ISSN:0975-9646

Fault Tolerant Instantaneous Utilization Factor Schedulability of Real Time Tasks

Kshama Kulkarni *MIT Aurangabad*
Radhakrishna Naik *MIT Aurangabad*
Vaishali Shinde *MIT Aurangabad*

Abstract.This paper put forwards review on single processor scheduling algorithm to arrange periodic tasks for soft real time system. The characteristics of the off-line scheduler are that on the basis of rate monotonic algorithm, tasks are scheduled and accomplishment is done on the basis of Instantaneous Utilization Factor (IUF) scheduling algorithm. The tolerances of max *f* faults which can occur at any point of time equal to the largest relative dead line of the task set are considered in it. The likelihood of completion of the task set is checked based on the maximum workload requested by the higher priority jobs within the released time and deadline of the job of each task that is released at that instant. It provides a possible copy titled recovery copy to be executed if any fault occurs in the system to save the time. Tasks are alienated in to number of sub tasks and are executed distinctly which benefits the execution time by saving it and workload to be executed for the recovery copy.

Keyword: Real Time system, real time scheduling, transient faults, fault tolerance, feasibility analysis.

I. INTRODUCTION

Tasks in *hard* real-time systems have rigorous limits. (i.e. meeting task deadlines). The bargain of satisfying the timing constraints, the desired service of the system must be achieved. *Fault* occurs in to the service of the system may cause the eccentricity [1].To avoid disastrous corollaries, hard real-time systems have to satisfy all the time limits even in the occurrence of faults. Effective *Fault tolerant scheduling* can increase the chance to meet the given deadline. A fault-tolerant system is one that continues to perform its specified service in the presence of hardware and/or software faults. Designing fault-tolerant systems, mechanisms must be provided to ensure the correctness of the expected service even in the presence of faults. Due to the real-time nature of many fault-tolerant systems, it is operational that fault tolerant facility provided in such system does not bargain the timing constraints of the real time applications.

Fault Tolerance is achieved in the computer systems, by *space* or *time* [2][3]. *Time redundancy is used in this paper* [2][3] to get the operative f*ault tolerance.* Transient faults are considered in this paper. When *time redundancy* in case of fault is used, an error occurs and detected. This faulty task is either re-executed or a substitute copy called *recovery copy* is used for the execution. This execution of recovery copy may cause a deadline miss.

This paper consists, a periodic task scheduling is presented on uniprocessor for feasibility analysis of fault tolerant fixed priority scheduling under the assumption of multiple occurrences of faults .This analysis gives us the result of exact feasibility test that will ensure, deadline can be met if and only if the exact test is satisfied.

The proposed feasibility test can be used to any fixed priority scheduling algorithm like Rate Monotonic or Deadline Monotonic [5] policy. The relative deadline of each task should not be greater than its period.

Fault Model considered is same as used by R.M. Pathan in [3]. Fault may occur in any task at any time, even in the recovery operations. Possibilities of occurrence of faults are as follow [3.]

- The inter-arrival time of two faults must be separated by a minimum distance [6], [7], [8], [9], [10]
- At most one fault may occur in one task [10], [11]
- The recovery operation is simply the re-execution of the original task [6], [7], [10], [12]

This article considers, multiple transient faults occurred in hardware. Transient faults occurs frequently, common [13], [14] and due to high complexity their number is continuously increasing ,hence to achieve the fault tolerance arise due to software faults we use *Time Redundancy*. Due to transient faults a number of errors can occur in a small duration of time interval [14] so there is lot of applications where rate of occurrence of transient fault is high.

The study activates about tolerating multiple transient fault within that duration, R.M. Pathan's Exact feasibility test can determine whether a maximum of f faults can be tolerated within any time interval of length D_{max} where D_{max} is the largest relative deadline of a periodic task [3]. This paper consists a fault model in which the recovery copies are considered in case of occurrence of fault. Due to the occurrence of fault, recovery copy execution causes Extra workload and time of execution which in turn minimizes the total task execution.

IUF scheduling algorithm by Naik and Manthalkar [4] proposes *Task splitting* which can be used with the FTRM algorithm for all the task so that they should not miss their deadline.

