Comparative Study of LST and SJF Scheduling Algorithm in Soft Real-Time System with its Implementation and Analysis

Jay Teraiya
Information and Technology Department
Marwadi Education Foundation, Rajkot
jay.teraiya@gmail.com

Apurva Shah
Computer Science and Engineering Department
The Maharaja Sayajirao University of Baroda
apurva.shah-cse@msubaroda.ac.in

Abstract – **The Least Slack Time First (LST) algorithm is a dynamic scheduling algorithm and also known as Least Laxity First. It decides the dynamic priority of the task based on slack time; The task having minimum slack time will be considered the highest priority. It is the most suitable algorithm for scheduling of tasks in soft Real-Time Operating System (RTOS). The Shortest Job First (SJF) algorithm is a static scheduling algorithm and decides the priority of the task based on execution time required for a given task. Task which has minimum execution time considered as the highest priority task in SJF. It is not directly used for scheduling Soft Real-Time system. In this paper, we have implemented the LST and SJF for the soft real-time operating system.These algorithms have been executed on periodic task set, and observations are gathered. We have observed Success Ratio & Effective CPU Utilization and compared both the algorithm in the same conditions. It is noted that the LST algorithm performs well in underload scenario but not well in an overload situation. SJF not able to schedule specific task even in underload situation but it comparatively performs well in an overload situation. Practical experiments have been conducted on a large dataset. Data Set contains 7500 task set, and each task set includes 1 to 9 processes. CPU load for each process set varies from 0.5 to 5. It has been tested on 500-time unit to validate the correctness of both algorithms.**

Keywords – **LST; SJF; RM; EDF; Scheduling; Real-Time Systems**

I. INTRODUCTION OF RTOS AND SCHEDULING ALGORITHMS

Since last few years, Real-Time systems usage has been increasing in time critical application. Designing systems which are expected to deliver real-time results involves an equal emphasis on managing timing constraints of various functionalities of the system. Processes in the real-time system has defined deadlines and need to complete the process within its deadline. Real-time systems need a scheduling algorithm that assign the tasks to the processor by taking into consideration the deadline constraints as well also supporting other requirements of scheduling. Depending upon the deadline constraints, real-time systems are categorized in hard and soft real-time systems. In a hard real-time system, if it is fail when the deadline is missed than results will be useless. However in a soft real-time system, upon missing the dead line, results wouldn't become useless but performance of the system may be degreded. [1].

In a real-time system, the appropriate scheduling approach should be selected based on the properties of the system and base on the tasks type. Real-Time system classified as Hard, Soft and Firm real-time system. Its task set classified as a Periodic, Aperiodic and Sporadic task. Task Set can also be categorized based on preemptive or non-preemptive task.

The general scheduling algorithm is looking for an order according to which the task should be executed such that various constraints are satisfied. The task is characterized by its execution time, arrival time, deadline, and resource requirements. Scheduling algorithm can be classified into two general categories based on its characteristics, static and dynamic. It depends on the approach they use. Static priority schedulers assigned a single priority value to each task during initialization throw out the scheduling process. Example of static priority scheduling is Shortest Job First(SJF), Rate Monotonic (RM) and Deadline Monotonic (DM).The dynamic schedulers change the priorities of tasks depending on the current situation of a given system. Example of dynamic priority scheduling is Earliest Deadline First (EDF) and Least Slack Time First (LST). EDF and LST algorithms are most effective under the situation that the jobs are preemptable, there is a single processor, and the processor is underloaded. However, these algorithms performance decreases quickly in overloaded condition[2][3]. In this paper, we have compared static priority algorithm SJF and dynamic priority algorithm LST with a different aspect. We are observing these algorithms regarding Effective CPU utilization and Success Ration. We observed both algorithms in underload and overload situation.

This paper is organized as follows: The LST and SJF algorithm has been discussed in Section II and III respectively. Section IV explains related work done on LST and SJF. Section V represents System Consideration and Task Model. Section VI describes experimental setup and performance measuring parameters. Section VII represents the results analysis of both algorithms and the paper is wrapping up with a brief conclusion in Section VIII.

II. THE LEAST SLACK TIME FIRST (LST) ALGORITHM

The least slack time first algorithm is a dynamic pre-emptive scheduling algorithm. The highest priority is assigned to the task having the small slack time. The slack time l is defined as per the following equation [4].

$$l = d - c - t \qquad (1)$$

Where,
t = current time
d = deadline
c = remaining execution time

The scheduling algorithm is necessary to execute when a currently running task completes or new task arrives. The flowchart of the algorithm has been shown in Figure 1.When a new task arrives or the currently executing task is finished, the scheduling algorithm will run and calculate the slack time for each task based on Equation 1. The new task selected for execution which has minimum slack time.

III. THE SHORTEST JOB FIRST (SJF) ALGORITHM

The Shortest Job First algorithm is a static priority scheduling algorithm. The highest priority is assigned to the task having the small execution time [5].

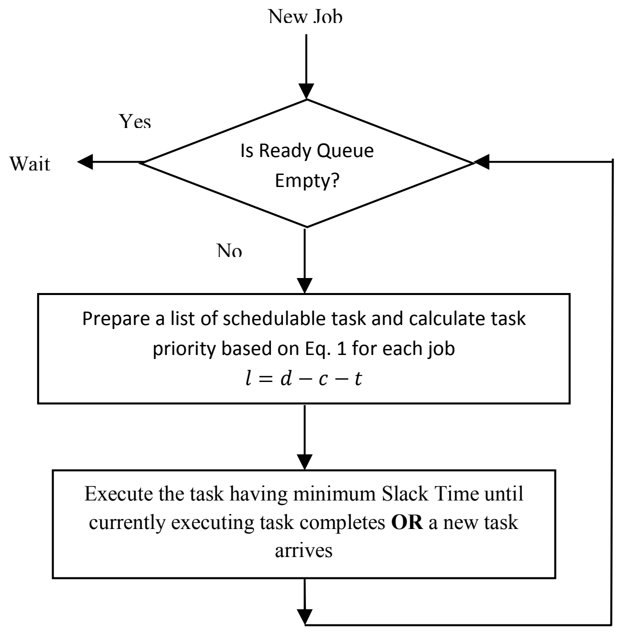
New Job
Yes
Wait
Is Ready Queue Empty?
No
Prepare a list of schedulable task and calculate task priority based on Eq. 1 for each job
$l = d - c - t$
Execute the task having minimum Slack Time until currently executing task completes **OR** a new task arrives

