ISSN (Print) : 0974-6846
Indian Journal of Science and Technology, *Vol 8(15), 55793, July 2015* ISSN (Online) : 0974-5645

Approaches for Transient Fault Tolerance in Multiprocessor - A State of Art

M. Shanmugasundaram[1*], R. Kumar[2] and Kittur Harish Mallikarjun[1]

[1]School of Electronics Engineering, VIT University, Vellore-632014, Tamil Nadu, India; mshanmugasundaram@vit.ac.in

[2]Wipro Technologies, Chennai, Tamil Nadu, India; rajagopal.kumar@wipro.com

Abstract

Objectives: This aim of this paper is to analyse different approaches for transient fault tolerance like PBS, RM-FT, FSP, BCE and MWFD. **Methods:** Transient faults are emerging as a critical issue in the reliability of real time systems. Besides multiprocessors being an apparent part of various booming technologies now-a-days, it becomes evident to make these systems susceptible, reliable and consistent to transient faults. Thus, to address the increasing susceptibility of multiprocessors systems to transient faults, different approaches are analysed to counter transient faults. **Findings:** To identify the optimal method for various constraints like slack time, reduction in time space and reconfiguration etc. **Applications:** This comparative analysis will help for us to identify the optimal approaches for tolerating the transient fault in critical real time systems.

Keywords: BCE, EDF, EH-EDF, EIT, Fault Tolerance, FSP, LTH, MWFD, PBS, Reliability, RM-FT, Transient Faults

1. Introduction

Consider a system with periodic task, which will request the resource at regular interval of time. Also assume a fault tolerance system, in which tasks may fail due to transient fault or inter failure. Here, a technique based on duplication of task; such that all jobs of each task meets their deadline by Probabilistic approach has been followed, i.e. minimum one task should meet deadline without fail.

In order to reduce the processor number, we can consider that each should have only two copies. One is primary, which must be always executed and another one is secondary copy (backup). When primary copy has been finished, it will force the secondary one to terminate which will be useful in saving resources.

In critical application we required a dependable situation to ensure that system will meet their requirement since prediction of occurrence of fault is not possible. While computing we frequently experience that task should execute periodically.

For real time periodic task, different schemes are proposed to support fault tolerance scheduling as per Figure 1.

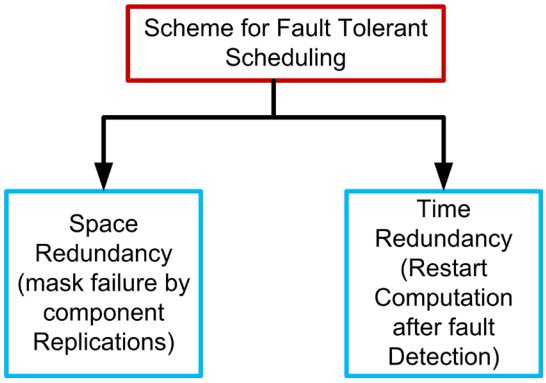

Figure 1. Scheme for Fault tolerant scheduling.

* *Author for correspondence*

As system becomes more complex, the design and implementation issues increase. Probability of fault increases as number of components increase in overall system. As compared to application level where time limit is not a big factor the real time system fault issues is more likely to be handle within a time limit.

In this paper we have analysed about some of the fault tolerant real-time scheduling algorithms such as RM-FT probabilistic algorithm, dynamic optimal solution, check pointing based technique, Primary/Backup (PB) algorithm and linear time heuristic algorithm. Other for periodic tasks which we will discuss is BCE algorithm, CAT algorithm and EIT algorithm. We have also proposed an energy efficient scheduling scheme for deterministic fault-tolerance system in homogeneous multiprocessor system.

2. Different Methodology

2.1 Scheduling Scheme

We can define a periodic real time task t_i as pair (C_j, T_j), where C_j is computation time and T_j is period[6]. These tasks are independent to which other. The utilization of periodic task is given as $u_j < C_j/T_j$ while total utilization of set of task is summation of utilization of all period task, which is given by $U = \sum Uj$ for $1 \leq j \leq n$. This is best suited for distributed real time systems which are interconnected by communication links and also assumed that processor has hardware fault only which can be transient or permanent or independent[8].

