Computers, Materials & Continua
DOI: 10.32604/cmc.2022.025059
Article

Tech Science Press

Fault Tolerance in the Joint EDF-RMS Algorithm: A Comparative Simulation Study

Rashmi Sharma[1], Nitin Nitin[2] and Deepak Dahiya[3,*]

[1]University of Petroleum & Energy Studies (UPES), School of Computer Science, Energy Acres Building, Bidholi, Dehradun, 248007, Uttarakhand, India
[2]Department of Electrical Engineering and Computer Science, College of Engineering and Applied Science, University of Cincinnati, 2600 Clifton Ave, Cincinnati, OH 45221, United States
[3]College of Computer and Information Sciences (CCIS), Majmaah University, Majmaah, 11952, Kingdom of Saudi Arabia
*Corresponding Author: Deepak Dahiya. Email: d.dahiya@mu.edu.sa
Received: 10 November 2021; Accepted: 21 February 2022

Abstract: Failure is a systemic error that affects overall system performance and may eventually crash across the entire configuration. In Real-Time Systems (RTS), deadline is the key to successful completion of the program. If tasks effectively meet the deadline, it means the system is working in pristine order. However, missing the deadline means a systemic fault due to which the system can crash (hard RTS) or degrade inclusive performance (soft RTS). To fine-tune the RTS, tolerance is the critical issue and must be handled with extreme care. This article explains the context of fault tolerance with improvised Joint EDF-RMS algorithm in RTS. The backup method has been derived to prevent the system from being recursively migrating the same task. If any task migrates three times, this migrated task will get shifted to the backup queue. This backup queue assigns the task to a backup processor and is destined for final execution. For performance evaluation purposes, a relative graph between fault and failure rates, failure and total processor utilization along with other averages have been evaluated. Furthermore, these archived results are compared with fault-tolerant Earliest Deadline First (EDF) and Rate Monotonic Scheduling (RMS) algorithms independently in relatively similar conditions. These comparisons show better performance against overloading conditions.

Keywords: Fault tolerance; joint edf-rms algorithm; real-time systems (RTS); distributed systems; migration

1 Introduction

RTS is a system where task execution is not dependent only on correct logic, but on-time execution (deadline) is equally important [1–3]. Based on deadline RTS is of two types: Hard and Soft [1–3]. In soft RTS, missing deadlines degrade the performance of the system, but hard RTS give catastrophic result along. Video game is a good example of soft RTS where the target timing of the game is

scheduled. If a player is unable to achieve the mentioned time then only the player performance affect. Whereas, in hard RTS, if an aircraft is unable to execute some operations on time, then catastrophic results like loss of life can happen. The above-mentioned catastrophe or performance deprivation may occur due to the occurrence of a fault in the system and the mechanism used to handle such problems called fault tolerance [4].

In RTS, priority-based scheduling algorithms employee to schedule the tasks, with criteria to decide the priority of tasks. If a task is unable to match the criteria, then that task is non-schedulable. In RTS, meeting the deadline means successful task execution and missing a deadline means failure of a task. The occurrence of a fault in scheduling algorithms is one of the reasons for missing a deadline. Many researchers have discussed the fault tolerance techniques in EDF and RMS [5–7]. Here, fault tolerance technique is embed in joint EDF-RMS real-time scheduling algorithm [8] to reduce the permanent fault along with intermittent. The organization of remaining paper is as follows: Next section will explain the fault tolerance mechanisms in Real-Time and Distributed Systems (both), along with the scheduling algorithms of RTS. Further Section 3 explains the fault tolerance in the Joint EDF-RMS algorithm. Simulation studies and results explain in Section 4 and 5. Conclusion and future work will depict at the end.

2 Background and Preliminaries

In today's era, features of RTS have been merged with many technologies i.e., cloud computing, distributed systems, mixed-criticality systems, big data and many more. In all stated systems, fault tolerance is the main concern of all algorithms. In [9], hybrid of traditional fault tolerance methods discuss for mobile distributed systems. Dependently fast library uses in [10] to implement the interface to prevent unexpected fail-slow tolerant distributed systems. A scheduling algorithm for cloud computing is proposed in [11] to minimize the response time of tasks present in backup after multiple failures of tasks. The stopwatch automata network model use to make real-time computer system (RTCS) fault tolerant [12]. Additionally, to provide reliability against transient faults high power-based backup processor use to execute selective backup tasks [12]. In [13] author has proposed two scheduling algorithms FEED-O and FEED-OD to handle the overloading of backup tasks in overlapping time intervals. In [14], Fault-tolerant multi-core mixed criticality systems use to cope with high power and temperature for dependent dual-criticality tasks. In [15], with server CAN (one of the types of storage), a fault tolerance mechanism has been used. In [16] also the fault-tolerance concept implements in big data. Based on aforementioned work, it is clear that fault tolerance is not important in RTS only, but in every domain [9–17].

The focus of this paper is on fault tolerance scheduling algorithms in Real-Time Distributed systems (RTDS). RTDS is a distributed system with temporal and dynamic behaviour, as it changes with time. Temporal activity/behaviour includes arrival time, waiting time, execution and completion time of a task. The completion time of any task affects the performance of dependent and enqueuer tasks. Therefore, timely execution of tasks is the uppermost priority of any RTS scheduling algorithms. Presence of fault in a system may affect its performance by continuous missing of tasks deadline. The system failure can happen due to many improperly handled faults, like preemption of lower priority tasks or lack of resources. This missing deadline is bearable in soft RTS, but not in hard [18,19]. The further section will explain the detailed information about failure, faults and their relationship with each other.