Fig.1- LST

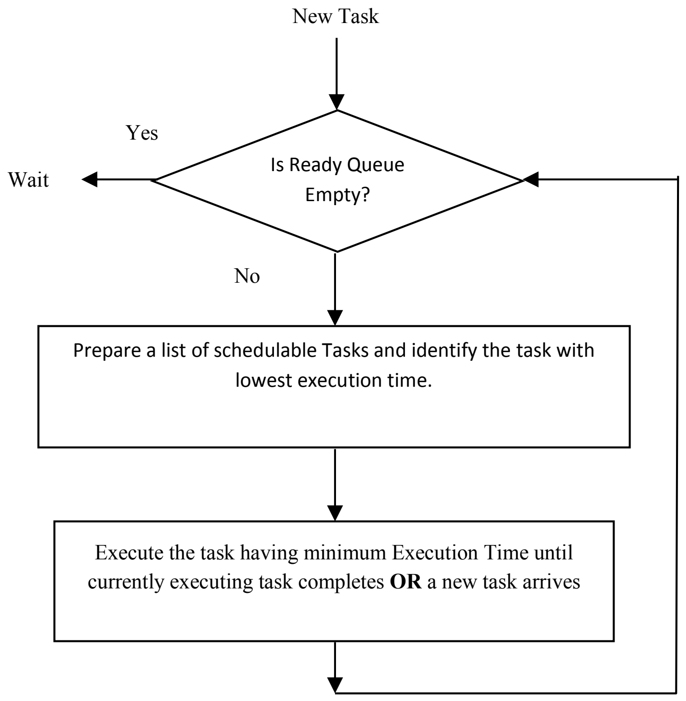
New Task
Yes
Wait
Is Ready Queue Empty?
No
Prepare a list of schedulable Tasks and identify the task with lowest execution time.
Execute the task having minimum Execution Time until currently executing task completes **OR** a new task arrives

Fig.2 - SJF

The execution time of a task is already known in the Real-time system and defined as CPU time required for completing the task. The scheduling algorithm is necessary to execute when a currently running task completes or new task arrives. The flowchart of the algorithm has been shown in Figure 2:

As shown in Figure 2, when a new task arrives, or the currently executing task is finished, the scheduling algorithm will run and identify the task with minimum execution time. The new task selected for execution which has minimum execution time.

IV. RELATED WORK ON LST AND SJF

For any set of the task, we can verify its schedulability is feasible or not. In the periodic task model, each task has its occurrence period(T), its execution time(C) and its deadline(D). The ratio U=C/T is called the utilization factor of the task and represents the fraction of processor time used by that task. At a given point of time for a set of N task, utilization factor can be calculated by the following equation.

$$\mathrm{U_p} = \sum_{i=1}^{n} \frac{\mathrm{C_i}}{\mathrm{T_i}} \qquad (2)$$

$\mathrm{U_p}$ is called the total processor utilization factor and represents the fraction of processor time used by the periodic task set. If $\mathrm{U_p} > 1$ no feasible schedule exists for the task set with an algorithm, and it is overload condition. If $\mathrm{U_p} < 1$, the feasibility of the schedule depends on the task set parameters and the scheduling algorithm used in the system [6].

In LST, tasks priorities are decided as per its slack times. The periodic task is pre-empted at the time when another task with less slack time arrives. It also take care that none of the tasks miss their deadlines. The dynamic scheduler LST performes better than the static scheduler in under load situation, and it can schedule the entire task set when $\mathrm{U_p} < 1$[1].

In SJF, tasks priorities are decided based on its execution time required. The scheduler selects the waiting task with the shortest execution time. SJF is advantageous because of easy to implement and because it maximizes process throughput. It also minimizes the average amount of time each task has to wait until its execution is complete [7]. We did not find any experimental setup which only uses SJF algorithm for Real-Time Scheduling. SJF has been used with EDF algorithm to decide the group priority in a non-preemptive scheduling algorithm for soft real-time systems [5]. We have applied SJF as a single algorithm in Soft Real-Time system in this paper.

V. THE SYSTEM CONSIDERATION AND TASK MODEL

We have assumed that the system knows task deadline and necessary information to compute the time required to execute the task on when the task is released. The task set is considered pre-emptive. We have considered that the system is not having resource clash problem. Moreover, pre-emption and the scheduling algorithm acquire no overhead.

In soft real-time systems, each task has a positive value. The goal of the system is to gain maximum value. If a task meets the deadline, then the system considers its value. If a task missed the deadline, then the system gets less value from the task [8]. In a particular case of soft real-time systems, called a firm real-time system, if task misses its deadline, then no value will be considered, but there is no disaster as well [9]. In this paper, we have implemented LST and SJF algorithm which applies to the soft real-time system. The value of the task has been considered the same as its computation time required [10].

VI. EXPERIMENTAL SETUP AND PERFORMANCE MEASURING PARAMETER

We have implemented LST and SJF algorithm in C programming language. These scheduling algorithms schedule the task when a new task arrives or currently executing task completes. These algorithms execute periodic tasks for validating their performance. For periodic tasks, $\mathrm{U_p}$ (processor utilization factor) can be defined as the summation of the ratio of executable time and period of each task. We considered it as Load of the system and calculated as per the Equation 1. To generate the task set we have developed one module in C language which produces random periodic task set. Using this tool we have built 7500 task set and each task set containing 1 to 9 tasks. These task sets load varies from 0.5 to 5.0. Overall LST and SJF have been tested with approx 35,000 task scheduling process on 500-time unit to get the results. Performance of LST and SJF has been measured based on following two parameters.

1. **SR (Success Ratio)** - In real-time systems, achieve the deadline is main key aspect, and we are concerned about finding whether the task is meeting the deadline. Therefore the most appropriate performance parameter is the Success Ratio(SR). SR defined as:

$$SR = \frac{Number\ of\ Task\ successfully\ scheduled}{Total\ Number\ of\ Task\ arrived} \quad (3)\ [11]$$

2. **ECU (Effective CPU Utilization)** - It is essential that how efficiently the scheduler utilizes the processes, particularly during overloaded condition. Therefore, the other performance metric is Effective CPU utilization (ECU). ECU defined as:

$$ECU = \sum_{i \in R} \frac{V_i}{T} \quad (4)\ [12]$$

Where,

- V represents the value of task and,
  - V = Computation time of the task, if the task completes within its deadline.
  - V = 0, if the task fails to meet the deadline.
- R represents a set of tasks, which are scheduled successfully, i.e., meets its deadline.
- T represents the total time of scheduling.

.

VII. RESULT AND ANALYSIS

Table I and Table II show the results gathered by executing LST and SJF algorithm on the simulator. Table I represents the scenario where task set contains 1 to 9 task and Load is less than 1 or equal to 1 ($U_p \leq 1$). Results show that ECU values remain nearly the same for LST and SJF, but SR is not 100% in case of SJF. When Load is less than 1, it means that task set is schedulable, and all process can meet their deadline, but SJF is not able to schedule all task whereas LST is successfully able to schedule these task set. It means in under load situation LST gives a guarantee to schedule all task, so it is advisable to use LST instead of SJF.