Assume that second processor cannot be failed before recovery of first processor system failure. If task arrived, which is not schedule previously at that time that particular task will be rejected and will not allows to go for execution at later stage, i.e. after completion of another task.

Following two techniques are used to achieve high schedulability.

2.1.1 Back Up Overloading

Schedules the backup during the multiple primary task execution in order to utilize processor time.

2.1.2 De-allocate

The reserved resources for backup copy after the successful completion of primary task.

For tolerating the fault, we should confirm the reservations of resources for backup copy. It must have a different strategy for scheduling. Note it is important to confirm that the scheduling of primary and backup copies will not increase the running time significantly, which is proportional to the task window and the average execution time of tasks[14].

The structure of scheduling scheme algorithm consists of task classifier, task allocator, scheduler and task manager. From above blocks task classifier classifies task into two classes, task allocator and scheduler will allocate the execution time while scheduler will schedule the priority of task[8].

Task classifier divide real time task set T into two categories

$$T_L = \left\{ t_i \in T \mid \frac{C_i}{T_i} \geq \frac{1}{2} \right\} \quad (1)$$

$$T_H = \left\{ t_i \in T \mid \frac{C_i}{T_i} < \frac{1}{2} \right\} \quad (2)$$

Above equation shows that Task TL has high task utilization than TH and task TH has higher execution time than Ti ,due to this, we can assign backup copy to the same processor so that transient fault will not occur. After this we assigned all TH task to the processor with their backup copies. Fixed priority is assigned to all the task, if backup copy of any of the task is not assigned to the same processor it the task will be assigned to the T_L. After adding all tasks to T_L the task is assigned to the processor that is available at that particular time with fixed priority scheduling scheme. But in this case the backup copies of task are assigned to different processor, and previously backup copies of T_H are replaced with backup copies assigned T_L to avoid permanent fault.

2.2 Dynamic Optimal Solution

Task is assumed to be fault free if it completes its execution within the specified time and gives proper results, otherwise the task is faulty[2].

Consider the task $T = (a, d, c)$ where a represents arrival time of task, d represents deadline and c refers to the worst case execution time. Let task window = $(d - a)$ and window ratio is given by

$$W = (d - a)/c \quad (3)$$

Assume $W \geq 2$. Fault tolerance can be guaranteed if the faults that occur are separated by Δf time interval.

Let the number of tasks to be scheduled be '*n*' and the queue of them be represented by *QT*, task starting time is *t0*. If the service is fault free, task will be executed within specified deadline that is if the following condition is satisfied,

$$t0 + \sum_{i=1}^{jn} c_i \leq d$$

While servicing, if any task is faulty, extra time is needed for recovery of the faulty task, so the time taken will be greater than

$$\sum_{i=1}^{n} c_i$$

Let t_j be the time that j tasks present in queue completes their function properly in presence of error, then T_j will properly execute within specified interval if $t_j < d_j$ [2].

The following equation will represent the estimate for time t_j, in which the task is re executed.

$$t_j = t0 + \sum_{i=1}^{n} c_i + \sum_{m=1}^{b} \beta m \quad (4)$$

β1…..βm are slot lengths of B1,B2,…Bm slots.

$$\beta m + \sum_{Tu \in Bm} Cu \leq \Delta f \quad (5)$$

$$\beta m \geq max \{Cu | Tu \in \beta m\} \quad (6)$$

Equation (4) gives time needed to execute tasks in queue in presence of faults. βm refers to the backup slot for re-execution. Equation (5) gives from βm slot almost one task to be re-executed. Equation (6) specifies that time required for re-execution of task which is in slot Bm will not be the time more than βm.

Finally we can say that:

Let QT=[T1,….Tn] be the queue of tasks to be scheduled. The task should be re- executed if the fault is detected during execution. If tj which will be calculated from above equations satisfies condition that execution time should be less than or equal to specified deadline (tj ≤ dj) then and if it is assumed that in the interval Δf, at most only one fault occur, then all tasks in queue will execute within specified deadline.