2.1 About Fault Tolerance

When we talk about fault tolerance, the scenario comes to our mind that there is a situation where an irregular, unusual, erroneous function is being played. It degrades the performance or sometimes fails the entire system. Now tolerance channels our system in such a way that these faults are bypassed, thus, maintaining the near normal functioning of the system. Sometimes multiple modules in a redundant fashion, mirror images of the existing system and some kinds of remedial measures are proposed inline. Tolerance methods are necessary for the successful running of any system because critical faults strike without warning. Following (Tab. 1) are few examples where tolerance methods for faults present:

Table 1: Examples of fault tolerance

System	Fault	Tolerance method (Back-up)
Storage server	Storage device (Hard disk) fails	Mirroring technique
Networking	Network card fails	The second card is there
Computer system	Power failure	UPS
Distributed systems	Task/process overloading	Task duplication/migration
RTS	Scheduled task missing deadline	Modified scheduling algorithms that accommodate the given fault
Some programming languages	Users enter vague data	Handled by Exception handling

2.2 Fault Tolerance in RTDS

Real-Time tasks schedule on distributed systems where faults of distributed systems affect respective real-time tasks or vice versa. Following Tab. 2 summaries, some common types of faults in both systems [20,21] and Fig. 1 displaying the visualizations of respective fault types:

Table 2: Types of faults

Fault types	Definition	Example w.r.t DS	Example w.r.t RTS
Transient Faults	Occurs once in a system and does not happen again after debugging	Network connection, once establish successfully	Successful deployment of software application.
Intermittent Faults	Difficult to debug as it appears again in the system	Loose connections in hardware	Missing deadline in hard/soft system
Permanent Faults	Does not go away once occur	The system runs out of memory	Domino's effect of EDF (or tasks overloading)

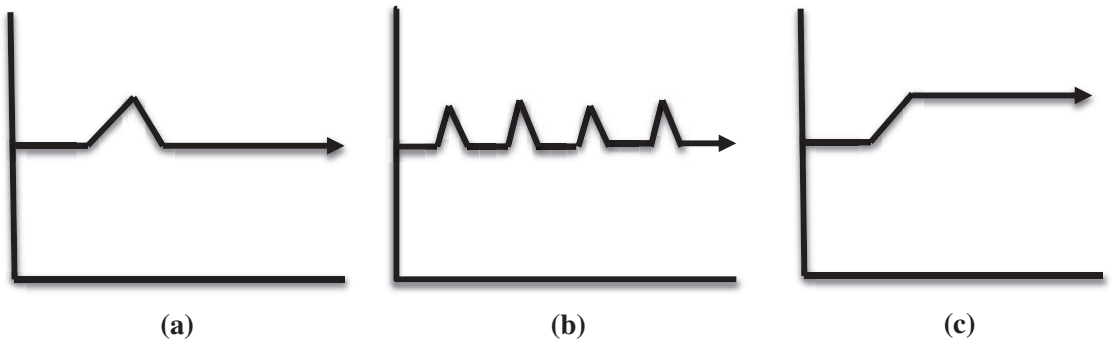


Figure 1: Visualization of (a) Transient fault (b) Intermittent fault and (c) Permanent fault

2.3 Fault Tolerance Techniques

The fault is an unintended behaviour of a system that either reduces the performance or fails the system. To protect the system, fault tolerance strategies are required. However, fault-masking and system reconfiguration are a few techniques for fault tolerance but commonly, redundancy in hardware, software, information or time is used [22–24]. In following Tab. 3 detailed discussions on redundancy techniques have explained:

Table 3: Types of redundancy techniques

Types of redundancy techniques	Description	Approach
Hardware Redundancy	Fault can be resolved by providing multiple physical copies of hardware components.	Passive, Active and Hybrid Techniques
Software Redundancy	-Used to detect hardware faults -Used to detect software faults	(for H/W faults) Consistency checks, capability checks, ALU tests, Testing for communication. (for S/W faults) N-Version Programming (NVP) and Recovery Blocks (RB)
Information Redundancy	Guarantee data consistency by exploiting additional information to achieve a redundant encoding	Error detection and correction codes
Time Redundancy	Computations perform continuously at different points (checkpoints) in time and then compared. Here, no need for extra hardware.	Used approaches, e.g., ALU: recomputing with shifted operands, with swapped operands etc.

Further section explains fault tolerance scheduling algorithms.

2.4 Fault Tolerance Scheduling Algorithms in RTS

While designing any scheduling algorithms for RTS, following criteria should be considered:

- Each task should execute once in a system i.e., parallelism between the same task should not be there
- All tasks should meet the deadline
- Number of processors failure should be tolerated i.e., failure of one processor should not affect the progress of the entire system (hence, always schedule a backup of tasks on separate processors)
- Preemption of tasks helps to improve the efficiency of the system.

Following are scheduling algorithms employed for the fault tolerance and avoidance as well [2,25–28]:

- First come first serve (FCFS)
- Shortest Job First (SJF)
- Preemptive
- Non- preemptive
- Round Robin Techniques

For all scheduling algorithms following terminologies are common:

Input: Arrival of Periodic tasks t_i with arrival time a_{t_i}, deadline d_{t_i}, period p_{t_i}, computation time c_{t_i}, task utilization u_{t_i}, CPU utilization U_{P_i}, a priority of respective task pr_{t_i} and time quantum TQ.

Output: Task meeting $meet_{t_i}$ and missing $miss_{t_i}$ of respective tasks

Table 4: Summary of real-time scheduling algorithms for fault tolerance techniques

Scheduling algorithms	Scheduling criteria	Gap	Type of fault occurs	Fault tolerance techniques
First Come First Serve (FCFS)	Task arrives first will be at the highest priority.	Convoy Effect (Starvation)	Intermittent Faults	Time and Software Redundancy
Shortest Job First (SJF)	Task having shorter computation/burst time at highest priority ($pr_{t_i} \propto \frac{1}{c_{t_i}}$).	• Starvation due to preemption of lower priority task by higher priority tasks multiple time • Difficult to determine CPU computing time in advance	Intermittent Faults	Time and Software Redundancy

(Continued)

Table 4: Continued

Scheduling algorithms		Scheduling criteria	Gap	Type of fault occurs	Fault tolerance techniques
Preemptive	Earliest Deadline First (EDF) (Dynamic)	Nearer the deadline higher the priority ($pr_{t_i} \propto \frac{1}{d_{t_i}}$)	• Domino's Effect	Intermittent and permanent faults (sometimes)	Time, Software and hardware redundancy
	Rate Monotonic Scheduling (RMS) (Static)	The task with a smaller period have higher priority ($pr_{t_i} \propto \frac{1}{p_{t_i}}$)	• Due to fixed / static priority and upper bound conditions, it offers less schedulability than EDF	Intermittent and permanent faults (sometimes)	
Non-Preemptive	Non-Preemptive Shortest Job First	Scheduling criteria is as per the previously mentioned SJF algorithm.	• Starvation due to task preemption by higher priority tasks. A process engages the CPU until termination or reaches a waiting state.	Intermittent Faults	Time and Software Redundancy
Round Robin Techniques	Time quanta/slice-based algorithm	Once a task executes for the assigned time quanta, task get preempted and another task starts execution for the given quantum of time.	• It spends more time in context switching • It does not give special priority to important tasks • Difficult to find correct time quantum	Intermittent Faults	Software Redundancy

The above-mentioned Tab. 4 is the summary of a few scheduling algorithms to tolerate the occurrence of faults. Most algorithms use for intermittent faults because missing deadlines are difficult to predict Hence, to endure such faults, many time-based scheduling algorithms exist that come under

software and time redundancy technology. Out of the above-mentioned categories, this paper deals with preemptive scheduling algorithm class where EDF and RMS both algorithms are used for fault tolerance purposes. The next section will describe the working of the joint EDF-RMS algorithm in a faulty environment.