Table II represents the scenario where task set contains 1 to 9 task and Load is greater than 1 ($U_p > 1$). Results show waste in ECU and SR values for LST and SJF. When Load is greater than 1, it means that the task set is not schedulable

TABLE I

(LST AND SJF PERFORMANCE IN UNDERLOAD)

Load	ECU%		SR%	
	LST	SJF	LST	SJF
0.50	49.49	49.49	100	100
0.55	54.66	54.31	100	100
0.60	59.39	59.39	100	100
0.65	64.35	64.35	100	100
0.70	69.35	69.35	100	100
0.75	74.31	74.31	100	100
0.80	79.22	79.22	100	100
0.85	84.16	84.15	100	99.99
0.90	89.16	89.00	100	99.84
0.95	94.17	93.89	99.99	99.78
1.00	99.10	96.74	100	98.74

TABLE II

(LST AND SJF PERFORMANCE IN OVERLOAD)

Load	ECU%		SR%	
	LST	SJF	LST	SJF
1.05	16.09	56.63	15.84	73.49
1.10	8.33	63.60	7.90	75.98
1.15	5.58	62.66	5.06	73.66
1.20	4.21	70.08	3.67	73.06
1.25	3.56	73.20	3.06	77.47
1.30	3.09	72.24	2.53	75.34
1.35	2.63	70.99	2.09	71.55
1.40	2.20	76.57	1.71	73.80
1.45	2.01	74.45	1.52	68.76
1.50	1.83	80.07	1.33	69.96
1.60	1.77	77.26	1.29	67.20
1.70	1.58	79.16	1.07	64.60
1.80	1.45	77.28	0.95	63.04
1.90	1.31	77.53	0.85	62.21
2.00	1.19	78.10	0.76	61.00
2.25	1.13	76.95	0.65	55.91
2.50	0.98	74.97	0.54	49.92
2.75	0.91	74.42	0.47	46.83
3.00	0.86	77.23	0.40	41.67
3.50	0.75	73.37	0.33	36.76
4.00	0.73	79.57	0.27	34.09
4.50	0.71	71.58	0.24	27.74
5.00	0.66	78.22	0.20	25.71

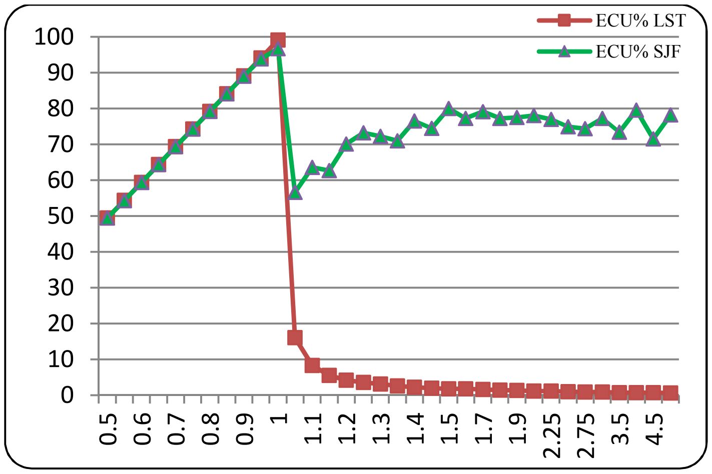

Fig. 3 – ECU% Vs Load

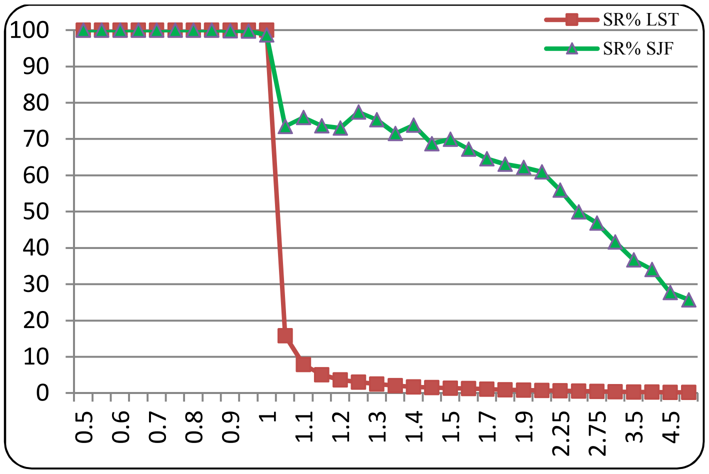

Fig. 4 – SR% Vs Load

and few tasks will miss their deadline. Table 2 observations reflect that in slightly overload situation LST performance degrades very poorly whereas SJF able to meet the deadline for few of their task set. It means in overload situation SJF gives better performance than LST. Figure 3 and 4 provides a graphical representation of Table 1 and Table 2 respectively.

VIII. CONCLUSION

The LST and SJF are implemented for scheduling of soft real-time system with a single processor and pre-emptive task sets. These algorithms are simulated with periodic task sets; results are obtained and compared. Observation suggests that dynamic algorithm LST performs well in underload situation and able to schedule most of all task when Load is 1. In overload (Load is > 1) situation, LST performs poorly. So in underload, LST is advisable but not with an overload situation. Static algorithm SJF performs moderately to underload situation. It has been observed that with specific task set even it is possible that all tasks meet their deadline but SJF is failed to schedule it. So in underload, SJF is not advisable, but in overload, it performs well compared to LST. This happens because the characteristic of SJF, it is selecting the task which has minimum execution time and because of that, it has more chance to meet their deadline even in an overload condition. Because of that ECU% and SR% is good in comparison with LST in an overload situation. LST work on slack time, which does not only depend on execution time. It iscalculated based on deadline and remaining execution time because of that ECU% and SR% decrease very fast in an overload situation.

In the future, we can propose a new algorithm which will use the characteristics of LST and SJF. Therefore it may perform well in overload and underload situation.

REFERENCES

[1] Belagali, R., Kulkarni, S., Hegde, V., & Mishra, G. (2016, December). “Implementation and validation of dynamic scheduler based on LST on FreeRTOS”, in *Electrical, Electronics, Communication, Computer and Optimization Techniques (ICEEECCOT),* Mysore, India, pp. 325-330.

[2] A. Mohammadi and S. G. Akl, “Scheduling Algorithms for Real-Time Systems”, in *School of Computing, Queen’ s University, Kingston, Ontario*, 2005.

[3] Thakor, D., & Shah, A. (2011, December). “D_EDF: An efficient scheduling algorithm for real-time multiprocessor system”, in *Information and Communication Technologies (WICT).,2011,* Mumbai, India, pp. 1044-1049.

[4] M. Patel and B. Oza, “An Improved LLF_ DM Scheduling Algorithm for Periodic Tasks by Reducing Context Switches,” in *International Journal of Advance Engineering and Research ., 2015*, vol. 2, pp. 248–254.

[5] Li, W., Kavi, K., & Akl, R. (2007). “A non-preemptive scheduling algorithm for soft real-time systems”, in *Computers & Electrical Engineering*, Vol. 33(1), pp. 12-29.

[6] Buttazzo, G. C. (2005). “Rate monotonic vs. EDF: judgment day”, in *Real-Time Systems*, Vol. 29(1), pp. 5-26.

[7] Chen, G., & Xie, W. (2011, March). “On a laxity-based real-time scheduling policy for fixed-priority tasks and its non-utilization bound”, in *Information Science and Technology (ICIST) 2011,* Tebessa, Algeria, pp. 7-10

[8] Locke, C. D. (1986). “Best-effort decision making for real-time scheduling” *(Ph. D Thesis), Computer Science Department, CMU.*

[9] Koren, G., & Shasha, D. (1995). “D_over: An Optimal On-Line Scheduling Algorithm for Overloaded Uniprocessor Real-Time Systems” in *SIAM Journal on Computing*, Vol. 24(2), pp. 318-339.