2.3 Feasibility Shortest Path Algorithm

It provides a mapping technique between a normal queue and fault-tolerant queue. It provides the assurance that all tasks can be completed within their deadlines if more than one fault will not occur within the short period Δf. By providing the shortest possible queue. Optimal mapping can be achieved by considering all possible placements of backups such that it should satisfy all conditions. In this algorithm a graph which contains various nodes is created then all paths in the graph must be explored[12]. The feasible shortest path can be achieved in graph G by satisfying the condition Vi, j < Di (where Vi, j is the time required) at each node Ni, j (Ni, j is node position). To achieve the optimal FSP, for a given queue with n tasks the backup will satisfy all the condition of fault tolerance regarding execution[2]. In the presence of multiple assignments of backups, the algorithm must have the capability to find the assignment that will reduce the queue length. After considering the1st Ti tasks in the queue, there are only two methods of placing the (i+1)th task in the queue (before or after the last backup). By exploring these paths, we can an optimal assignment. A dynamic programming approach can be used to get an optimal placement of backups,. Some set of tasks and their backups can be well organized at each node to detect the actual placement of back up if needed. This algorithm would be sufficient for a queue of non-real-time tasks. Whereas, in real-time applications, the deadlines will be taken into account. If the deadline of task Ti is not met when it is added to the queue. This algorithm will return fault tolerant guarantee by the presence of single feasible assignment. If the backup's position in the queue is to be known, the edges on the shortest path can be followed to get at which points in the queue backups need to be inserted with the worst case complexity O (n^2).

More computation power should be sacrificed while missing the scheduling of all critical tasks with fault tolerance. The accepted task must be completed before the deadline even in the presence of faults. In dynamic systems, the arrival time of tasks cannot be predicted. The task can be generated at arbitrary times due to events external to the system such as the system operator activating a task and the task has to be scheduled as soon as it arrives. FSP can be used for scheduling dynamically arriving tasks if the deadlines are not important such that there is sufficient time to run algorithm.

2.4 Linear Time Heuristic Algorithm

Initially δ = β = t0 = 0

Where length of queue between two backup slots will be

tracked by δ, length of last backup will be specified by β and tj represents queue length when j numbers of tasks are considered. The algorithm is as follows:

{

for $j = 1\ldots..,n+1$ do

if ($\delta + Cj + max\ \{\beta, Cj\} \leq \Delta f$)

then $\delta = \delta + Cj$;

$t(j) = t(j-1) + Cj$; $\beta max\ \{\beta, Cj\}$;

else $t(i) = t(i-1) + Cj + \beta$;

$\beta = Cj$; $\delta = Cj$;

if ($tj + \beta \geq dj$) then

return (Not guaranteed FT)

else

return(Guaranteed FT);

}

The above algorithm will return that guaranteed FT, if only one fault is detected in interval Δf .Then 'n' tasks which are present in queue will be executed properly that is fault freely within given deadline[2].

2.5 BCE Algorithm

In periodic tasks the probability of occurrence of fault is considered an uniformly distributed function. Imprecise computations, which is proposed by Lin et all which concentrates on iterative calculations[1].

To provide deadline in real-time systems Campbell et al. Proposed an idea which tells that two versions are provided for each task, in which the one is primary and the other is backup versions.

This BCE algorithm is an offline scheduling algorithm which deals with last chance strategy. In this approach, primary will start executing first until the completion of given time. If in assigned time, primary does not complete the task, alternative will be given a chance to execute, otherwise if primary executes task completely and properly within scheduled time, alternative will not be given chance. The alternatives are given chance to pre-empt the currently executing primary when the time interval assigned for them appears.

2.5.1 Checking Availability Time (CAT)

CAT refers to Checking Availability Time. The processor time will be wasted if the alternatives reassume the execution due to failure of primary. The maximum Availability time is given by

$$AT(kl)=(Vkl - CT) - \sum_{k=1}^{m} Ik \qquad (7)$$

Where *CT* refers to Current time and Vkl denotes the Notification time and *Ik* denotes the time interval that the system as assigned for other tasks[1]. If the calculated maximum availability time is less than that of primary version, the alternate version will be executed by the system; otherwise it will execute the primary version.

