accepted (3 same results)	Accepted (2 same results)	Omission failure(4 different results)
	Both copies are faulty	Accepted(2 same result)	Omission Failure(all different results)	Omission failure (all different results)

5. RESPONSE TIME ANALYSIS

Traditional response time analysis for RM scheduling as in [16] is not suitable when multiple faults are considered. The response time of task τ_i depends on the distribution of *f* faults within PC. If all *f* faults occur within the same task instance, the extra *f* copies are scheduled only for that task instance. If the *f* errors are occurred in many task instances, then for each of the erroneous task instances *f* extra copies will be scheduled. For at most *f* faults with total *m* different task copies of different tasks instances, one has to consider m^f different fault distributions if traditional response time analysis is considered. Instead of considering all possible distributions of *f* faults, in this paper, only *the* worst-case distribution of *f* faults is considered. In section 6, an algorithm **RM-IND** to find the response time for individual task instance in fault free environment is developed. By considering only the worst-case distribution of *f* faults within PC and using the result of **RM-IND**, the feasibility of a schedule of a task set is checked in another algorithm RM-FT is developed in section 7. At last, a probabilistic measure of system success for any task set based on RM-FT is determined in section 8.

6. RESPONSE TIME OF TASK INSTANCES IN FAULT-FREE ENVIRONMENT

To find the response time of individual task instances in fault free environment the following functions are defined:

RD(Γ,t): The set of ready task at time *t*. It represents all the task instances that are released at time *t*.

REL_LD(t): The release load in a fault-free environment at time *t*. It represents the total execution time required for the task instances in set **RD(Γ,t)** at time *t*.

Ψ(Γ,t): The amount of work still to be done at any time *t* in the fault-free environment. This function is defined recursively as follows:

$$\Psi(\Gamma,t)=\begin{cases} \textbf{REL\_LD}(0) & \text{if } t=0 \\ (\Psi(\Gamma,t-1) -- 1)+\textbf{REL\_LD}(t) & \text{if } t<PC \\ \Psi(\Gamma,PC-1) -- 1 & \text{if } t=PC \end{cases}$$

This binary operator "-- " with operands *a* and *b* is defined as: a--b = 0 if b>a, else a--b=a-b. Using the function **Ψ**, the following algorithm **RM-IND** is developed that finds the response time of individual task instances.

1.1 Algorithm: RM-IND

```
1 For t=0 to PC
2   If t>0 then
3    If [Ψ(Γ,t)-REL_LD(t)]decreases between time slot (t-1) and t
4            The highest priority task executes 1 time unit.
5            If(Highest priority task instance finishes execution)
6                     Record the finish time of the task instance
```

Fig. 2 Pseudocode for finding the response-time of each task instance in fault-free environment

The for loop at line 1 iterates PC times and at each time it checks in line 3 if there is a task eligible to execute in the time slot between (t-1) and t in fault free environment. When any particular task instance finishes execution in line 4, the finishing time is recorded in line 6.

6.2 EXAMPLE: SIMULATION OF RM-IND

For the following task set in Table II, the start time and the finishing time of each task instances is found in the Table III and the corresponding schedule is shown in Fig. 3 along with the value for ψ.

Table II Task Set

	Period (Ti)	Execution Time (Ci)
τ_1	3	1
τ_2	6	1

Table III Simulation of RM –IND for task in TableII

Iteration t	Ψ	Highest priority task	Comment
1	4 to3	τ_{11}	First primary copy of task instance τ_{11} executes for one time unit.
2	2	τ_{11}	Second primary copy of task instance τ_{11} executes for one time unit. Task τ_{11} finishes execution of both primary copies at t=2.
3	1	τ_{21}	First primary copy of task instance τ_{21} executes for one time unit.
4	2	τ_{12}	Task instance τ_{21} is pre-empted. Newly released task instance τ_{12} executes for one time unit (First primary copy).
5	1	τ_{12}	Second primary copy of task instance τ_{12} executes for one time unit. Task τ_{12} finishes execution both copies at t=5.
6	0	τ_{21}	Second primary copy of task instance τ_{21} executes for one time unit. Task τ_{21} finishes execution at time t=6.