3 Fault Tolerance in Joint EDF-RMS Algorithm

In RTS, failure of single task may affect the system performance and destroy it as well (catastrophe result). Hence, fault management is required in RTS. Many researchers actively participate and implement/simulate the fault-tolerant system with the help of some scheduling algorithms. Therefore, the performance of Joint EDF-RMS algorithm improvises by embedding the feature of fault tolerance in it. Following Fig. 2 is the pictorial representation of mentioned algorithm with fault tolerance management.

There are four blocks A, B, C and D. Block 'A' represents a global queue where tasks will assign to randomly selected processors. According to RMS, the task having highest priority will assign first to a chosen processor of block 'B' and similar trend will follow for remaining tasks. To avoid overloading in a queue, RMS criteria has been used i.e., $taskutilization <= n\left(2^{\frac{1}{n}} - 1\right)$, where n represents the total number of tasks. Further, a selected processor of the 'B' block use EDF algorithm to schedule the signed tasks (0.81 is the upper bound of CPU utilization [8,29]). To handle overloaded (victim) tasks, blocks 'C' and 'D' are available. Victim task is a task that migrates minimum of three times and remains unexecuted (may become responsible for missing the deadline of dependent tasks). For such unexecuted tasks, a backup queue is there that schedule victim tasks to the selected processor. Here, RMS algorithm use for the scheduling of victim tasks on assigned processors.

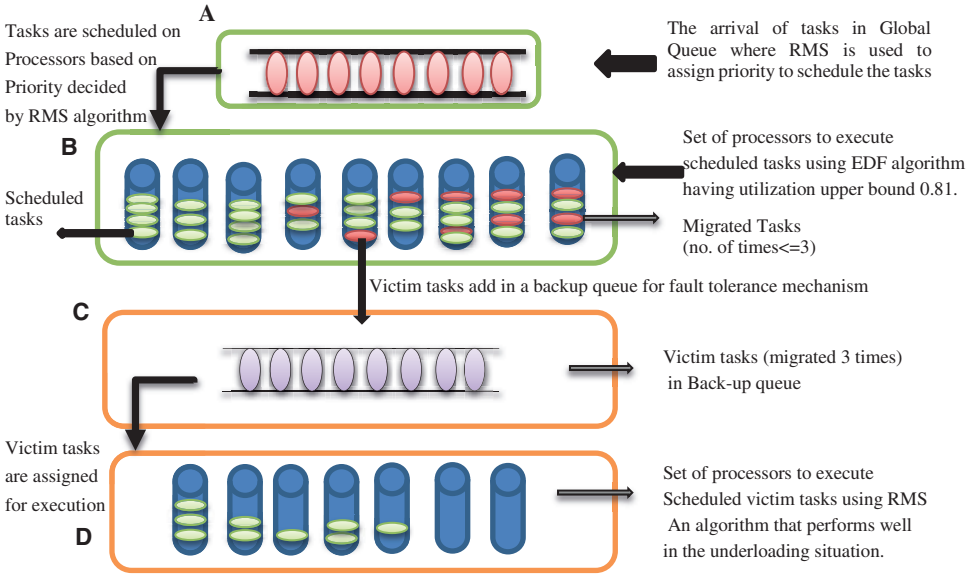


Figure 2: Working of joint EDF-RMS algorithm with fault tolerance technique

Following is the algorithm for the proposed work:

Fault-Tolerant Joint EDF-RMS Algorithm

Algorithm_3.3: Joint EDF-RMS Algorithm with Fault-Tolerant Technique

Input: Arrival of tasks with $\tau_{arrival}, \tau_{wcet}, \tau_{period}, \tau_{dline}$
Output: Number of tasks successfully meet and unable to achieve the deadline with other parameters
BEGIN
GlobalScheduler( ) // Global task Queue

1. Arrival of tasks t_i // Periodic arrival of tasks with arrival time, wcet, assigned deadline and period
2. $tasku = wcet/Period$;
3. $UB = n*$(Math.*pow*(2, 1.0/*n*)-1);
4. **IF** $tasku <= UB$
5. Generated task is schedulable
6. *pselection(task)*
7. **Else**
8. The task is non-schedulable

pselection(task) // Random Selection of Processors

1. Random Selection of Processor
2. *PQueue(task);*

PQueue(task) // Processors local queue

1. Assign priorities to tasks on the basis of deadline
2. mig = 0 //migration counter initially 0
3. $task_{priority} \propto \frac{1}{task_{deadline}}$
4. *TaskExecution(task, mig)*

TaskExecution(task,mig) // Task Execution by using EDF Scheduler

1. **If** tasku <= 1
2. $U = U+$ tasku //Cumulative accumulation of task utilization
3. **If**(U <=.810) //Processor utilization upper bound
4. The task is ready for execution
5. **Else**
6. *Task Migration (task, tasku, mig) // now mig = 1*

Taskmigration(task, tasku , mig) // Task Migration on the basis of a processor utilization factor

1. Sorting of all processor utilization
2. Give task to less utilized processor
3. *PQueue(task, mig);* //mig take care number of times given task migrate
4. **If** mig == 3
5. Backup-Global scheduler(task,tasku,mig)
6. **Else**
7. Taskmigration(task, tasku , mig)// *now mig = 2*

(Continued)

Algorithm_3.3: Continued

Backup-Global scheduler(task,tasku,mig)|| Global scheduler for tasks

1. Random Selection of Back-upProcessor (3 processors)
2. *BkupPQueue(task);*

BkupPQueue(task) // Back-up Processors local queue

1. RMS algorithm is used here.
2. $task_{priority} \propto \frac{1}{task_{period}}$
3. *BkupTaskExecution(task)*

BkupTaskExecution(task) // RMS is in action

1. **If** tasku $<= n(2^{\frac{1}{n}} + 1)$
2. $U = U+$ tasku // cumulative totaling of task utilization
3. **If**($U <= n(2^{\frac{1}{n}} + 1)$)&& (U <= 1) // Acceptance test for victim task execution
4. The victim task is acceptable for execution
5. **Else**
6. *Task Migration (task, tasku)*