2.5.2 Eliminating Idle Time (EIT)

EIT refers to Eliminating Idle Time. The possibility of system or processor being in idle state will be increased if we use the CAT algorithm[1]. This can be overcome by the EIT algorithm which employs a method which allows alternate version to execute the task when processor is reaching its idle state.

2.6 RM-FT Algorithm

Theorem A: For a given set of tasks Tall, the lowest priority task tij completes is given by Tij in RMFT, if and only if δ(T)=0 for some t[4,13] .

Corollary A: The necessary condition required for all Tall and for all 'f' faults can be obtained from Theorem 1.

Let 'j' be the number of tasks and where Tj contains 1 to j high priority tasks in Tall.

Example:

	Period(Ti)	Execution time(Ci)
T1	4	1
T2	8	1

Here task set contains Tall=[T11,T12,T13]

Algorithm:

N=|Tall|

R={T11}

FOR j=1 to N

Array A[1…..j]//of size 'j'

A=RM-IND(R) //create a schedule of set R

Tlk=lowest priority task is R

For t=0 to PC

If (t=fuin(Tij) and fuin(Tij)>=T1*(k-1)) then

If δ(R)=0 and fuin(Tij)<=t<=Tij then

Continue

Else

Not Schedulable And Stop

End If

End If

End For

If(j<N) then

Q=Q U{Tij}//where Tij having next highest priority in

taskset (Tall-R)
End if
End For
Task set Tall is Schedulable
Stop
Example:

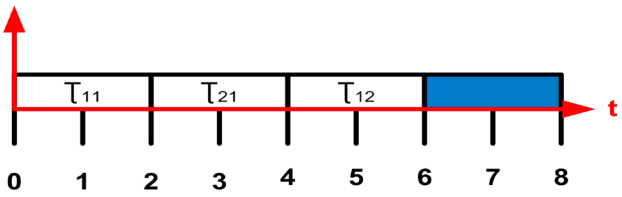

Figure 2. Example for RM_IND algorithm.

Consider the maximum fault = 2. Figure 2 and 3 describes the RM_IND schedule j= 1,2, and 3.

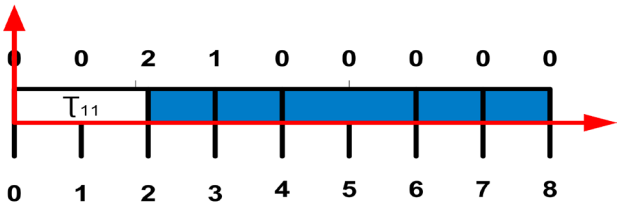

Figure 3. RM_IND Schedule.

When j = 1, R = {T11} and δ(T) = 2. Figure 4 represent the RM_FT Scheduling[3].

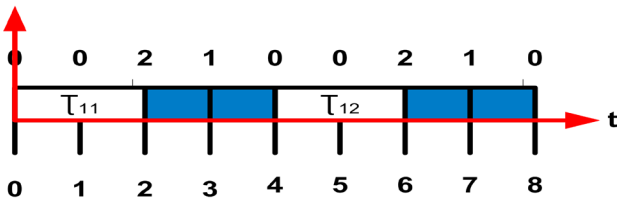

Figure 4. RM_FT Schedule.

Task instance T11 in RM_FT schedulable since δ = 0 at time t = 4.

For j = 2, R = [T11,T12] in RM_FT and δ11 = 2 and δ12 = 2. The Schedule is given in Figure 5.

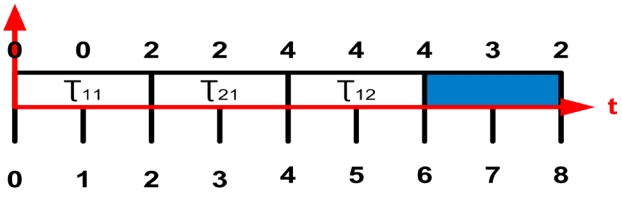

Figure 5. Non Schedulable task instance.