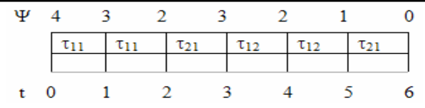


Fig. 3 RM Schedule for task set in Table II

The function ***fin***(τ_{ij}) is defined as the time when task instance τ_{ij} completes execution in RM-IND. So, ***fin***(τ_{11})=2, ***fin***(τ_{12})=5 and ***fin***(τ_{21})=6 as in Fig. 3. In next section, fault-tolerant algorithm **RM-FT** is developed considering the worst possible distribution of *f* faults.

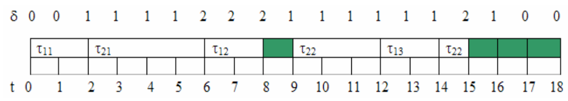
7. FAULT-TOLERANT ALGORITHM: RM-FT

The algorithm RM-FT in section 7.1 checks the schedulability of a set of tasks in environment where faults are likely. Inspired by the work in [14], the function $\delta_f(t,\Gamma_{all})$ in defined as the amount of extra work that still need to be done at time *t* due to *f* faults in an environment where faults are likely. For the task set in Table IV, the amount of extra work that still needs to be done is calculated in Fig. 4. If *f*=1, the amount of extra work when τ_{11} finishes execution is $\delta_1(2,\Gamma_{all})=1$ since if the task instance is in error, we need to run one more extra copy. Note that, $\delta_1(8, \Gamma) =2$, since when task instance τ_{12} finishes execution, the only fault if occurs in the task instance τ_{21} represents the worst-case distribution of one fault. Observe that, for *f*=1, the lowest priority task (τ_{22}) finishes before deadline since the amount of extra work at *t*=15 is $\delta_1(15, \Gamma)=2$ that becomes zero before the deadline of that task instance. But higher priority task instance τ_{21} cannot finish before deadline which is *t*=9 since at *t*=9, the extra work still to be done is $\delta_1(9, \Gamma)=1$. Hence the task set in Table IV is not schedulable.

<u>Theorem 1:</u> Given task set Γ_{all} and the lowest priority task τ_{ij} in Γ_{all} completes by its deadline which is $(T_i \times j)$ in fault tolerant schedule, if and only if, $\delta_f(\Gamma,t)=0$ for some t, ***fin***$(\tau_{ij}) \le t \le T_i \times j$.

Table IV Task set

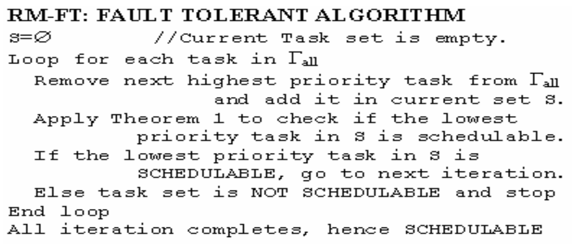
f=1	Period (Ti)	Execution Time (Ci)
τ_1	6	1
τ_2	9	2


δ 0 0 1 1 1 1 2 2 2 1 1 1 1 1 1 2 1 0 0
τ11 τ21 τ12 τ22 τ13 τ22
t 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
Fig. 4: The schedule with the value of δ at each time *t*

<u>Corollary 1:</u> A necessary and sufficient condition for the feasibility of the fault tolerant schedule for a given task set Γ_{all} for any distribution of *f* or less faults can be obtained by applying Theorem 1 to N task sets Γ_j for j=1,..,N where Γ_j contains the jth highest priority task from Γ_{all}.