FT-IUF algorithm considers the original copy as a recovery copy. Tasks are split in advance and executed individually so that time is saved and deadline can be reached.

Organization of the paper:

The rest of paper is organized as follows. Section II represents the related work. Section III represents the system model with specifications of terminologies and the flow of the model. Section IV represents the proposed work related to the fault tolerance. Section V shows performance evaluation of the proposed algorithm. Section VI represents the performance analysis part. Section VII represents the conclusion and future work for the proposed system.

II. RELATED WORK

Considering Real-Time scheduling fault tolerance problem has been tackled by many researchers for criticality of the tasks and its timing constraints.[3],[14],[8],[12].

The utilization bound for RM scheduling on uniprocessor was derived by Ghosh *et al.* backup. Utilization concept was sued for single and multiple transient faults [3][14].

Paper [12] by Liberto, Melhem and Mosse , exact feasibility test for tolerating transient fault was derived for EDF algorithm. They considered the recovery copy as a re-execution of primary copy.[3]

Burns, Davis, and Punnekkat derived an exact fault tolerant feasibility test for any fixed priority system using recovery blocks or re-execution [8][3]. This study was lengthened by using check point concept to recover the fault [10]. Based on re-execution, an optimal fixed priority assignment to tasks for fault-tolerant scheduling was proposed by Lima and Burns [9].

After the first development of FTRM algorithm by Mr. R.M. Pathan in 2010, there has been lot of development done in fault tolerance mechanism like one by Mr. J Yin, H Song, L Yuan & Q Cui on 'A real time fault tolerant scheduling algorithm for software / Hardware hybrid tasks. Later Mr. W Qui, Z Zheng, X Wang and X Yang developed an efficient fault tolerant scheduling algorithm for periodic real time tasks in heterogeneous platforms. Analysis and design of fault tolerant scheduling for real time tasks on earth-observation satellite was developed by X Zhu, J Wang, X Qin. J wang, X Zhu, W Bao developed a real time fault tolerant scheduling based on primary back up approach in virtualized clouds. Fault tolerant RT scheduling algorithm for tolerating multiple transient faults by R M Pathan presents a fault tolerant real time scheduling algorithm, RM-FT, by extending the rate monotonic scheduling for real time system.

2.1 Fault Tolerant Scheduling on Uniprocessor Platform

This paper follows the basic mechanism of FTRM algorithm developed by R. Pathan in [1]. It divides the complete model in to 3 different groups and derived the concept of FTRM algorithm. Detaching the calculation part of Fault removal ,the basic assumptions are considered in this paper.

A Task Model considers the fault tolerant scheduling of *n* independent periodic tasks on uniprocessor. Each task generates a sequence of jobs. Priority of tasks is assigned as per RM scheduling policy. i.e. high priority task will be executed first.

Fault model informs about the nature of fault. Transient faults are considered here. It may be of software or hardware. It is assumed that transient faults are not long lived and may appear again. Main executing task is called a primary copy. When a fault arises it creates an alternate copy called recovery copy automatically. And re executes the task with the same priority as of the primary copy.

Fault tolerant mechanism executes the primary copy at normal condition. When a fault occurs a recovery copy is automatically executed. If one fault occurred on that task a single recovery copy is executed if faults are multiple then the recovery copies equivalent to number of fault are executed.[1]

At most *f* faults are considered in the maximum interval of length of a task.

2.2 IUF Scheduling

IUF Scheduling algorithm, developed by Naik and Manthalkar, divides the task in to number of quantum. It makes use of IUF to split the task in to number of small units based on the priority of highest utilization factor first. It may change the original priority of the task to be executed by RM scheduler. But task which is not feasible by Rate Monotonic scheduler is feasible by IUF scheduler.

III. SYSTEM MODEL

This paper combines the concepts of IUF algorithm and FTRM algorithm to find the solution for the problem of feasibility of all tasks to be executed even in case of occurrence of faults.

3.1 Concept of Task Split.

In the FTRM algorithm, the in the normal execution task units get executed in the regular way of RM scheduling. If fault occurs the execution stops and the recovery copy of the task is automatically created gets executed after the point where it was stopped its execution.