2.7 Modified Worst Fit Decreasing (MWFD)

This scheme can be used to analyse the MWFD scheme we carried out an experiment on, randomly generated 10,000 tasks. The periods of the tasks are so taken that all tasks have the same probability of high, medium and low. The standard deviation of utilization is limited by maximum value[11]. The maximum value is a function of mean of utilization .let us assume the maximum number of faults be 'K' by taking into consideration that at maximum one fault can be tolerated by each task at a particular instance.

We validate the MWFD scheme and found that it does not degrade the feasibility of the schedule. The feasibility can be calculated as the percentage LPZ = multiprocessor utilization bound for scheduling a task set based on generic allocation schemes and conservative RMA.

From the analysis we have found the MWFD exhibits better schedulability performance than FFD with ratio of standard deviation to that of the maximum of standard deviation, i.e., change in task utilization is small. In other case, when utilization is very different among tasks of FFD.

MWFD and FFD provide greater comparable schedulability performance. Superior schedulability and performance to LPZ in all cases is obtained by MWFD. MWFD and WFD achieve similar schedulability performance with increase in overall faults 'K'. The analysis shows that in the absence of faults MWFD leads to energy saves up to 60% as compared to FFD and WFD, also with increase in number of faults the average energy consumption per feasible task set increases slightly.

2.8 Primary/Backup (PB) Algorithm

Study of Primary/Backup (PB algorithm) FT scheduling, allows us to identify the solution for processor transient/ permanent faults. Backup overloading and de-allocation methods can be recommended for fault tolerance which demands less processor time. Assume a system consists of homogenous multiprocessors with central task scheduler handling non precedence, independent tasks[6]. By this approach we can handle both permanent and transient faults.

Task Ti modeled as shown in Figure 6

$T_i = <a_i, r_i, d_i, c_i>$

Where,

a_i = arrival time, r_i = ready time,

d_i = dead line, c_i = processing time

Consider task Window should be at least twice large as processing time.

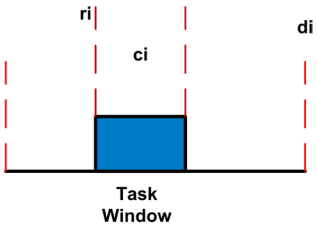

Figure 6. Task Parameters.

The scheduler should guarantee to execute even though any one processor in failure state by assuring the second failure will not happen before recovering from first failure. If scheduler confirmation is not assured, then the task should be rejected. Figure 7 represents the major techniques of scheduling.

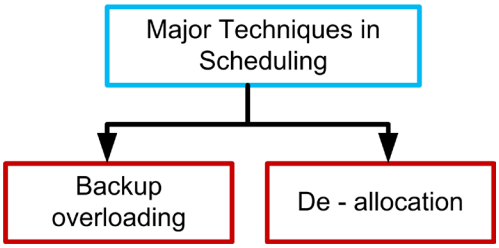

Figure 7. Scheduling Techniques.

Two different time slots are available. Primary/backup time slots are required when primary and backup tasks scheduled. Overloaded time slots are required if we try to schedule backup copies of more than one tasks in same slot.

Scheduling Restrictions:

Let Primary copy P_{ri} and Secondary copy of task B_{ki}

- Primary task and secondary can't be scheduled on same processor.
- Begin time of secondary task has to be greater than primary So that backup can be executed after fault detection.
- Both primary and backup tasks have to be scheduled between r_i and d_i (ready time and deadline)
- If two primary tasks are allotted in one processor then their backup must be scheduled in some other processor without overlap.

For the set of task Ti, the fault tolerant scheduling algorithm is

- Schedule P_{ri} as early as possible.
- Try to schedule B_{ki} by overloading technique on existing backup slot. If not, schedule it by as late as possible on free slot.
- If schedule of both P_{ri} and B_{ki} have been found, then accept those task otherwise reject it.

Algorithm principle

- When new task arrives, schedule should be done for both primary and backup.
- Maintain the details of existing slots while scheduling both the copies.
- Apply AEAP principle for primary and ALAP for backup.
- After successful completion of primary, its backup is de-allocated.
- The de-allocated slots can be used for scheduling new tasks.