7.1 ALGORITHM: RM-FT

Using Corollary1, the algorithm **RM-FT** for feasibility check of a schedule is developed and simulated in 7.2 and 7.3 using the two example task sets in Table V:



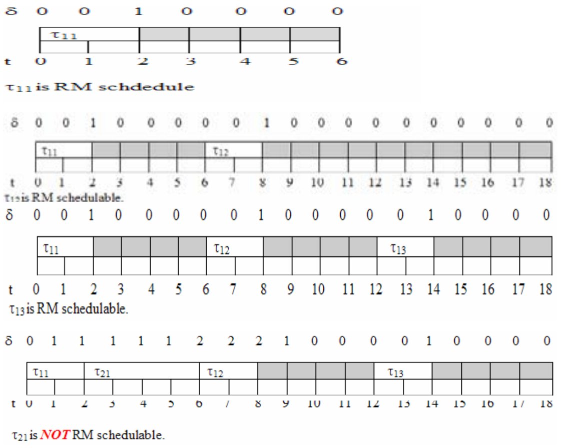
```
RM-FT: FAULT TOLERANT ALGORITHM
S=∅          //Current Task set is empty.
Loop for each task in $\Gamma_{all}$
  Remove next highest priority task from $\Gamma_{all}$
                  and add it in current set S.
  Apply Theorem 1 to check if the lowest
          priority task in S is schedulable.
  If the lowest priority task in S is
          SCHEDULABLE, go to next iteration.
  Else task set is NOT SCHEDULABLE and stop
End loop
All iteration completes, hence SCHEDULABLE
```

Fig. 5 Pseudocode for checking schedulability

Table V: Two task sets with same period but different execution time.

		EXAMPLE 1	EXAMPLE 2
	Period (T_i)	Execution Time (C_i)	Execution Time (C_i)
τ_1	6	1	1
τ_2	9	2	1

7.2 RM-FT SCHEDULE WITH δ VALUE For *f*=1 (EXAMPLE 1):


δ 0 0 1 0 0 0 0
τ11
t 0 1 2 3 4 5 6
τ11 is RM schdedule
δ 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
τ11 τ12
t 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
τ12 is RM schedulable.
δ 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0
τ11 τ12 τ13
t 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
τ13 is RM schedulable.
δ 0 1 1 1 1 1 2 2 2 1 0 0 0 0 1 0 0 0 0
τ11 τ21 τ12 τ13
t 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
τ21 is NOT RM schedulable.
Fig. 6 Task schedule for Example1 using RM-FT with δ value. The task set is not schedulable.

7.3 RM-FT SCHEDULE WITH δ VALUE FOR *f*=1(EXAMPLE 2):

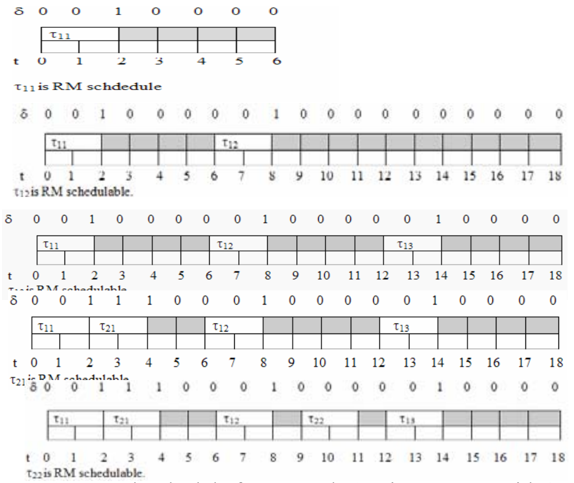


Fig. 7 Task schedule for Example 2 using RM-FT with δ value. The task set is schedulable

8. PROBABILITY OF SCHEDULABILITY:

8.1 PARAMETERS OF PROBABILITIES:

In this section, a probabilistic measure of system success in case of *f* faults is derived. The result of injecting faults into applications provides the parameters of certain probabilities that are given in the following Table VI. The values in the 3rd column are taken from experiment of injecting faults in a 68340 microprocessor [6-8, 15].

Table VI Parameters for the probabilities from fault injection experiment in 68340 microprocessor.