Thus it loses the work of execution before fault occurred. WCET required completing the task after fault is more. Hence lower priority task may not get enough time to complete the tasks within its deadline. Solution to this problem is to split the task in to number of units.

Example 3 units of task is split in to 3 different units

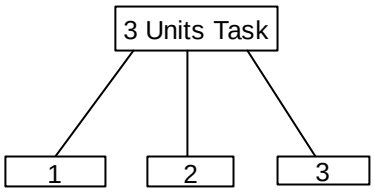

Figure 1: Example of Task Split.

IUF algorithm helps us to divide the task into number of quantum. This quantum will minimize re-execution time used to execute the recovery copy created at the occurrence of fault.

3.2 Notations:

P_i = Period of invocation of i^{th} task.
C_i= Computation time of i^{th} task.
U_i = Utilization of i^{th} task.
U_j^i = Instantaneous utilization of ith task.
C_j^i = Instantaneous remaining computation time of i^{th} task for j^{th} quantum iteration.
P_j^i = Instantaneous remaining period of i^{th} task..
Q_i= Quantum for which of i^{th} task is applied to CPU for the execution.
Q = Total sum of quantum of execution of all tasks for tat instant of time.

3.3 Division of System Model

To Find the fault tolerance and to form a system model, study is divided into following parts [1]:

1. Task Model
2. Fault Model
3. Fault tolerant mechanism

3.3.1 Task Model.

We consider in a fault tolerant scheduling.
Periodic Task: *n* independent tasks
Task Set $\Gamma = \{\tau_1, \tau_2, \ldots . \tau_n\}$ on Uniprocessor.

For a Task τ_i
$Period = T_i,$
$Relative\ Deadline = D_i,$
$WCET = C_i.$
For $Task\ \tau_i$, $Its\ D_i < T_i,$

All the tasks are assumed to be released at zero.
Each Task $\mathcal{T}_i$ generates a sequence of jobs.
$jth\ job\ of\ Task\ T_i = \tau_{i,j}\ \ For\ j = 1,2, \ldots . \infty$

Step 1: Initially for given task set, calculate CPU utilization of each task using formula [2]

$U_0^i = C_0^i / P_0^i$
$U_0^i =$ Initial utilization of i^{th} task.
$C_0^i =$ Initial Computation time.
$P_0^i =$ Initial period of invocation.

Based on utilization [U_0^i] the task which is having higher value of utilization is mapped for the CPU.

Step2: Now let us calculate the value of $U_{1\ 1}^i$ i.e. next Instantaneous utilization Factor of i[th] task.

$C_1^i = C_0^i - Q_i$
$P_1^i = P_0^i - Q$
$\Sigma Qi = Q.$
$U_1^i = C_1^i / P_1^i$

$U_1^i =$ Instantaneous utilization of i^{th} task.
$C_1^i =$ Instantaneous computation time of i^{th} task.
$P_1^i =$ Instantaneous period of invocation of i^{th} task.

Again the task which is having highest instantaneous utilization will be having highest priority of execution for second iteration quantum.
Likewise, calculate

$C_j^i = C_{j-1}^i - Q_i$
$P_j^i = P_{j-1}^i - Q$
$\sum Qi = Q.$
$U_j^i = C_j^i / P_j^i$
$j = PTC\ end\ point.$

3.3.2 Fault Model.

Schedulability of fault model is done by a fault model. Fault tolerant algorithm F.P.S. considers at most *f* fault occurrence and their recovery. Time interval of fault occurrence of *f* faults is $D_{max.}$ Faults are transient in nature and occur in software or hardware. Hard ware faults are assumed to be short-lived. Software faults are not permanent in nature. If it is permanent then it is removed by using time redundancy, by using recovery blocks.

3.3.3 Fault Tolerant Mechanism

It is based on time redundancy.

Time redundancy:

When a fault is detected faulty task is simply re-executed when a job of a task is for the first time, it is called a *Primary copy* of the task. Re-execution of the task or executing recovery block is called *recovery copy* of that task. It is executed after the CPU mapping when fault arise, that corrupted copy is removed and recovery copy is executed at the time of execution of task at either at free slot or by shifting lower priority task to the next free slot of processor.