[10] A Shah , “Adaptive scheduling algorithm for real-time distributed systems ”, in *Biologically-Inspired Techniques for Knowledge Discovery and Data Mining* pp. 236-248, 2014.

[11] Ramamritham, K., Stankovic, J. A., & Shiah, P. F. (1990). “Efficient scheduling algorithms for real-time multiprocessor systems” in *IEEE Transactions on Parallel and Distributed systems*, Vol.1(2), 184-194.

[12] Shah, A., & Kotecha, K. (2010, November). “Scheduling Algorithm for Real-Time Operating Systems Using ACO” in *Computational Intelligence and Communication Networks (CICN), 2010,* Bhopal, India, pp. 617-621.

[13] J.W.S.Liu, “Real-Time Systems,” in *Pearson Education*, India, 2001.

[14] Baruah, S., Koren, G., Mishra, B., Raghunathan, A., Rosier, L., & Shasha, D. (1991). “On-line scheduling in the presence of overload” in *Proceedings 32nd Annual Symposium of Foundations of Computer Science.* Puerto Rico, USA.

Analysis of Earliest Deadline First and Rate Monotonic Scheduling Algorithm in Soft Real-Time System

Jay Teraiya
Computer Engineering Department
Marwadi University
Rajkot, India
jay.teraiya@gmail.com

Apurva Shah
Department of Computer Science and Engineering
The Maharaja Sayajirao University of Baroda
Vadodara, India
apurva.shah-cse@msubaroda.ac.in

Abstract— The Earliest Deadline First (EDF) is a dynamic scheduling algorithm, and It gives priority to the task based on its absolute deadline; the task having the nearest deadline will have the highest priority. EDF one of the best suitable for scheduling tasks with Soft Real-Time Operating System (RTOS). The Rate Monotonic (RM) algorithm is a static scheduler. It gives priority to the task based on its occurrence period, or we can say it gives priority based on the rate of the task. A task which has the lowest rate will assign the highest priority in the RM algorithm. In this paper has implemented the EDF and RM for the Soft-RTOS. These algorithms have been tested with *the periodic* task set, and observations are gathered. Algorithms are compared based on Success Ratio & Effective CPU Utilization in similar conditions. It has been observed that the EDF algorithm performs well in underload conditions, but in an overload situation, performance gets degraded. Whereas RM not able to schedule specific tasks set in underload condition but it reasonably performs well in an overload condition compare to EDF. Practical experiments have been executed with an extensive process set. Process Set contains a 6000-task set, and every task set has a different number of tasks between one to nine. Every process set also has different CPU utilization factor 0.5 to 5. These algorithms have been evaluated on a 500-time line to validate the performance in all scenarios.

Keywords—RTOS, Real-Time Systems, Scheduling, RM, EDF

1. Introduction

The usage of Real-Time based systems is getting increased day by day. Developing a system that is expected to generate real-time results need to manage timing constraints of all functionalities. All tasks in RTOS have their related deadlines and have to finish the task within the given deadline. Based on RTOS type, it is necessary to select a scheduler that assigns a task to the processor by taking into considering the timing constraints and supporting all other needs of scheduling. Based on the time criticality, real-time systems divided into three significant categories hard, soft, and firm real-time systems. In Hard RTOS, if the deadline is missed, the disaster will occur even though the miss is minor. A Soft RTOS if the deadline is missed, the disaster will not happen, but the overall performance of the system will degrade. [1].

Based on the property of the RTOS and task sequence, the appropriate scheduling method should be applied. In RTOS, the task can be categorized as Periodic, Aperiodic, and Sporadic task. It is also classified based on the non-preemptive or pre-emptive task.

The scheduler is organizing the sequence of tasks such a way that it can satisfy its different conditions. A task has characteristics like execution and arrival time. It also has a deadline, period, and other requirements. The Static and Dynamic algorithms are two types of scheduling algorithm which is used depends on the approach. The scheduler, which assigns priority only once at the time of initialization, is referred to as a Static scheduling algorithm. Rate Monotonic (RM) is one of the examples of static priority scheduler. The scheduler, which keeps changing the priority based on the current situation, is referred to as a dynamic scheduling algorithm. The Earliest Deadline First (EDF) is one of the examples of the dynamic scheduling algorithm. [2][3]. This paper has evaluated the Earliest Deadline First and Rate Monotonic schedulers with a diverse scenario. This paper has evaluated these algorithms based on two different parameters called Effective CPU utilization (ECU) and Success Ratio (SR). Paper has evaluated both schedulers in underload and overload situations [4].

Paper has been arranged in the following way: The scheduling method EDF and RM described in Section 2 and 3. Related work is described in Section 4. Algorithm Evolution Criteria and Practical Setup are described in Section 5. Section 6 discussed the analysis and evaluation of both schedulers, and the paper is ended with a conclusion in Section 7.

2. The EDF Algorithm

The EDF algorithm is a dynamic pre-emptive scheduler. It gives priority to the task based on the absolute deadline. Priorities of tasks are allocated dynamically and are inversely proportional to the absolute deadlines of the active tasks [6][10]. Figure 1 shows the flow of the EDF algorithm. When the currently executing task is completed, or a new task comes, the scheduler will run and check the absolute deadline of each active task. The task which has the earliest deadline will be selected for the next execution.

3. The RM Algorithm

The RM algorithm is a static pre-emptive scheduler. It gives priority to the task based on its Rate (task occurrence period). The task with ***the smallest*** Rate will get high priority [5][6]. The period of any task is pre-defined in RTOS and defined as the task occur again in a given duration. Figure 2 shows the flow of the RM algorithm. When the currently executing task is completed, or a new task comes, the scheduler will run and check the lowest rate of each active task. The task which has the lowest rate will be selected for the next execution [10].

4. EDF and RM Related Work

It is possible to verify the stimulability of any set of the periodic task set. The periodic task set has its deadline(D), its occurrence period(T), and its execution time(C). The utilization factor $U = \frac{C}{T}$ gives the time used by the assigned task of the processor. For any point in time, the utilization factor can be calculated with the following equation.