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 double execution (DE)	68 (of 373)	18%
P_T	Given that an error is generated, the error is detected by timer monitor.	19 (of 373)	5%
P_{EDM}	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 the timer monitor, the error is masked by TEM	18 (of 286)	6%
$P_{EDM,M}$	Given that an error is detected by an EDM, the error is masked by TEM	194 (of 286)	68%

8.2 PROBABILITY OF FAULT OCCURRENCE IN TASK τ_{ij}:

For a maximum of *f* faults, the probability of a fault occurrence in task instance τ_{ij} is **P**(F_{ij}) is defined as:

$$\mathbf{P}(F_{ij}) = \frac{(f+2)*Ci}{PC} \times \frac{PC - Rel(\tau ij)}{PC}$$

Here, $\frac{(f+2)*Ci}{PC}$ represents the task utilization of task τ_{ij} in case of faults within one PC. Since the probability of fault occurrence decreases as the tasks' completion time within one PC decreases, therefore, we scale the probability by multiplying $\frac{PC - Rel(\tau ij)}{PC}$ where Rel(τ_{ij}) is the release time of task τ_{ij}.

8.3 PROBABILITY OF SCHEDULABILITY Y_{ij} FOR A TASK τ_{ij}:

$$Y_{ij} = \begin{cases} 0 & \text{if task } \tau_{ij} \text{ is not schedulable by RM-FT} \\ 1 & \text{if task } \tau_{ij} \text{ is schedulable by RM-FT} \end{cases}$$

8.4 PROBABILITY OF ERROR MASKING:

In this analysis, the probability of error detection and masking by any one of the techniques (double execution, timer monitor, or EDM) is: ($P_{DE} \times P_{DE,M}$) + ($P_T \times P_{T,M}$) + ($P_{EDM} \times P_{EDM,M}$). When a fault occurs, and the fault leads to an error and the error is detected, then the error needs to be masked. Denote the probability error masking of all task instances by P_{Error} is defined as:

$$P_{Error} = \prod_{\tau_{ij} \in \Gamma_{all}} Y_{ij} \times \sum_{\tau_{ij} \in \Gamma_{all}} [P(F_{ij}) \times P_x \times (P_{DE} \times P_{DE,M} + P_T \times P_{T,M} + P_{EDM} \times P_{EDM,M})] \ldots \ldots (I)$$

Using the probabilities given in Table VI, ($P_x \times P_{DE} \times P_{DE,M} + P_T \times P_{T,M} + P_{EDM} \times P_{EDM,M}$)=0.120122

8.5 PROBABILITY OF NO ERROR MASKING:

Let's denote the probability of task execution without the need for error masking by $P_{NoError}$. This occurs when: (i) No fault occurs and the probability is 1-$\sum P(F_{ij})$ for $\tau_{ij} \in \Gamma_{all}$ (ii) Fault occurs but no error is generated, and the probability is [$P(F_{ij}) \times (1-P_x)$], and (iii)Fault occurs and error generated but error is not detected and the probability is $P(F_{ij}) \times P_x \times P_{ND}$. So, the probability when errors don't need to be masked for all task instances is:

$$P_{NoError} = [1 - \sum_{\tau_{ij} \in \Gamma_{all}} P(F_{ij})] + [\sum_{\tau_{ij} \in \Gamma_{all}} P(F_{ij}) \times (1-P_x)] + [\sum_{\tau_{ij} \in \Gamma_{all}} P(F_{ij}) \times P_x \times P_{ND}] \ldots \ldots (II)$$

8.6 PROBABILITY OF SYSTEM SUCCESS:

Let's denote the probability of the system success by $P_{success} = P_{NoError} + P_{Error}$ … …(III)

9. CALCULATING $P_{SUCCESS}$ WITH EXAMPLE TASK SETS:

In this section the probability of system success is calculated using different example task sets.