IV. PROPOSED WORK

4.1 Architecture

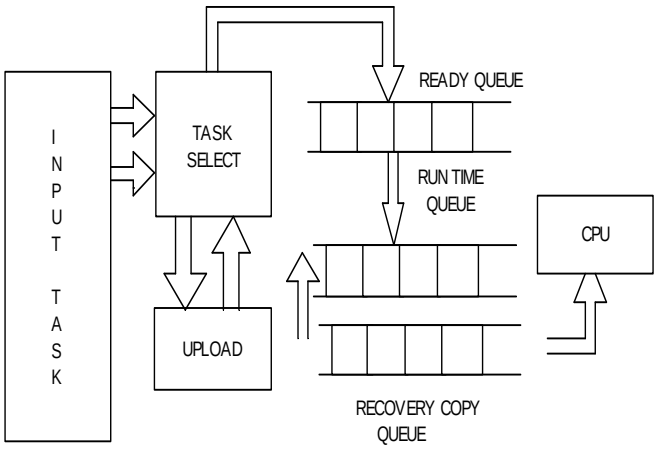

Figure 2: Architecture of FT-IUF Scheduler.

Fault Tolerant IUF scheduling Algorithm.

1. Take input of tasks containing period, deadline.
2. Calculate the utilization factor for each task.
3. Check the schedulability for tasks according to their utilization
4. Generate CPU mapping for tasks.
5. Execute the tasks according to the highest instantaneous utilization.
6. Meanwhile if a fault occurs then a recovery copy is created automatically so that the task can be re-executed.
7. Check the processor Idle time and shift the recovery copy of the task in that slot.
8. If the free slot is beyond its deadline then shift the lower priority task to the free task and allocate the slot to the task.

Assumptions.

- Transient Faults are Short Lived and may re-occur on the same task.
- Removing and reloading time of recovery copy are assumed equal to 0.
- Primary copy is re-executed as a recovery copy.

V. PERFORMANCE EVALUATION

5.1 Case Study I: FT-IUF scheduling Algorithm

Consider the following task set
$T_1 = (3,10)$, $T_2 = (3,15)$, $T_3 = (9,40)$

1. Calculate initial utilization using equation
$U_0^i = C_0^i / P_0^i$

Table 1: Description of Task Set of Case Study 1:

T_i	C_i	P_i	U_i
T_1	3	10	0.3
T_2	3	15	0.2
T_3	9	40	0.225

After first step, in Table 1 it is observed that T_1 has the higher initial utilization so is mapped for execution.

For quantum 1,
Calculate the new values of C_1, P_1, and U_1 using the above values.
Values can be calculated as follows:

$C_1^1 = C_0^1 - Q_1$
$P_1^1 = P_0^1 - Q$
$\sum Q_i = 1$

$$U_1^1 = {C_1^1 - Q_1}/{P_1^1 - Q}$$
$$= {3-1}/{10-1}$$
$$= 0.22$$
$$U_1^2 = {C_1^2 - Q_2}/{P_1^2 - Q}$$
$$= {3-0}/{15-1}$$
$$= 0.21$$
$$U_1^3 = {C_1^3 - Q_3}/{P_1^3 - Q}$$
$$= {9-0}/{40-1}$$
$$= 0.23$$

Now we get the table as

Table 2: Task Set After first quantum of Case Study1.

T_i	C_i	P_i	U_i
T_1	2	9	0.22
T_2	3	14	0.21
T_3	9	39	0.23

As U_3 has the highest value it has the higher priority and quantum Q_3 will be set to 1.

Thus repeat the steps up to 40 as the T_3 Task has the highest deadline in our E.g.. The final mapping of the tasks will be like in the figure.

Example. in case study can be simulated using IUF scheduling as shown in Fig. 3

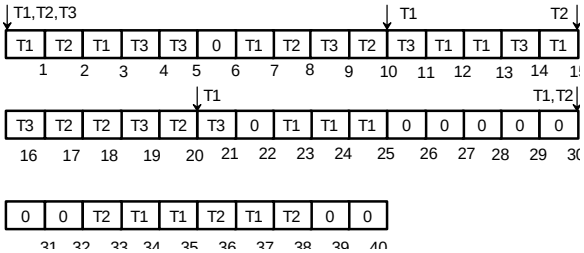

Figure 3: IUF Scheduling.