END

4 Simulation

- **Experimental Set-up**

For simulation purposes, the Eclipse Oxygen.3a version use for the java programming language. The concept of thread uses here to create, wait and execution of tasks. Maximum 2100 and minimum 300 tasks are used in distributed systems of six (3 for backup) processors. The implementation of all three algorithms EDF, RMS and joint EDF-RMS done on same tasks in synchrony to provide the same simulation parametrization. Following parameters use to evaluate the performance of the above-mentioned algorithms:

1. **Fault rate:** In RTS, if the task has less chance to meet the deadline as per utilization calculations and *if* $(task.utilization < cpu.utilization)||(\sum_{i=1}^{t_i} task.utilization > CPU.utilization)$ then the scheduling algorithm allow it for execution, and that task can be a reason for the failure of many remaining tasks. Based on schedulability test, victim tasks identify here. Following equation is used to calculate the fault rate:

$$fault\ rate = \frac{victim\ task}{total\ number\ of\ tasks\ arrive\ in\ the\ system} \tag{1}$$

2.**Failure rate**: Tasks fail to meet the deadline due to fault done by scheduling algorithms by allowing tasks to execute. Following equation calculate failure rate:

$$failure\ rate = \frac{number\ of\ tasks\ miss\ thedeadline}{total\ number\ of\ tasks\ arrive\ in\ the\ system} \tag{2}$$

3. **Processor utilization threshold**: The reason behind task migration is utilization of CPU. If any task crosses the threshold limit of CPU utilization then, that task will migrate. Hence, migration point in RMS is $n(2^{1/n}-1)$, in EDF 1 *or* 100% and in joint EDF-RMS it is 0.81 *or* 81%.

5 Results and Discussion

Before the discussion on the results, the following lemma explains the dependency of task execution on the aforementioned parameters:

- *Lemma 5.1* Failure of task is dependent on victim tasks, migration and CPU utilization.

Proof

Considering the arrival of N tasks in the system S where $\tau_1, \tau_2, \tau_3, \ldots\ldots\ldots, \tau_k \in N$ and their CPU utilizations are: $\frac{w_1}{\rho_1}, \frac{w_2}{\rho_2}, \frac{w_3}{\rho_4}, \ldots\ldots\ldots\ldots, \frac{w_k}{\rho_k}$ respectively. Every time a new task arrives and the system computes its utilization whether it is less than or greater than CPU utilization i.e., 1 (100%) or not.

Table 5: Demonstration without migration

$\frac{w_1}{\rho_1} < 1, \left(\frac{w_2}{\rho_2} < 1\right)$ and $\frac{w_1}{\rho_1} + \frac{w_2}{\rho_2} < 1$ (3)	Where w *and* ρ are the worst-case execution time and period of task. When a task arrives in the system its utilization will calculate i.e., $\frac{w}{\rho}$. If $\frac{w}{\rho} \leq 1$, then the task is schedulable else not. After every new (non-zero) task arrival, system utilization load gets incremented, as already occupied task run on CPU has utilization factor.
$\frac{w_3}{\rho_3} < 1$ and $\frac{w_1}{\rho_1} + \frac{w_2}{\rho_2} + \frac{w_3}{\rho_3} \leq 1$ (4)	Arrival of task τ_3, reaches the maximum utilization limit of processor (almost all resources are engaged with tasks execution)
$\frac{w_4}{\rho_4} < 1$ and $\frac{w_1}{\rho_1} + \frac{w_2}{\rho_2} + \frac{w_3}{\rho_3} + \frac{w_4}{\rho_4} > 1$ (5)	Task τ_4 is schedulable but it exceeds the limit of processor's utilization.
$\frac{w_5}{\rho_5} < 1$ and $\frac{w_1}{\rho_1} + \frac{w_2}{\rho_2} + \frac{w_3}{\rho_3} + \frac{w_4}{\rho_4} + \frac{w_5}{\rho_5} > 1$ (6)	Similarly, task τ_5 is schedulable but due to task τ_4 it has to wait for its execution.

Now, here the arrival of tasks τ_4 *and* τ_5, exceeds the CPU utilization, as it becomes more than 1. Here, (Tab. 5) τ_4 *and* τ_5 tasks are victim tasks that can create a fault in the system by overloading it. Overloading means resources of CPU engage (critical) in the execution of previous tasks (τ_1, τ_2, τ_3) such that the system may start missing deadlines. Hence, victim tasks have either to wait for their turn or migrate on other processors. Hence, the following two conditions arise in the system:

I. The system is occupied in the execution of tasks τ_1, τ_2, τ_3. Due to which remaining tasks have to wait for their execution, this increases by waiting time (Δt) and thus affect approaching of deadline. As $a_1, a_2, a_3, a_4, a_5, a_6, a_7$ are the arrival time of tasks $\tau_1, \tau_2, \tau, \tau_4, \tau_5, \tau_6, \tau_7$ respectively with deadlines $\partial_1, \partial_2, \partial_3, \partial_4, \partial_5, \partial_6, \partial_7$ and execution time $w_1, w_2, w_3, w_4, w_5, w_6, w_7$. Δt is the amount of waiting time that affects the assigned deadline. Following will be the scenario of scheduled victim tasks or tasks arrive after victim tasks:

Initially tasks fulfill the condition $a_4 < \partial_4$, allowed to enter in semaphore, due to resource unavailability task τ_4 has to wait for Δt amount of time i.e., $a_4 + \Delta t < \partial_4$. This increment of waiting time increases the tasks in a waiting list i.e., $\tau_5, \tau_6 \ldots\ldots\ldots \tau_k$. $a_5 + (\alpha * \Delta t) < \partial_5$,$a_6 + (2\alpha * \Delta t) \leq \partial_6$, $a_7 + (m\alpha * \Delta t) = \partial_7$ and it goes on till any of the tasks meet the deadline/migrate from one processor to another in a system. In this way, waiting time affects deadlines and tasks start missing deadlines.

II. In the above case, Tab. 5 migration was not in the picture. Here, Tab. 6 if more than one processor is there in the system S, then the victim task can migrate towards another processor and can get the required resources of execution.