9.1 EXAMPLE 1

Table VII Task set

	Period (T_i)	Execution Time (C_i)
τ_1	9	1
τ_2	18	1
τ_3	36	1

If *f*=0, by running algorithm **RM-FT**, all tasks are schedulable. Hence, Y_{ij} =1 for all task instances. The probability of fault occurrence in each task instance within PC is given as follows:

$P(F_{11})$=.055555, $P(F_{12})$=0.04166, $P(F_{13})$=0.027777, $P(F_{14})$=0.01388, $P(F_{21})$=0.05555, $P(F_{22})$ =0.027777, $P(F_{31})$=0.055555

So, $\sum_{\tau_{ij} \in \Gamma_{all}} P(F_{ij}) = 0.277773$

Using equation (I), (II) and (III) , P_{Error} =0.033366, $P_{NoError}$=0.952778, $P_{success}$=0.986145.

If f=1, P_{Error} =0.05005, $P_{NoError}$=0.92916,

$P_{success}$=0.979219. If f=2, P_{Error} =0.07006, $P_{NoError}$=0.90083, $P_{success}$=0.97088. If f=3, P_{Error} =0.083417 and $P_{NoError}$=0.88195 and $P_{success}$=0.96536. If f=4, the task set is not schedulable using RM-FT. To see why, observe the following schedule in Fig. 8.

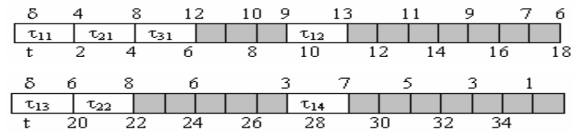

δ 4 8 12 10 9 13 11 9 7 6
τ_{11} τ_{21} τ_{31} τ_{12}
t 2 4 6 8 10 12 14 16 18
δ 6 8 6 3 7 5 3 1
τ_{13} τ_{22} τ_{14}
t 20 22 24 26 28 30 32 34

Fig. 8 Schedule for *f*=4 using RM-FT of task set in Table VII.

In the schedule, the lowest priority task is τ_{31} is schedulable. However, task τ_{12} finishes at time t=11 and the value of δ=13 (amount of extra work due to faults). It is not possible to complete the extra work δ before the deadline of τ_{12} which is t=18, since δ≠0 for any t such that ***fin***(τ_{12})=11 ≤ t ≤ (T_1 ×2) =18. So, task set in Table VII cannot be scheduled. Hence, Y_{12} =0. And, $\prod Y_{ij}$ =0. Hence, if *f*=4, P_{Error}=0 and $P_{NoError}$=0.858336 and $P_{success}$=0+0.858336=0.858336. Obviously, for the task set in Table VII, if f>3, $P_{success}$ =$P_{noerror}$.

9.1.1 DISCUSSION (EXAMPLE 1)

The graph in Fig. 9 shows that, as *f* increases, the probability of fault occurrence also increases. In Fig. 10, it can be seen that the probability of masking faults increases (P_{Error}) as we increase *f* since the system now employs error-masking capabilities to mask errors. However, in practice, no system is capable of tolerating an infinite number of faults. So, after a certain value of *f*, the fault masking capability will diminish to zero (see the column in Fig. 10 for *f*=4). In Fig. 11, the probability of system success without fault masking capability ($P_{NoError}$) decreases as *f* increases. This is because as more faults are likely, the system becomes more vulnerable to faults and the probability of schedulability decreases. Fig. 12 shows that as *f* increases, there is a decrease in the probability of overall system success ($P_{success}$). As more errors are occurring, there is an increase in probability of fault masking (P_{Error}) and there is also a decrease in the probability of success without fault masking ($P_{NoError}$). However, the sum of these two probabilities has a downward trend. This is because, as the number of maximum fault *f* occurrence increases, the probability of overall system success is more dependent on the fault masking capability of the system, which is limited for any practical system. So, for higher number of *f*, the probability $P_{success}$ is low.

9.2 EXAMPLE 2: The execution time of task τ_1 in Table VIII is increased by 1 time-unit in Example 2 than it is given in Example1 in Table VII.