- IUF Scheduling algorithm splits the a task into different subunits.
- Schedules the sub task as per the higher utilization first.
- 0 indicates the idle period of processor.

Consider the arrival of fault at the following tasks instances.

- F_1 = Second instance of $T_{1,1}$.
- F_2 =Second instance of $T_{1,1}$.
- F_3 =Second instance of $T_{2,1}$
- F_4 =First instance of $T_{2,2}$
- F_5 =Second instance of $T_{2,2}$
- F_6 =Second Instance of $T_{1,3}$
- F_7 =Second instance of $T_{1,3}$

Now let the fault F_1, occurred at (3) T_1, So it will check its deadline, and if there is any idle period of processor is found it executes its recovery copy in that slot, otherwise it will shift lower priority tasks to later idle periods. Thus faulty task will change its sequence of priority as it suffers due to fault. But all tasks get chance for the complete execution.

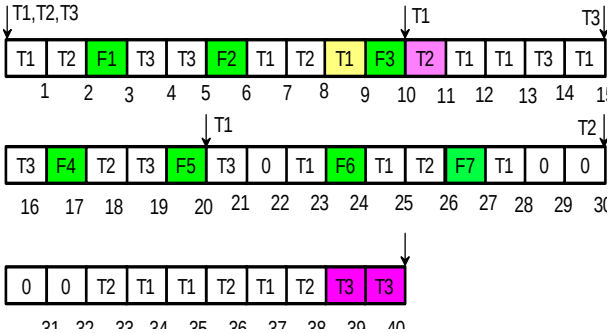

Figure 4: Occurrence of fault.

Shuffled Tasks are as follows:

- F_1 on T_1 $(2-3)$shift to $0(5-6)$ $=T_1$
- F_2 on T_1 $(5-6)$shift to T_3 $(8-9)$ $=T_1$
  $T_3(8-9)$ shift to $0(38-39)$ $=T_3$
- F_3 on $T_2(9-10)$shift to $T_3(10-11)$ $=T_2$
  $T_3(10-11)$shift to $0(39-)$ $= T_3$
- F_4 on T_2 $(16-17)$shift to $0(21-22)= T_2$
- F_5 on T_2 $(19-20)$shift to $0(25-26)=T_2$
- F_6 on T_1 (23-24) shift to $0(26-27)= T_1$
- F_7 on T_1 $(26-27)$ shift to $0(27-28)= T_1$

Table 3: Description of Task Set of Case Study 2.

Task Name	Period	Deadline
T_1	2	6
T_2	2	10
T_3	3	15

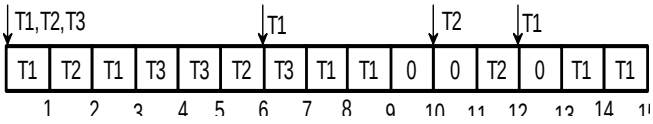

Figure 5: IUF Scheduling.

Consider the arrival of fault at the following tasks instances.

- F_1= First instance of $T_{1,1}$..
- F_2=Second instance of $T_{2,1}$

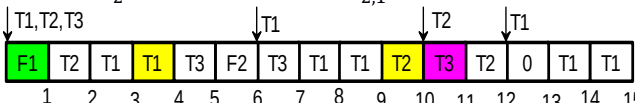

Figure 6 : Removal of fault.

After removal of fault shuffled tasks:

- F_1 on T_1 $(1-2)$shift to T_3 $(3-4)$= T_1
  T_3 $(3-4)$ shift to $0(10-11)$= T_3
- F_2 on T_2 (3-4) shift to $0(9-10)$= T_2

Table 4: Description of Task Set of Case Study3.