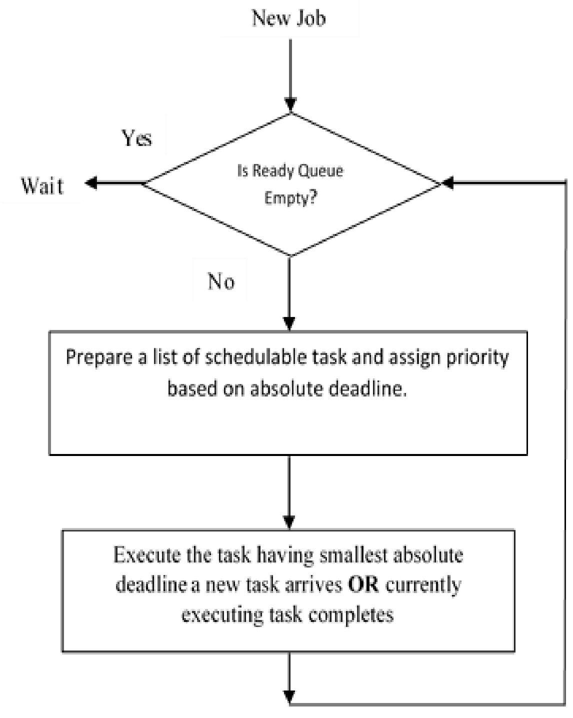
New Job
Yes
Wait
Is Ready Queue Empty?
No
Prepare a list of schedulable task and assign priority based on absolute deadline.
Execute the task having smallest absolute deadline a new task arrives OR currently executing task completes

Fig.1 - EDF

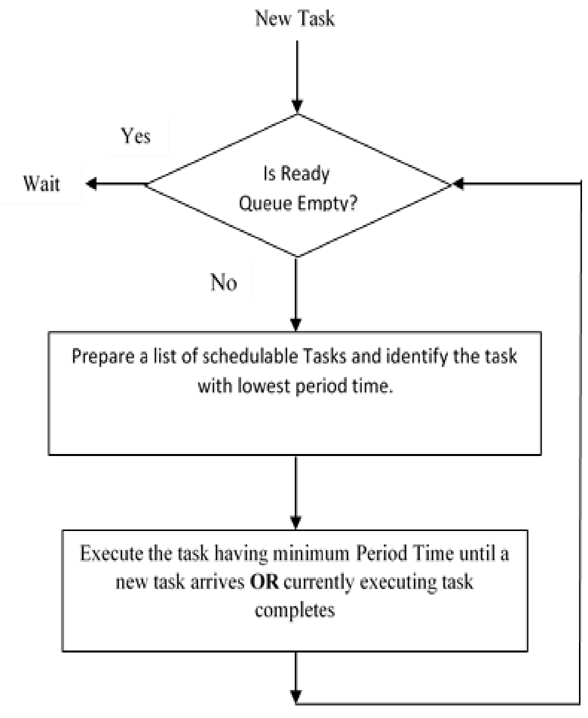
New Task
Yes
Wait
Is Ready Queue Empty?
No
Prepare a list of schedulable Tasks and identify the task with lowest period time.
Execute the task having minimum Period Time until a new task arrives OR currently executing task completes

Fig.2 -RM

$$U_p = \sum_{i=1}^{n} \frac{C_i}{T_i} \qquad (1)$$

The total CPU utilization factor $\mathrm{U_p}$ stats the fraction of processor time used by the periodic task set. The given task set is schedulable or not will be decided based on the value of $\mathrm{U_p}$. If vale of $\mathrm{U_p}$is less than 1, then it is possible to schedule the given task set, but if the value of $\mathrm{U_p}$ is greater than one than there is no scheduler exist which can schedule tasks set completely. [6].

The EDF assigns priorities based on its absolute deadline. The periodic task can be pre-empted when a new task with the smallest absolute deadline arrives. The EDF scheduler performs well compare to any other static scheduler in underload scenario, and it is possible to schedule all the task within the task set if $\mathrm{U_p} < 1$[1]. The RM assigns priorities based on its occurrence period (Rate). The scheduler chose the task from the whole ready task with the shortest period to execute next. RM has advantages like easy to implement, it has less runtime overhead, simple to evaluate, and it is predictable in overload scenario [6][7].

5. Algorithm Evaluation Criteria and Practical Setup

The Soft-RTOS task set has the required data to calculate the time required to complete the task when the task is dispatched. This paper is assuming that the task set is periodic and pre-emptive. During the evolution of these algorithms, it has been considered that the task does not have any resource clash issue, and it has also been considered that there is no overhead in the pre-emption and scheduling algorithm.

This paper evaluating EDF and RM method, and these algorithms are implemented using the C programming language. These algorithms execute and schedule the task as per Figure 1 and Figure 2. This paper has considered a periodic task set for evaluating the performance of the algorithm. The task set has been generated using a software module that is developed in C language. This module has generated a large amount of task set, which has 1 to 9 tasks in each set. Each task set has a different utilization factor, and it varies from 0.5 to 5.0 [11]. At a glance, EDF and RM have been evaluated with more than 30,000+ task to prove its performance. Each task set has been scheduled for a 500-time unit to test the effectiveness of the algorithm.

Evaluation of EDF and RM algorithms have been measured based on following two-parameter

Success Ratio (SR) - Soft RTOS expects to meet all the deadlines of a given task in the task set, and it is a crucial parameter for any scheduler to check its performance. This paper is trying to find out that any given task can meet their deadline or not. Because of that essential parameter is SR and it defines as below [8][9],

$$SR = \frac{Number\ of\ Task\ successfully\ scheduled}{Total\ Number\ of\ Task\ arrived} \quad (2)$$

Effective CPU Utilization(ECU) – This parameter will calculate the effective use of CPU. It shows the time which is used by the task to schedule the task and the task which can meet their deadline. ECU defined as:[8][9]

$$ECU = \sum_{i \in R} \frac{V_i}{T} \, (3)$$

Here,

V represents task value and,

- Value of task = Execution time of the task, if it completes its execution before its deadline.
- Value of task = 0, if the task miss deadline.
- R is a set of tasks, which are scheduled successfully, i.e., completed within their deadline.
- T is the total time of scheduling.

6. Result and Discussion

EDF and RM algorithm has been evaluated on the simulator, which is developed in the C programming language. Results have been gathered and represented in Table I and Table II. Underload scenario results have been displayed in Table I, where task set have utilization factor which is less than or equal to 1. It has been observed that EDF can meet all the deadlines, whereas RM is missing a few of them. Based on this observation, we can say that the EDF algorithm is advisable in the underload scenario compare to RM. Overload scenario results have been displayed in Table II, where task set have utilization factor which is greater than 1. Table II reflects a significant performance difference between EDF and RM in the overload scenario. If the utilization factor is more than 1 for any given task set than it is not possible to schedule a task set, and few of their task will miss their deadline. Table II observation says that EDF performance degraded very rapidly in slightly overload situations, whereas RM is still able to meet a few of their deadlines. Table I and Table II have been represented in the plotted graph in Figure 3 and Figure 4.