Thus utilization is increased and overhead is reduced.

After arrival of 2 new primary tasks:

1. More than one fault can be tolerated by this algorithm.
2. Need of scheduling more backup copies for tolerating two faults arrives at same time.

Transient fault tolerance using check pointing based technique.

Checkpoint refers to the state of the system at a particular instance of time. The process of check pointing involves periodically storing the checkpoint (in local or remote memory) during the execution of a task. In the event of a transient fault, execution is continued from the last valid checkpoint. One important parameter of check pointing is checkpoint overhead which is defined as the increase in the execution time. This overhead is dependent on

- Number of checkpoints N.
- Time for checkpoint capture and storage.
- Time for recovery from a checkpoint, Tr.
- Fault arrival rate λ.

Following are the assumptions regarding check pointing based transient fault-tolerance.

- Transient faults follow Poisson distribution with a rate of λ failures per unit time.
- Transient faults are point failures i.e. these faults induce an error in the system and disappear.
- The probability of multiple transient faults in each checkpoint segment is negligible.
- Checkpoints can be inserted anywhere in the execution time. This assumption although difficult to accomplish in practice, gives a first order approximation on the problem at hand.

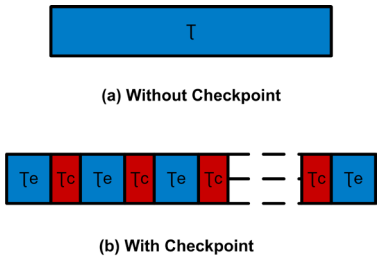

Figure 8. Task Execution with and without Checkpoint.

Figure 8 shows an example task execution with N checkpoints. Let T denote the time taken by the task for execution and Tc, the time taken by the task for execution in each checkpoint segment. Clearly, Tc = T/N+1. The probability of at least one fault in inter checkpoint interval ($T_c + T_o$) is

$$P_e = 1\text{-}e^{-\lambda (Tc+To)} \quad (8)$$

Assuming fault arrival follows Poisson process, the probability of more than one fault in the inter checkpoint interval is negligible as $\lambda T_c << 1$. Using first order approximation, the expected length of checkpoint segment E [Tc] is calculated as

E [T_c] = P {no error in segment} ∗ normal checkpoint interval + P {error in segment} ∗ modified checkpoint interval.

2.9 Earliest Deadline First Scheduling

It is a dynamic scheduling algorithm mainly used in real time system where tasks are placed in queue according to their priority. Whenever any event for scheduling happens (pausing of task, finishing of task, release of new task, etc.) the queue will starts its searching for the process having nearest deadline[9]. It is one of the optimal scheduling algorithms used in uniprocessor.

The scalability test for EDF is

$$Ub = \sum_{i=0}^{n} \frac{Wi}{Ii} < 1 \quad (9)$$

Where W_i=worst case computation time for n number of processes; I_i is the inter arrival time of each process; generally for kernels implementation this scheduling algorithm is used.

2.10 A Probabilistic Approach

2.10.1 A Computational Model

Consider a task $\tau_i = (C_i, T_i)$, where C_i is a completion time and T_i is a period and we have to schedule that tasks. In this case pre-emption is permitted[5]. Let $T = \{T1, T2. . . Tn\}$ we will denote as periodic task of system. *Utilization* $Ui = Ci / Ti$.

$$U_{sum}(T) = \sum_{Ti \in T} \frac{t_i C_i}{T_i} \quad (10)$$

Now, the maximum utilization $U_{max}(\tau)$ of periodic task can be defined as follows

$$Umax\ (\tau) = max_{\tau i \in \tau}\{U_i\} \quad (11)$$

2.10.2 Schedulability

The condition for scheduling instance I by EDF among m processor is

$$U_{sum}(\tau) \leq m\ s - (m-1)U_{max}(\tau) \quad (12)$$

If transient fault will occur and it can be represented by exponential law.

We have $p_i = 1 - e - C_i$ which depends on the task T_i.

$$P = \prod_{i=1}^{n} \left(1 - p_i^{t_i}\right)^{n_i} \quad (13)$$

Where P is the probability of meeting the deadline during the interval [0, P] by assuring to meet the deadline by at least one copy of each task.