Suppose three processors are there in a system S and each processor is engaged in the execution of tasks. Following are the tasks involved in each processor:

Table 6: Demonstration with migration

$S = \{P_1, P_2, P_3\}$		$P_i \in S, where i \leq 3$
$P_1 = \{\tau_{11}, \tau_{12}, \tau_{13}, \tau_{14}\}$ $P_2 = \{\tau_{21}, \tau_{22}, \tau_{23}, \tau_{24,}\tau_{25}\}$ $P_3 = \{\tau_{31,}\tau_{32}, \tau_{33}\}$		$P_i = i^{th} processor\ of\ the\ system\ S$ $where\ i \leq 3\ (in\ mentioned\ scenario)$ $\tau_{ij} = j^{th}\ task\ of\ processor\ i$
$\frac{w_{11}}{\rho_{11}} + \frac{w_{12}}{\rho_{12}} + \frac{w_{13}}{\rho_{13}} + \frac{w_{14}}{\rho_{14}} < 1$	(7)	After passing the schedulability criteria, CPU utilization has been incremented by task utilization ($\frac{w}{\rho}$) value. The utilization of P_1 *and* P_3 is less than 1, which means resource availability is there. On the other side, P_2 utilization is greater than 1, which means it is exhausted and the remaining tasks will miss the deadline. Hence, few tasks of P_2 can migrate towards P_1 and P_3.
$\frac{w_{21}}{\rho_{21}} + \frac{w_{22}}{\rho_{22}} + \frac{w_{23}}{\rho_{23}} + \frac{w_{24}}{\rho_{24}} + \frac{w_{25}}{\rho_{25}} > 1$	(8)	
$\frac{w_{31}}{\rho_{31}} + \frac{w_{32}}{\rho_{32}} + \frac{w_{33}}{\rho_{33}} < 1$	(9)	
$\frac{w_{11}}{\rho_{11}} + \frac{w_{12}}{\rho_{12}} + \frac{w_{13}}{\rho_{13}} + \frac{w_{14}}{\rho_{14}} + \frac{w_{25}}{\rho_{25}} > 1$	(10)	Migration of task τ_{25} of P_2 exceeds the utilization of processor P_1 i.e., greater than 1 or more than 100% (enough resources are not available). Hence, for timely execution migrate this task to processor P_3.
$\frac{w_{21}}{\rho_{21}} + \frac{w_{22}}{\rho_{22}} + \frac{w_{23}}{\rho_{23}} + \frac{w_{24}}{\rho_{24}} < 1$	(11)	
$\frac{w_{31}}{\rho_{31}} + \frac{w_{32}}{\rho_{32}} + \frac{w_{33}}{\rho_{33}} < 1$	(12)	
$\frac{w_{11}}{\rho_{11}} + \frac{w_{12}}{\rho_{12}} + \frac{w_{13}}{\rho_{13}} + \frac{w_{14}}{\rho_{14}} < 1$	(13)	Now, after two migrations, task τ_{25} can meet the deadline on processor P_3. Hence, the migration rate will be 1 out of 5 tasks. if no processor will available. So, in this way with the help of migration, missing deadlines (task failure) can be minimized.
$\frac{w_{21}}{\rho_{21}} + \frac{w_{22}}{\rho_{22}} + \frac{w_{23}}{\rho_{23}} + \frac{w_{24}}{\rho_{24}} < 1$	(14)	
$\frac{w_{31}}{\rho_{31}} + \frac{w_{32}}{\rho_{32}} + \frac{w_{33}}{\rho_{33}} + \frac{w_{25}}{\rho_{25}} < 1$	(15)	

Hence, we can say that CPU utilization, victim tasks and migration of tasks affect the successful execution of upcoming tasks.

EDF, RMS and Joint EDF-RMS algorithms implement and test up to 2100 tasks. These tasks divide into the transaction of 300, 600, 900, 1200, 1500, 1800 and 2100 tasks. As fault tolerance is the main point of concern here, the relation of fault with task failure, CPU utilization and migration of victim tasks are under consideration.

-Fault rate *vs*. Failure rate

Fig. 3 demonstrates the number of victim tasks out of total tasks in a given transaction. In EDF, tasks exceed the CPU utilization of more than 1, in RMS $n(2^{1/n}-1)$ and Joint EDF-RMS 0.81. Fig. 4, explains the failure rate (tasks unable to execute on time) of tasks in given algorithms. Fig. 5, represents the average fault rate and failure rate. In EDF on an average 22 tasks fails due to 3 victim tasks, similarly, 22 tasks fail due to 2 victim tasks in RMS but 18 tasks miss the deadline due to 3 victim tasks. Hence, the Joint EDF-RMS algorithm performance is better as compared to the remaining two algorithms.

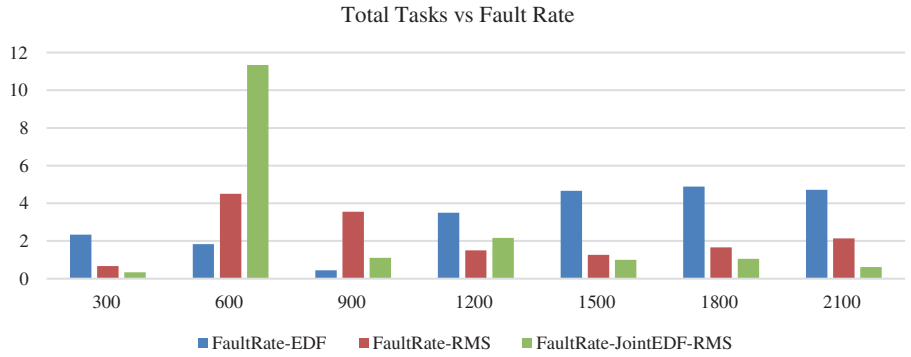


Figure 3: Arrived tasks *vs*. fault rate calculations based on EDF, RMS and Joint EDF-RMS algorithm