Table I: Underload Scenario

Load	ECU%		SR%	
	EDF	***RM***	***EDF***	***RM***
0.50	49.49	49.49	100.00	100.00
0.55	54.66	54.40	100.00	100.00
0.60	59.39	59.39	100.00	100.00
0.65	64.35	64.35	100.00	100.00
0.70	69.35	69.35	100.00	100.00
0.75	74.31	74.31	100.00	100.00
0.80	79.22	79.22	100.00	100.00
0.85	84.16	84.16	100.00	100.00
0.90	89.16	89.15	100.00	99.99
0.95	94.17	94.08	100.00	99.93
1.00	99.10	97.78	100.00	98.92

Table II : Overload Scenario

Load	ECU%		SR%	
	EDF	***RM***	***EDF***	***RM***
1.05	17.45	70.85	18.27	78.49
1.10	9.21	75.82	9.31	80.49
1.15	6.29	73.20	6.19	75.88
1.20	4.62	83.50	4.22	79.47
1.25	4.06	79.05	3.67	77.58
1.30	3.63	75.66	3.19	73.81
1.35	3.12	74.65	2.65	70.77
1.40	2.66	83.55	2.24	75.47
1.45	2.50	79.75	2.00	69.03
1.50	2.21	85.27	1.71	70.33
1.60	2.17	85.61	1.61	69.52
1.70	2.03	86.26	1.42	65.99
1.80	1.93	86.12	1.30	65.98
1.90	1.90	85.83	1.29	63.51
2.00	1.84	85.78	1.20	62.88
2.25	1.76	84.27	1.04	56.16
2.50	1.55	87.06	0.89	53.82
2.75	1.46	89.21	0.78	52.07
3.00	1.32	94.46	0.63	48.36
3.50	1.27	93.48	0.57	44.50
4.00	1.11	95.04	0.43	39.52
4.50	1.08	96.77	0.38	36.45
5.00	0.97	98.13	0.31	31.72

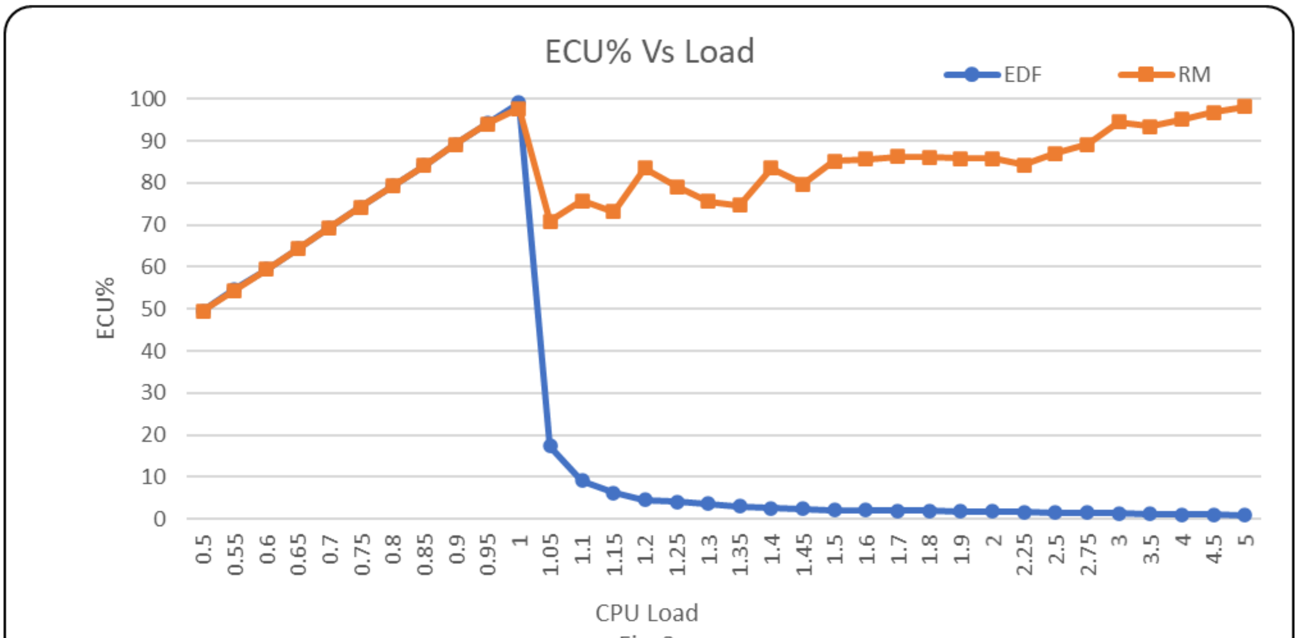

Fig. 3

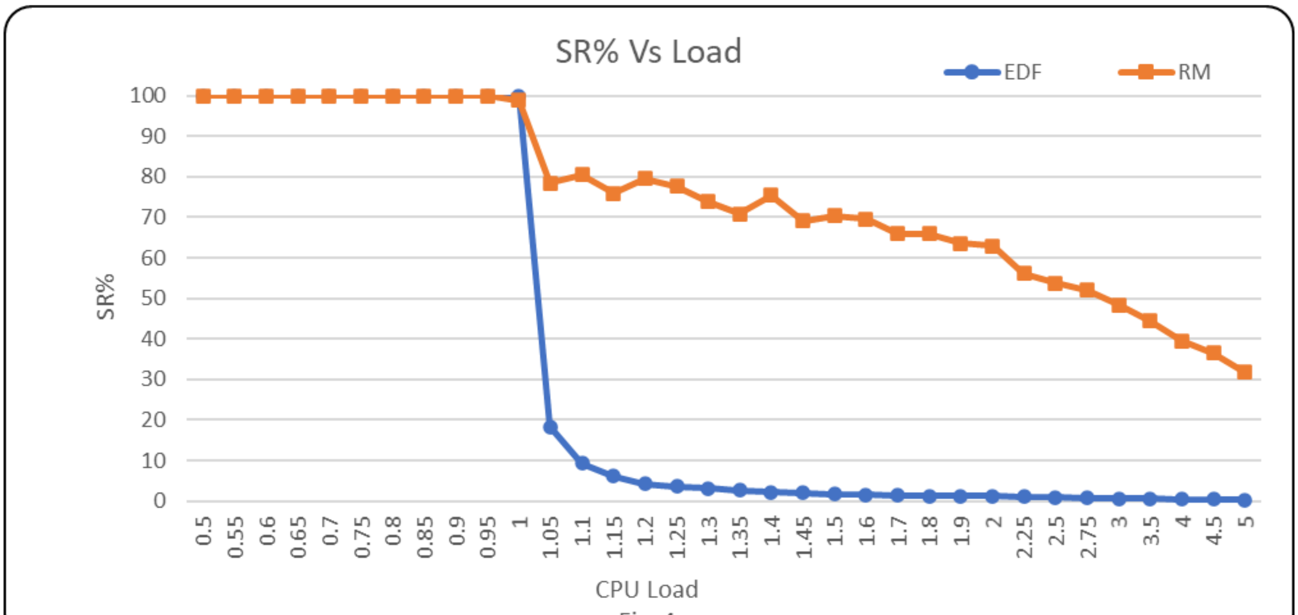

Fig. 4

7. Conclusion

This paper has assessed the EDF and RM scheduling methods for Soft RTOS by considering the periodic task set with a single processor and also believed that the task set is pre-emptive. A comparison of results is given in Table I, which suggests that the EDF, which is dynamic scheduling methods, meets a 100% deadline in the given task set in the underload scenario. In contrast, it is possible to schedule a given task set, but RM failed to schedule it.

In an overload scenario, which results described in Table II, where the EDF scheduling method misses most of the deadline in the given task set, whereas the RM scheduling method still able to meet some the deadline and performs well compared to EDF. Based on the above practical observation, it is advisable to use EDF (dynamic scheduling method) in the underload scenario, whereas RM (static scheduling method) in overload scenario so scheduling method will get more effectiveness.