Task Name	Period	Deadline
T_1	3	10
T_2	4	20
T_3	7	39

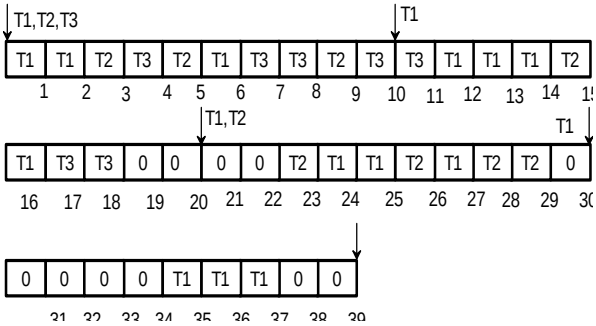

Figure 7: IUF Scheduling.

Consider the arrival of fault at the following tasks instances.

- F_1= Second instance of $T_{1,1}$..
- F_2=Second instance of $T_{1,1}$.
- F_3=Third instance of $T_{2,1}$..
- F_4=Third instance of $T_{2,1}$.
- F_5=Second instance of $T_{2,2}$.,
- F_6= First instance of $T_{2,1}$

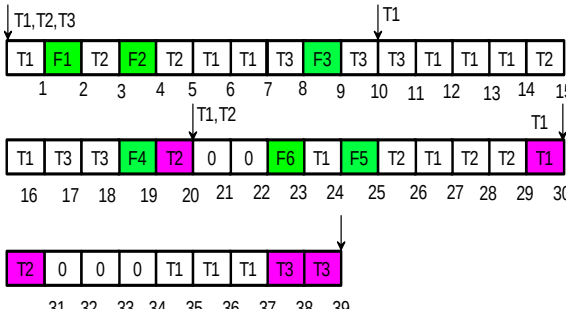

Figure 8 : Removal of Fault.

Shuffled Tasks are as follows:

- F_1 on T_1 $(1-2)$ shift to T_3 $(3-4)$ =T_1
  T_3 $(3-4)$ shift to $0(37-38)$ =T_3
- F_2 on T_1 $(3-4)$shift to T_3 $(6-7)$ =T_1
  T_3 $(6-7)$ shift to $0(38-39)$ =T_3
- F_3 on T_2 $(8-9)$ shift to $0(18-19)$= T_2
- F_4 on T_2 $(18-19)$ shift to $0(19-20)$= T_2
- F_5 on T_1 $(24-25)$shift to $0(29-30)$ = T_1
- F_6 on T_2 $(22-23)$ shift to $0(30-31)$= T_2

VI. PERFORMANCE ANALYSIS.

Proposed algorithm FT-IUF is compared with FTRM algorithm. Total faults occurred in the algorithm of each case study are marked. Then a comparison between actual time and the output of two algorithms is given in the table.

Table 5: Performance Analysis of Case Study1.

Number of Faults	**Actual period. Number of fault occurrence**	**FTRM Fault occurrence**	**FT-IUF Fault occurrence**
Total Execution time	30%	62.5%	47.5%
Average Execution time	0	32.5%	17.5%

Table 6: Performance Analysis of Case Study2.

Total Execution time	**26.66%**	**46.66%**	**40%**
Average execution Time	**0**	**20%**	**13.66%**

Table 7: Performance Analysis of Case Study3.

Total Execution Time	**30.76%**	**56%**	**43.58%**
Average execution Time	**0**	**25.24%**	**12.82%**

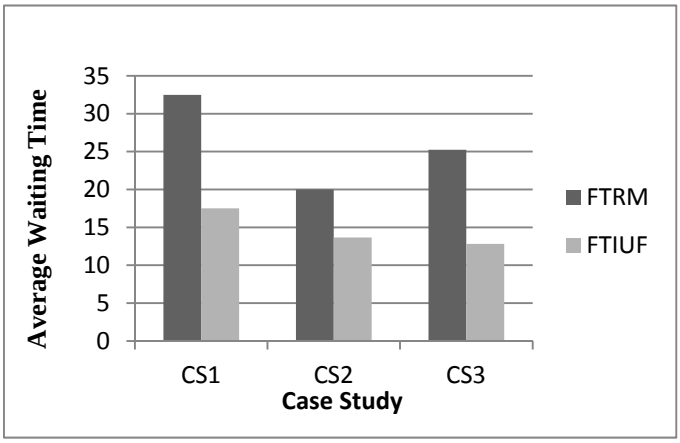

Figure 9: Average Waiting Time Analysis

VII. CONCLUSION AND FUTURE SCOPE:

FT-IUF algorithm proposes a task split mechanism which saves the work load of the task as recovery copy and execution of primary copy both are divided in to small pieces which will require less execution time as compared to FTRM algorithm.