Two different problems have been taken into account.

Minimizing the Number of Processors

Consider a periodic system τ having Є as maximum tolerated probability of failure along with F a time frame. It will give outputs as t_i the copies of each tasks τ_i and m minimum number of processor and it should meet below mentioned constraints (1) minimum m, (2) probability of failure during *[0, F]* must be lower than Є.

Optimizing Reliability

*U*sed for minimizing the failure probability of task in given time interval [0, *F]*. This algorithm will take task and size of multiprocessor as input along with *F* which is nothing but time frame. The following constraints should be met: (1) minimum Є. (2) The required number of processor must not be greater than *m.* At the beginning of the Algorithm only one copy of each task is available. Then as we will schedule the system, we will make copy for each task. Schedulability will be checked and after that failure probability of system.

Duplicating tasks: We have analysed 5 different solutions to increase number of copies for task

Increase all: By increasing copies for all tasks

$\forall i\ ti \leftarrow ti + 1.$

Min utilization

$$i = argmin\left(\frac{t_i C_i}{T_i}\right) \quad (14)$$

The number of processors increases with U_{sum}. The drawback is that the probability of failure of tasks will not be taken into account.

Min failure: The task will be duplicated which has the highest probability to fail and increase t_i such that

$$i = argmax\left(p_i^{t_i}\right) \quad (15)$$

Min failure-request: The task will be duplicated, which is having the highest probability of failure. To increase it we will follow below equation

$$i = argmax\left(\frac{F}{T_i} p_i^{t_i}\right) \quad (16)$$

Min failure-utilization: The task will be duplicated which has higher failure probability and low utilization: The t_i will be increased in such a way that

$$i = argmin\left(\frac{C_i}{T_i p_i^{t_i}}\right) \quad (17)$$

2.11 Recovering from Transient Fault

2.11.1 FT Algorithm

- Determines low list and low list backup as well as high list and high list backup from task list
- High priority is given to the primary task
- If backup copy is not assigned then corresponding task moved to the low list
- Schedule low list which is driven by fixed priority scheduling scheme. A backup copy is assigned to processor which is different from processor with primary task.
- Schedule high list backup with low list backup on different processor from those of primary task.

After the completion of primary task a backup copy in T_H is allocated to the processor due to this we can utilize slack interval time without violating the execution time period deadline[7,8]. When the FT algorithm is applied to the processor task execution scheduled based on fixed priority driven scheduling algorithm. When fault occur on processor P at particular time t_f, that task is adopted If transient fault occurs in task t_i of T_H a backup copy will be executed in same processor by using time redundancy we can minimize transient fault. When failure of task occurs we can easily recovered uncompleted task at T_H by assigning backup copies of that particular task to the different processor.

2.12 EH - EDF (Energy Harvesting -Earliest Deadline First)

Some real time systems are powered by renewable energy sources and rechargeable units like photo voltaic cells. In this case there is chance of transient fault due to the variations in energy source. So we have to consider the characteristics of energy source energy consumed by each task. In this scheduling decisions are taken at runtime without the prior knowledge of energy source by healthy processor[10].

Table 1. Analysis Table

Algorithm	Advantage	Dis-advantage
Feasibility shortest path:	It minimizes the length of the queue. Complexity is high but it is more reliable.	This system is more complex
Linear time heuristic:	Used in static system and Low complexity algorithm. Uses less number of nodes of graph.	We have to know the time of the task in the queue.
BCE:	Backup will only run when primary fails	More execution time
Checking Availability Time:	We can calculate availability time of processor.	Required alternate copy of task
Eliminating Idle Time:	When the processor is found to be idle, choose an alternate for execution.	Requires more execution time
RM-FT:	The algorithm consists separate blocks for a task classification, task allocation and task scheduling and fault management.	Requires more memory since it creates backup of various tasks.
EDF Scheduling:	This algorithm ensures that all the task complete within the deadline.	It can't be used in real time system in industry.
PB Scheduling:	Less processor time to provide fault tolerance.	Hardware requirement is more.
Probabilistic approach:	Hardware requirement is less.	Required more execution time.
EH-EDF:	Fault tolerance with Energy minimization.	Decision made without the prior knowledge of energy source.