8. References

[1] R. Belagali, S. Kulkarni, V. Hegde, and G. Mishra, “Implementation and validation of dynamic scheduler based on LST on FreeRTOS,” 2016 Int. Conf. Electr. Electron. Commun. Comput. Optim. Tech. ICEECCOT 2016, pp. 325–330, 2017.

[2] F. Lindh, T. Otnes, and J. Wennerström, “Scheduling algorithms for real-time systems,” Dep. Comput. Eng. Malardalens Univ. Sweden, 2010.

[3] D. Thakor and A. Shah, “D_EDF: An efficient scheduling algorithm for real-time multiprocessor system,” Inf. Commun. Technol. (WICT), 2011 World Congr., pp. 1044–1049, 2011.

[4] J. Teraiya and A. Shah, “Comparative Study of LST and SJF Scheduling Algorithm in Soft Real-Time System with its Implementation and Analysis,” 2018 Int. Conf. Adv. Comput. Commun. Informatics, ICACCI 2018, pp. 706–711, 2018.

[5] W. Li, K. Kavi, and R. Akl, “A non-preemptive scheduling algorithm for soft real-time systems,” Comput. Electr. Eng., vol. 33, no. 1, pp. 12–29, 2007.

[6] G. C. Buttazzo, “Rate Monotonic vs. EDF: Judgment day,” Real-Time Syst., vol. 29, no. 1, pp. 5–26, 2005.

[7] G. Chen and W. Xie, “On a laxity-based real-time scheduling policy for fixed-priority tasks and its non-utilization bound,” 2011 Int. Conf. Inf. Sci. Technol. ICIST 2011, pp. 7–10, 2011.

[8] K.Ramamritham, J.A.Stankovik, and P.F.Shiah, “Efficient scheduling algorithms for real-time multiprocessor systems,” IEEE Transaction on Parallel and Distributed Systems, vol. 1, April 1990.

[9] Shah, A., & Kotecha, K. Scheduling Algorithm for Real-Time Operating Systems Using ACO. In Computational Intelligence and Communication Networks (CICN), 2010 International Conference on (pp. 617-621). IEEE. November 2010.

[10] Mohammadi, A., & Akl, S. G. Technical Report No. 2005-499 Scheduling Algorithms for Real-Time Systems∗. 2005.

[11] http://processdataset.in/

Analysis of Dynamic and Static Scheduling Algorithms in Soft Real-Time System with Its Implementation

Jay Teraiya and Apurva Shah

Abstract The earliest deadline first (EDF) and least slack time first (LST) are dynamic schedulers in real-time system. It chooses the priority of the processes grounded on deadline and slack time correspondingly. The process which has the shortest deadline and smallest slack time will have more priority in EDF and LST. EDF and LST are more appropriate for scheduling of process in soft real-time operating system (RTOS). The rate monotonic (RM) and shortest job first (SJF) are static schedulers in real-time system. It chooses the priority of the processes grounded on its occurrence and time required to execute for given process correspondingly. The process which has the smallest period and smallest time required to execute will be considered as more priority in RM and SJF. In this paper, we have implemented the two dynamic scheduling algorithms (EDF and LST) and two static algorithms (RM and SJF) for the soft RTOS. Algorithms are tested with a periodic task set, and results are collected. We have observed the success ratio (SR) and effective CPU utilization (ECU) for all algorithms in a similar environment. It has been observed that the EDF and LST (dynamic algorithms) perform well in underload condition, but in overload situation, they are not able to perform well, whereas the RM and SJF (static algorithms) are failed to schedule a specific process in the underload scenario as well. They perform well in an overload situation compared with static algorithm. Practical investigations have been led on a huge dataset. Dataset consists of the 7000+ process set, and each process set has one to nine processes, and load varies between 0.5 and 5. It has been tried on 500-time unit to approve the rightness everything being equal.

Keywords Real-time systems · RTOS · Scheduling · LST · SJF · RM · EDF

J. Teraiya (✉)
Marwadi University, Rajkot, Gujarat, India
e-mail: jay.teraiya@gmail.com

A. Shah
The M.S. University Baroda, Vadodara, Gujarat, India

M. Pant et al. (eds.), *Soft Computing: Theories and Applications*,
Advances in Intelligent Systems and Computing 1053,
https://doi.org/10.1007/978-981-15-0751-9_69

1 Introduction

The use of real-time systems for time-critical applications has been increasing over the past years. All tasks in RTOS have associated deadlines, and it needs to be completed within given time. RTOS needs a scheduling algorithm that assigns the processor to tasks by considering the timing constraints as well as supporting all other requirements of scheduling. Depending upon the time criticality, RTOS is divided into hard and soft RTOS. A hard RTOS is one which would fail when the deadline is missed even though the miss is very small. A soft RTOS is one in which a deadline miss is acceptable, but it degrades the overall performance of the system [1].

In RTOS, the suitable scheduling algorithm needs to select grounded on the characteristics of the RTOS and the process type. RTOS categorizes as hard, soft, and firm system. Its process set classifies as a periodic, aperiodic, and sporadic process. Process set can be divided into the preemptive and non-preemptive process. The process has a different characteristic like its execution time, arrival time, deadline, and resource requirements. The scheduler can be divided into two categories, static and dynamic, which depend on the priority they follow. The static algorithm uses a unique priority to each process to throw out the scheduling. Rate monotonic (RM) and deadline monotonic (DM) are an example of static priority algorithms. Dynamic algorithm priority changes during the scheduling process. Earliest deadline first (EDF) and least slack time first (LST) are an example of dynamic priority algorithms. Dynamic algorithms perform well in underload situation and when processes are preemptable. However, the limitation of these algorithms is their performance decreases exponentially if the system becomes slightly overloaded [2, 3].

In the paper, a comparison of dynamic and static algorithms has been compared with a different aspect. Algorithms have been compared with parameters like ECU and SR, and the algorithms are observed in overload and underload scenario. This paper is prepared as: The dynamic and static algorithms have been discussed in Sects. 2 and 3. Section 4 explains background work. Section 5 represents the process set and system consideration. Section 6 defines the practical environment and measuring parameters. Section 7 represents the result and analysis of all four algorithms, and the paper is wrapping up with a brief conclusion in Sect. 8.

2 Dynamic Scheduling Algorithms

Dynamic schedulers make decisions during the runtime of the system. This allows to not only design a more flexible system, but also associate calculation overhead with it. The dynamic schedulers decide what task to execute depending on the importance of the task, called priority. The task priority may change during the runtime [4, 5]. In this section, we have explained two dynamic scheduling algorithms EDF and LST as follows.