Proposed work of FT-IUF algorithm shows the quantitative result as compared to FTRM algorithm as the task splitting is getting advantageous for executing the recovery copy in case of fault occurrence. we have presented the result in the percentage form as FTRM algorithm are 32.5%,20% and 25.24% and of FT-IUF algorithm are 17.5%, 13.66% and 12.82% i.e. lesser than the FTRM algorithm which will be improving the performance as the lower priority tasks are now feasible for the execution

Hence task can be feasible in case of fault occurrence. Feasibility of the system increase and hence performance of the system is improved.

FT-IUF algorithm can be implemented further for aperiodic tasks in the real time scheduling. When an aperiodic task appears to the algorithm then its fault tolerant policy could be designed in the future scope.

REFERENCES

[1] H. Aydin. Exact Fault-Sensitive Feasibility Analysis of Real Time Tasks. *IEEE Trans.on Comp.*,56(10), Year 2007, PP 1372-1386.

[2] I. Koren and C.M.Krishna. *Fault-Tolerant System.* Morgan Kaufmann, Year 2007.

[3] Risat Mahmud Pathan and Jan Jonsson," Exact Fault-Tolerant Feasibility analysis of Fixed –Priority Real Time tasks". DOI 10.1109/RTCSA.Year 2010,PP 27-48.

[4] Radhakrishna Naik1, R.R.Manthalkar2 "Instantaneous Utilization Based Scheduling Algorithms for Real Time Systems", *Pune University1, SGGS Nanded2,Year.* 2011, PP.654-662.

[5] J. Y. T. Leung and J. Whitehead. On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks. *Performance Evaluation*, 2:Year.1982, PP 237–250.

[6] S. Ghosh,R.Melhem,and D.Mosse. Enhancing Real-Time Schedules to Tolerate Transient Faults.In *Proc.of the RTSS,*Year 1995,PP 120-129.

[7] M. Pandya and M. Malek. Minimum Achievable Utilization for Fault-Tolerant Processing of Periodic Tasks *IEEE Trans. on Comp.*, 47(10):Year.1998, PP 1102–1112.

[8] A. Burns, R. Davis, and S. Punnekkat. Feasibility Analysis of Fault-Tolerant Real-Time Task Sets. In *Proc. of the ECRTS*, Year. 1996, PP 522–527.

[9] G. M. de A. Lima and A. Burns. An Optimal Fixed-Priority Assignment Algorithm for Supporting Fault-Tolerant Hard Real-Time Systems. *IEEE Trans. on Comp.*, 52(10):Year.2003, PP 1332–346.

[10] S. Punnekkat, A. Burns, and R. Davis. Analysis of Check pointing for Real-Time Systems. *Real-Time Systems.*, 20(1):Year.2001, PP 83–102.

[11] C.-C. Han, K.G. Shin, and J. Wu. A Fault-Tolerant Scheduling Algorithm for Real-Time Periodic Tasks with Possible Software Faults. *IEEE Trans. on Comp*., 52(3): Year 2003, PP 362-372.

[12] F. Liberato, R. Melhem, and D. Moss´e. Tolerance to Multiple Transient Faults for periodic Tasks in Hard Real-Time Systems. *IEEE Trans. on Comp.*, 49(9):Year.2000, PP 906–914.

[13] R. K.. Iyer , D. J. Rossetti, and M.C. Hsueh. Measurement and Modeling of Computer Reliability as Affected by System Activity. *ACM Trans. on Comp. Syst.* 4(3)Year 1986, PP 214,237.

[14] A. Campbell, P. McDonald,and K. Ray. Single Event Upset Rates in Space. *IEEE* Trans. on Nuclear Sci.,39(6), Year 1992, PP 1828-1835.

[15] S.Ghosh,R.Melhem,and D.Mosse, and J. S. Sarma Fault-Tolerant Rate Monotonic Scheduling.Real Time Systems15(2) Year 1998,PP 149-181.