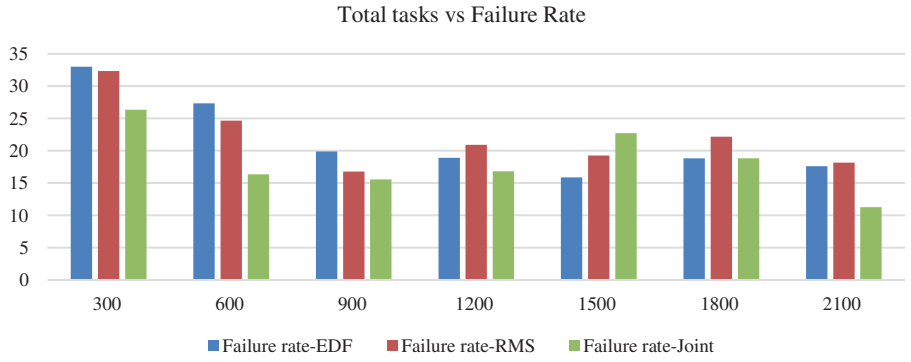


Figure 4: Arrived tasks *vs*. failure rate of tasks calculations based on EDF, RMS and Joint EDF-RMS algorithm

–Processor utilization *vs*. failure rate

Up to three processors use in this simulation. The Fig. 6, demonstrates the effect of total CPU utilization on the failure of tasks. As the participation of processors in the system increases, the total CPU utilization escalates accordingly. The behaviour of system while using the EDF algorithm illustrates in the Fig. 6. When it increases the limit by more than 3 ($U \leq 1$), tasks miss the deadline due to victim tasks.

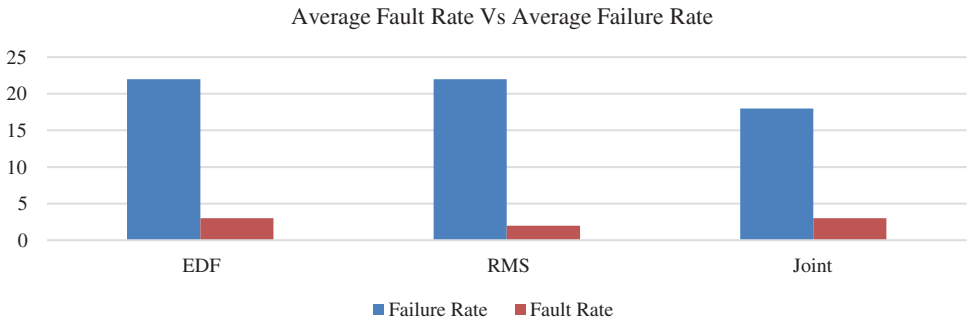
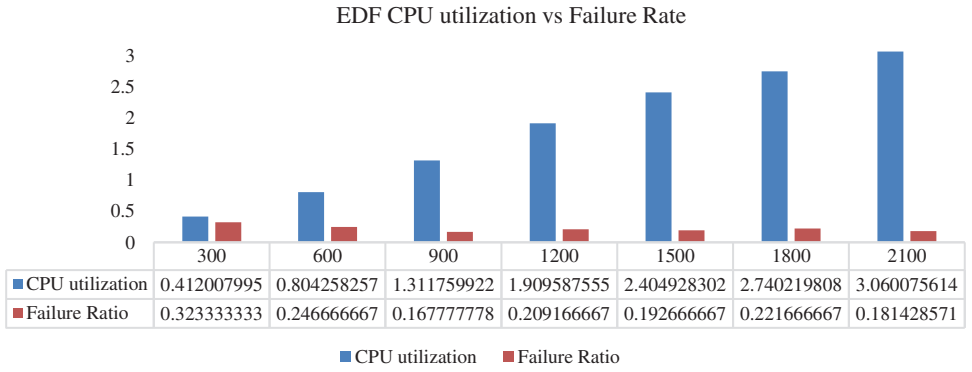


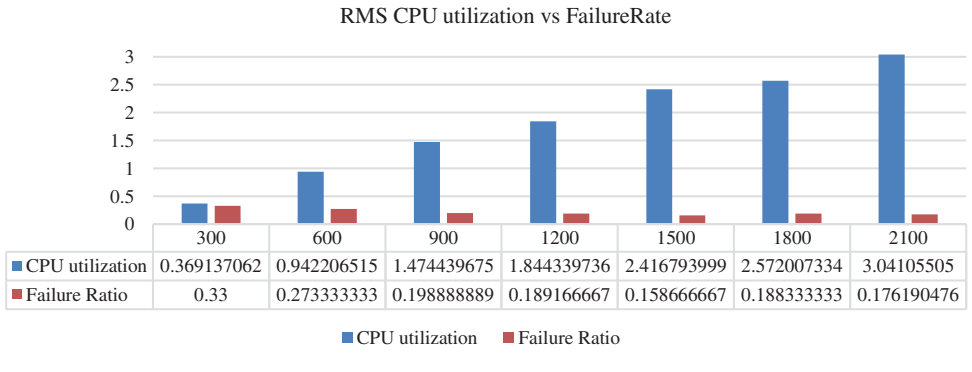
Figure 5: Average Impact of fault rate on failure rate in EDF, RMS and Joint EDF-RMS algorithm



	300	600	900	1200	1500	1800	2100
CPU utilization	0.412007995	0.804258257	1.311759922	1.909587555	2.404928302	2.740219808	3.060075614
Failure Ratio	0.323333333	0.246666667	0.167777778	0.209166667	0.192666667	0.221666667	0.181428571

Figure 6: CPU utilization *vs*. failure rate based on EDF scheduling algorithm

Fig. 7 explains the total CPU utilization effect on failure task when the RMS algorithm is in use. In Joint EDF-RMS, RMS algorithm uses to handle arrival of tasks in global queue and EDF algorithm use with threshold limit 0.81 in every processor. Hence, in Fig. 8 the failure ratio is less as compared to others due to the threshold limit of EDF algorithms and usage of RMS for priority assignment.