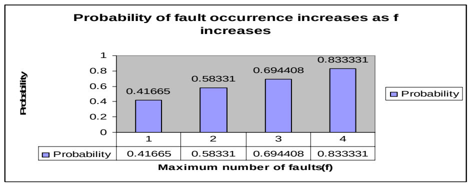


Fig. 9: The probability of fault increases as *f* increases

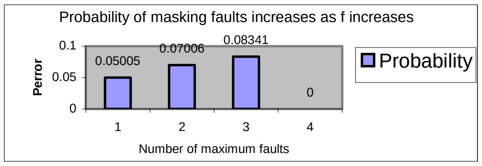


Fig. 10: Probability of fault masking increases

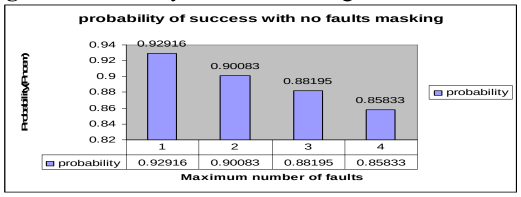


Fig.11: Probability of system success without fault masking decreases as *f* increases

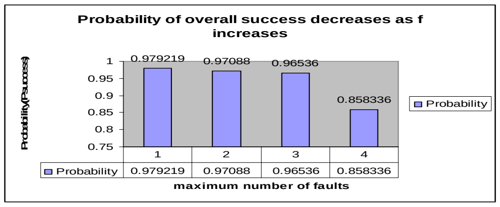


Fig. 12: Probability of system success decreases as *f* increases

Table VIII Task set

	Period (T_i)	Execution Time (C_i)
τ_1	9	2
τ_2	18	1
τ_3	36	1

Using (I), (II) and (III), it can be derived that, if f=1, P_{Error} =0.07507, $P_{NoError}$=0.89375, $P_{success}$=0.96882; if f=2, P_{Error}=0, $P_{NoError}$=0.858335 and $P_{success}$=0.858335. Intuitively, for this task set if f>3, $P_{success}$ =$P_{NoError}$.

9.2.1 DISCUSSION (EXAMPLE 2):

Observe that, Example 2 has larger execution time for task τ_1 than the execution time of task τ_1 given in Example 1. For *f*=1, the probability of $P_{success}$ in Example 2 ($P_{success}$=0.96882) is lower than the probability of $P_{success}$ in Example1 ($P_{success}$=0.979219). This is because with increased execution time there is less slack in the schedule and hence task with higher execution time contributes to the increased value of δ. Similarly, for *f*=2, the probability of $P_{success}$ in Example 2 ($P_{success}$=0.858335) is lower than the probability of $P_{success}$ in Example1 ($P_{success}$=0.97088). Consequently, it can be said that with higher task utilization, the less number of faults can be masked and thereby having a lower probability of system success.

9.3 EXAMPLE 3:

Table IX Task set

	Period (T_i)	Execution Time (C_i)
τ_1	9	1
τ_2	18	1
τ_3	36	5

9.3.1 DISCUSSION (EXAMPLE 3):

Observe that, Example 3 in Table IX has larger execution time for task τ_3 that the execution time given in Example 2 but both examples task set have same task utilization. For f=1, the probability of $P_{success}$ in Example 3 ($P_{success}$=0.964632) is lower than the probability of $P_{success}$ in Example 2 ($P_{success}$=0.96882). This is because, even if both have same utilization, tasks with large execution time runs for a long time in a fault tolerant schedule when extra/recovery copies need to be run. Hence, less slack is provided in the schedule. So, not only utilization but also the execution time of individual task is a major success factor of system schedulability.

10. CONCLUSION:

Meeting task deadlines is the main objective of hard real-time systems. If faults are likely, mechanisms must be employed to tolerate the faults if the system has to avoid catastrophic consequences. Use of redundancy is the solution for achieving fault tolerance. In this paper, probabilistic analysis of schedulability shows that, with increased *f*, probability of system success increases with fault masking capability up to a certain value of *f*, after which the probability of system success decreases, as the system can't tolerate unlimited number of faults. Moreover, not only higher task utilization but also individual task's execution time determines the probability of system success.