3. Conclusion

We have considered fault-tolerant systems since system may fail to meet the deadline due to transient fault or any kind of error. We have done the analysis in order that tasks meet their deadline**.** The discussion showed that Min failure-request is the best one. It has been proved as optimal by 3 best probability based heuristics. It has been concluded that in RM-FT algorithm if j=1 the task can be schedulable, but if j=2 then the task is not schedulable. So the higher value of j makes the possibility of less schedulable in RM_FT. EDM mechanism can have the capability to detect the error before the execution of primary copy and hence we are provided with more slack. To provide better fault tolerance, deadline mechanism is used for periodic tasks which deal with modified BCE algorithm. A gradient-based technique (check pointing) has been proposed to improve the lifetime of a homogeneous reconfigurable multiprocessor system while optimizing for transient fault tolerance. Experiments conducted with variable fault-tolerance requirement demonstrate that the proposed solution improves lifetime by 10 percent to 60 percent as compared to the state-of-art transient fault tolerant technique. The gradient-based technique provides up to 500× reduction in design space exploration time with less than 5% distance from optimality.

4. References

1. Asadi M, Menhaj MB, Yavari E. A modified BCE algorithm for fault-tolerance scheduling of periodic tasks in hard real-time systems. Third Asia International Conference on Modelling & Simulation; 2009.
2. Ghosh S, Melhem R, Mosse D. Fault-tolerant scheduling on a hard real-time multiprocessor system. Proceedings of 8th Inernational Parallel Processing Symposum; 1994 Apr. p. 775–82.
3. Pathan RM. Fault-tolerant real-time scheduling algorithm for tolerating multiple transient faults. 4th International Conference on Electrical and Computer Engineering (ICECE 2006); 2006 Dec; Dhaka, Bangladesh.
4. Pan X, Yao X, Ping L. Research on real-time scheduling strategy for transient fault tolerance in NC system. Sixth International Conference on Intelligent Systems Design and Applications (ISDA '06); 2006 Oct. p. 684–9.
5. Berten V, Goossens J, Jeannot E. A probabilistic approach for fault tolerant multiprocessor real-time scheduling. 20th International Parallel and Distributed Processing Symposium (IPDPS 2006). 2006 Apr.
6. Mosse D, Melhem R, Ghosh S. Analysis of a fault-tolerant multiprocessor scheduling algorithm. 24th International Symposium on Fault-Tolerant Computing (FTCS-24). 1994. p. 16–25.
7. Pandya M, Malek, Miroslaw. Minimum achievable utilization for fault-tolerant processing of periodic tasks. IEEE Transactions on Computers. 1998 Oct; 47(10):1102–12.
8. Goo HW. 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.
9. Chen Y, Yu X, Xiong G. Fault-tolerant earliest deadline first scheduling with resource reclaim. Proceedings of Fifth International Conference on Algorithms and Architectures for Parallel Processing; 2002 Oct. 278–85.
10. Shanmugasundaram M, Kumar R, Kittur HM. A survey of uniprocessor transient fault. International Journal of Applied Engineering Research. 2014; 9(21):8329–36.
11. Wei T, Mishra P, Wu K, Liang H. Fixed-priority allocation and scheduling scheme for energy-efficient fault-tolerance in real time multiprocessor systems. IEEE Transactions on Parallel and Distributed Systems. 2008 Nov; 19(11):1511–26.
12. Kim H, Lee S, Jeong B-S. An improved feasible shortest path real-timefault-tolerant scheduling algorithm. Proceedings of Seventh International Conference on Real-Time Computing Systems and Applications. 2000 Dec. p. 363–7.
13. Beitollahi H, Deconinck G. Fault-tolerant rate-monotonic scheduling algorithm in uniprocessor embedded systems. 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06); 2006.
14. Mosse D. Enhancing real-time schedules to tolerate transient faults. Proceedings of 16th IEEE Real-Time Systems Symposium REAL-95; 1995.