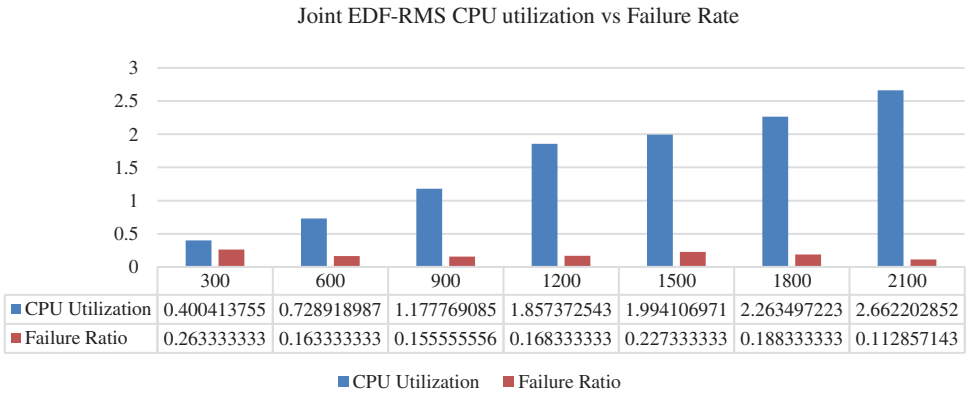


	300	600	900	1200	1500	1800	2100
CPU utilization	0.369137062	0.942206515	1.474439675	1.844339736	2.416793999	2.572007334	3.04105505
Failure Ratio	0.33	0.273333333	0.198888889	0.189166667	0.158666667	0.188333333	0.176190476

Figure 7: CPU utilization *vs*. failure rate based on RMS algorithm

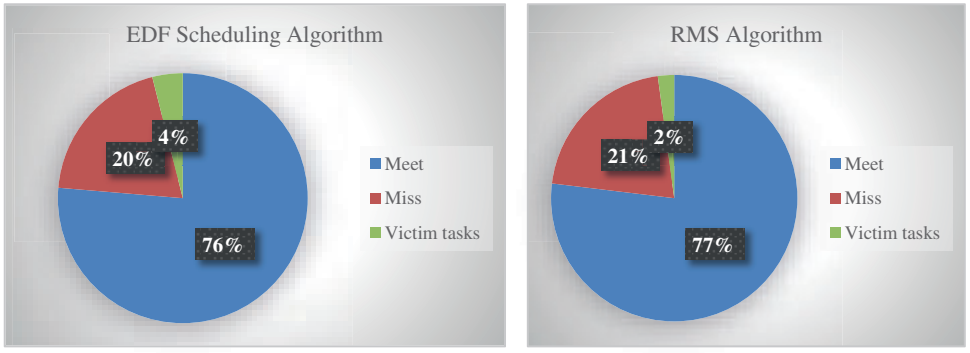
CPU utilization is the main parameter due to which victim tasks can recognize and migrate to other processors, that helps in failure tasks reduction. Fig. 9 are showing the complete result of all three scheduling algorithms. On average for every 1200 tasks the failure percentage of Joint EDF-RMS is less i.e., 17% as compared to the remaining two scheduling algorithms. Similarly, due to the

application of migration technique on 0.81% threshold, the victim tasks percentage is also very less i.e., only 2% in Joint EDF-RMS algorithm, whereas in EDF and RMS victim tasks percentage is greater or equal i.e., 4% and 2%, respectively.

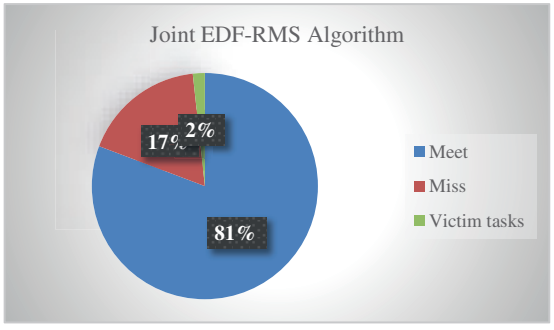


	300	600	900	1200	1500	1800	2100
CPU Utilization	0.400413755	0.728918987	1.177769085	1.857372543	1.994106971	2.263497223	2.662202852
Failure Ratio	0.263333333	0.163333333	0.155555556	0.168333333	0.227333333	0.188333333	0.112857143

Figure 8: CPU utilization *vs*. failure rate based on Joint EDF-RMS algorithm



(a) Task's status in EDF Scheduling Algorithm (b) Task's status in RMS Algorithm



(c) Task's status in Joint EDF-RMS Algorithm

Figure 9: Task's status in all three scheduling algorithms based on average number of tasks

6 Conclusion and Future Work

The fault tolerance mechanisms in RTS with the help of migration techniques in distributed systems successfully simulated. Basic scheduling algorithms (EDF and RMS) select to evaluate the performance of hybrid scheduling algorithm (Joint EDF-RMS) with fault tolerance mechanism. The vital role of fault tolerance algorithms in various applications discuss in previous sections. Here, fault tolerance implant in new hybrid algorithm (Joint EDF-RMS) and simulate in Real Time Distributed Systems. How overloaded (victim) task affect the execution of upcoming or enqueuer tasks have been explained with the help of lemma (with or without migration) and simulation. Overall, the behaviour of EDF, RMS and Joint EDF-RMS algorithm evaluates in faulty environment of RTDS. The Joint EDF-RMS algorithm handle faults efficiently due to the presence of both EDF and RMS algorithms that overcome limitations of each other. Additionally, usage of RMS algorithm in Backup processors improve the performance because this algorithm is good to handle overloading conditions due to its schedulability criteria.

This fault tolerance concept will implement in information-theoretic entropy based work [30–32]. The simulation of information theoretic entropy based EDF scheduling algorithm is already done and in future, its behaviour in faulty environment will check by using fault tolerance mechanism use in this paper.

Funding Statement: Deepak Dahiya would like to thank Deanship of Scientific Research at Majmaah University for supporting this work under Project No. R-2022-56.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] P. A. Laplante, *Real-time Systems Design and Analysis*, 3rd ed., New York: Wiley, IEEE press, 2004.
[2] J. W. S. Liu, *Real-Time Systems*, Upper Saddle River, New Jersey: Prentice Hall, 2000.
[3] R. Alur and D. Dill, "Automata for modeling real-time systems," in *Int. colloquium on automata, languages and programming*. Berlin, Heidelberg: Springer, 1990.
[4] H. Aysanüseyin, "Fault-tolerance strategies and probabilistic guarantees for real-time systems," Ph.D. dissertation. Mälardalen University, Sweden, 2012.
[5] L. I. U. Huai and F. E. I. Shu-Min, "A fault-tolerant scheduling algorithm based on EDF for distributed control systems," *Journal of Software*, vol. 14, no. 8, pp. 1371–1378, 2003.
[6] Y. Guo, D. Zhu and H. Aydin, "Generalized standby-sparing techniques for energy-efficient fault tolerance in multiprocessor real-time systems," in *Proc. 19th Int. Conf. on Embedded and Real-Time Computing Systems and Applications*, Taipei, Taiwan, IEEE, pp. 62–71, 2013.
[7] S. Ghosh, R. Melhem, D. Mossé and J. S. Sarma, "Fault-tolerant rate-monotonic scheduling," *Real-Time Systems*, vol. 15, no. 2, pp. 149–181, 1998.
[8] R. Sharma, N. Nitin, M. A. R. AlShehri and D. Dahiya, "Priority-based joint edf-rm scheduling algorithm for individual real-time task on distributed systems," *Journal of Supercomputing*, vol. 77, no. 1, pp. 890–908, 2021.
[9] Y. Wu, D. Liu, X. Chen, J. Ren, R. Liu *et al.,* "MobileRE: A replicas prioritized hybrid fault tolerance strategy for mobile distributed system," *Journal of Systems Architecture*, vol. 118, no. 10–11, pp. 102217, 2021.
[10] A. Yoo, Y. Wang, R. Sinha, S. Mu and T. Xu, "Fail-slow fault tolerance needs programming support," in *2021 the Workshop on Hot Topics in Operating Systems*, Ann Arbor, Michigan, pp. 228–235, 2021.