Running *f* extra copies requires more slack in the schedule which may not be available for many task sets in hard real-time systems. If the number of extra copies is decreased more slack would be available in the schedule and more task sets could be schedulable at node level. However, by running less than *f* extra copies, all errors may not be possible to mask at node level. If error could not be masked at node level, system level fault tolerance has to be employed. Future work could be to find a trade-off between system level and node level fault tolerance.

11. REFERENCES

[1] D. Briere and P. Traverse, " AIRBUS A320/A330/A340 electrical flight controls- A family of fault-tolerant systems", FTCS-23 The Twenty-Third International Symposium on Fault-Tolerant Computing, 22-24 June 1993, Toulouse, France, 1993.

[2] L. Andrade and C. Tenning, "Design of Boeing 777 electric system", IEEE Aerospace and Electronics Systems Magazine, vol. 7, pp 4-11, 1992.

[3] X. Castillo, S. R. McConnel, and D. P. Siewiorek. "Derivation and Calibration of a Transient Error Reliability Model". IEEE Trans. On Computers, C-31(7):658-671, July 1982.

[4] D. P. Siewiorek, V. Kini, H. Mashburn, S. McConnel, and M. Tsao. "A Case Study of C.mmp, Cm*, and C.vmp: Part 1: Experiences with Fault Tolerance in Multiprocessor Systems". Proceedings of the IEEE, 66(10):1178-1199, Oct. 1978.

[5] L. Liestman, R. H. Campbell, "A fault-Tolerant Scheduling Problem," IEEE Trans. Software Eng., vol.12, no.11, pp.1089-1095, 1986.

[6] Joakim Aidemark, Peter Folkesson, and Johan Karlsson. "A framework for node level fault tolerance in distributed real time systems," in *Proc. International Conference on Dependable Systems and Networks (DSN-2005)*, Yokohama, Japan, June 2005.

[7] Joakim Aidemark, J. Vinter, Peter Folkesson, and Johan Karlsson. "Experimental evaluation of time redundant execution for a brake-by-wire application," in *Proc. International Conference on Dependable Systems and Networks (DSN-2002)*, IEEE Computer Society Press, Washington DC, USA, June 2002, pp. 210-215.

[8] Joakim Aidemark, J. Vinter, Peter Folkesson, and Johan Karlsson. "Experimental dependability evaluation of the Artk68-FT real-time kernel," in *Proc. of the International Conference on Real-Time and Embedded Computer Systems and Applications*, Göteborg, Sweden, August 2004, pp. 625-645.

[9] M. Pandya and M. Malek. "Minimum Achievable Utilization for Fault-Tolerant Processing of Periodic tasks". Technical Report TR 94-07, University of Texas at Austin, Dept. of Computer Science, 1994.

[10] A. Burns, S. Punnekkat, L. Strigini, and D. R. Wright. "Probabilistic scheduling guarantees for fault tolerant real time systems". Technical report. Department of Computer Science, University of York, 1998.

[11] A. Burns, R. Davis, and S. Punnekkat. "Feasibility Analysis of Fault Tolerant Real time task sets". In 8th Euromicro Workshop on Real-Time Systems, Jun 1996.

[12] N. Kandasamy, J. P. Hayes, and B.T. Murray, "Tolerating Transient Faults in Statically Scheduled Safety Critical Embedded Systems". Proc. 18th IEEE symposium Reliable Distributed System(SRDS), pp. 212-221, 1999.

[13] G. M. de A. Lima, A. Burns, "An optimal fixed-priority assignment algorithm for supporting fault-tolerant hard real-time systems", IEEE trans. On Computers, 52(10):1332-1346, Oct, 2003.

[14] Frank Liberato, Rami Melhem, and Daniel Mosse. "Tolerance to multiple faults for aperiodic tasks in hard real time system". IEEE Trans. Computers 49(9):906-914, 2000.

[15] Joakim Aidemark. "Node-Level Fault Tolerance for Embedded Real-Time Systems". Ph. D. Thesis, Chalmers University of Technology, 2004.

[16] C. M. Krishna and Kang G. Shin, "Real-Time Systems", McGraw-Hill, 1997.