[11] P. Gupta, P. K. Sahoo and B. Veeravalli, "Dynamic fault tolerant scheduling with response time minimization for multiple failures in cloud," *Journal of Parallel and Distributed Computing*, vol. 158, no. 1, pp. 80–93, 2021.
[12] A. Glonina and V. Balashov, "A stopwatch automata-based approach to schedulability analysis of real-time systems with support for fault tolerance techniques," in *Proc. 2021 Int. Conf. on Dependability and Complex Systems*, Cham, Springer, pp. 116–125, 2021.
[13] S. Saha, A. Adetomi, X. Zhai, S. Kasap, S. Ehsan *et al.,* "EnSuRe: Energy & accuracy aware fault-tolerant scheduling on real-time heterogeneous systems," in *Proc. 27th Int. Sym. on On-Line Testing and Robust System Design (IOLTS)*, Italy, IEEE, pp. 1–4, 2021.
[14] B. Ranjbar, A. Hosseinghorban, M. Salehi, A. Ejlali and A. Kumar, "Toward the design of fault-tolerance- and peak-power-aware multi-core mixed-criticality systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1, 2021. DOI 10.1109/TCAD.2021.3082495.
[15] T. Nolte, G. Rodríguez-Navas, J. Proenza, S. Punnekkat and H. Hansson, "Towards analyzing the fault-tolerant operation of server-CAN," in *Proc. Int. Conf. on Emerging Technologies and Factory Automation*, Catania, Italy, IEEE, vol. 1, pp. 4, 2005.
[16] M. Saadoon, S. H. A. Hamid, H. Sofian, H. H. Altarturi, Z. H. Azizul *et al.,* "Fault tolerance in big data storage and processing systems: A review on challenges and solutions," *Ain Shams Engineering Journal*, vol. 13, no. 2, pp. 101538, 2021.
[17] S. Bansal, R. K. Bansal and K. Arora, "Energy efficient backup overloading schemes for fault tolerant scheduling of real-time tasks," *Journal of Systems Architecture*, vol. 113, pp. 101901, 2021.
[18] A. Burns, A. Welling, K. Ramamritham, J. Hooman, S. Schneider *et al.,* "Real time systems: Specification, verification and analysis," M. Joseph (Ed.), vol. 62. New York, NY, USA: Prentice Hall, 1996.
[19] K. Berns, A. Köpper and B. Schürmann, Real time systems in technical foundations of embedded systems, In *Lecture Notes in Electrical Engineering*, vol. 732, Cham: Springer, pp. 339–361, 2021.
[20] E. Ertugrul and O. K. Sahingoz, "Fault tolerance in real time systems: A review," in *Proc. Int. Conf. on Intelligent Systems Design and Applications*, Cham, Springer, pp. 283–293, 2017.
[21] A. C. Persya and T. R. G. Nair, "Fault tolerant real time systems," in *Proc. 2008 Int. Conf. on Managing Next Generation Software Application (MNGSA-08)*, Coimbatore, India, pp. 177–180, 2008.
[22] J. C. Laprie, "Dependable computing and fault tolerance: Concepts and terminology," in *Proc. 15th Int. Sym. on Fault-Tolerant Computing (FTSC-15)*, Ann Arbor, Michigan, IEEE, vol. III, pp. 2–11, 1985.
[23] J. Kaur and S. Kinger, "Analysis of different techniques used for fault tolerance," *International Journal of Computer Science and Information Technologies (IJCSIT)*, vol. 5, no. 3, pp. 4086–4090, 2014.
[24] A. Avizienis, "Fault-tolerant systems," *IEEE Transactions on Computers*, vol. 25, no. 12, pp. 1304–1312, 1976.
[25] A. A. Bertossi and L. V. Mancini, "Scheduling algorithms for fault-tolerance in hard-real-time systems," *Real-Time Systems*, vol. 7, no. 3, pp. 229–245, 1994.
[26] N. Audsley, A. Burns, R. Davis, K. Tindell and A. Wellings, Real time system scheduling. in *Predictably Dependable Computing Systems*, Berlin, Heidelberg: ESPRIT Basic Research Series, Springer, pp. 41–52, 1995.
[27] R. Mehta and M. Upasna, "Real time system fault tolerance scheduling algorithm," *International Journal of Computer Science and Mobile Computing*, vol. 3, no. 9, pp. 297–306, 2014.
[28] A. Silberschatz, G. Gagne and P. B. Galvin, *Operating System Concepts*, United States, 8th ed., John Wiley & Sons, 2008.
[29] R. Sharma and Nitin, "Performance evaluation of new joint EDF-RM scheduling algorithm for real time distributed system," *Journal of Engineering, Hindawi publishers*, vol. 2014, no. 1, pp. 1–13, 2014.
[30] R. Sharma and Nitin, "Entropy, a new dynamics governing parameter in real time distributed system: A simulation study," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 29, no. 6, pp. 562–586, 2014.

[31] R. Sharma and Nitin, "Visualization of information theoretic maximum entropy model in real time distributed system," in *Proc. 3rd Int. Conf. on Advances in Computing and Communications*, Cochin, IEEE, pp. 282–286, 2013.
[32] R. Sharma and Nitin, "Evaluation and comparison of load balancing in RTDS using Information theoretic entropy," in *Proc. Int. Advance Computing Conf. (IACC)*, Gurgaon, India, IEEE, pp. 674–679, 2014.