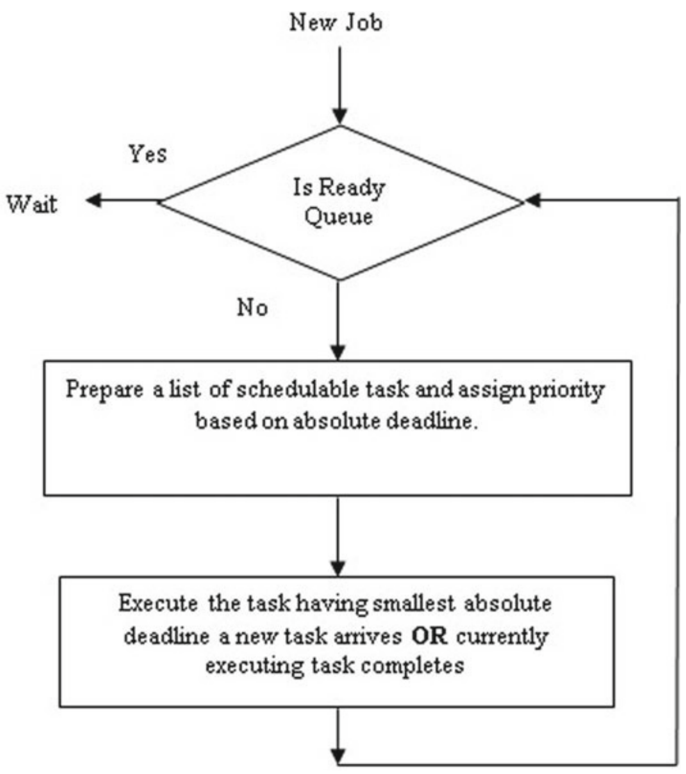
New Job
Is Ready Queue
Yes
Wait
No
Prepare a list of schedulable task and assign priority based on absolute deadline.
Execute the task having smallest absolute deadline a new task arrives OR currently executing task completes

Fig. 1 EDF

2.1 *Earliest Deadline First (EDF)*

The earliest deadline first is a dynamic scheduling algorithm, which gives the highest priority to the task which has a nearest absolute deadline. Priorities of tasks are allocated dynamically and are inversely proportional to the absolute deadlines of the active processes [6]. The algorithm executes when the current process completes or new process arrives. Figure 1 shows a flowchart for the EDF algorithm.

2.2 *Least Slack Time First (LST)*

The LST is a dynamic scheduling algorithm, which gives maximum priority to the process which has the smallest slack time. The slack time (l) can be calculated at time t with the deadline interval d and remaining execution time c [7].

$$l = d - c - t \tag{1}$$

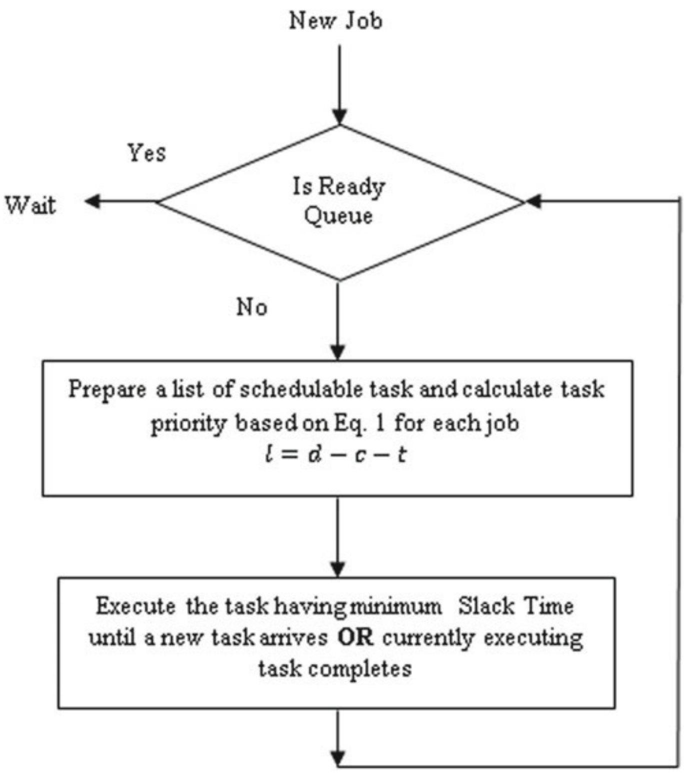
New Job
Is Ready Queue
Yes
Wait
No
Prepare a list of schedulable task and calculate task priority based on Eq. 1 for each job
$l = d - c - t$
Execute the task having minimum Slack Time until a new task arrives OR currently executing task completes

Fig. 2 LST

The algorithm executes when the current process completes or new process arrives. Figure 2 shows a flowchart for the LST algorithm. The new process selected for execution has the smallest slack time.

3 Static Scheduling Algorithms

The static scheduler can calculate the order of execution before runtime as well. The static scheduler also decides the sequence of task based on priority, but the priority value will not change during runtime [8]. In this section, we have explained two static scheduling algorithms RM and SJF as follows.

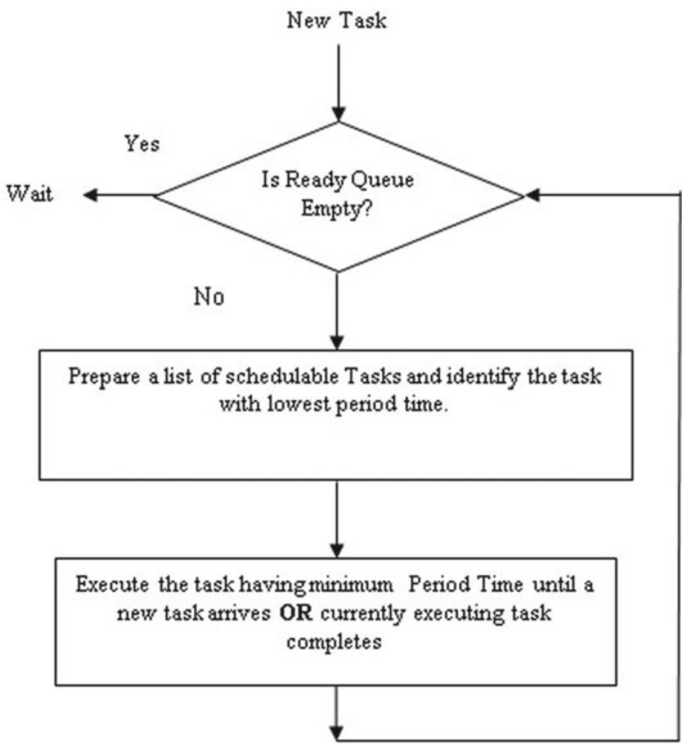
New Task
Yes
Wait
Is Ready Queue Empty?
No
Prepare a list of schedulable Tasks and identify the task with lowest period time.
Execute the task having minimum Period Time until a new task arrives **OR** currently executing task completes

Fig. 3 RM

3.1 The Rate Monotonic (RM)

The rate monotonic is a static scheduling algorithm, which gives maximum priority to the process which has the smallest period or smallest rate [6, 9]. The rate of a process is already known in RTOS and defined as the task occurs again in a given duration. The algorithm executes when the current process completes or new process arrives. Figure 3 shows a flowchart for the RM algorithm.

3.2 The Shortest Job First (SJF)

The shortest job first algorithm is a static scheduling algorithm, which gives maximum priority to the process which has the smallest execution time [9]. The execution time of a process is already known in RTOS and defined as the process that needs CPU time to complete the given task. The algorithm executes when the current process completes or new process arrives. Figure 4 shows a flowchart for the SJF algorithm.

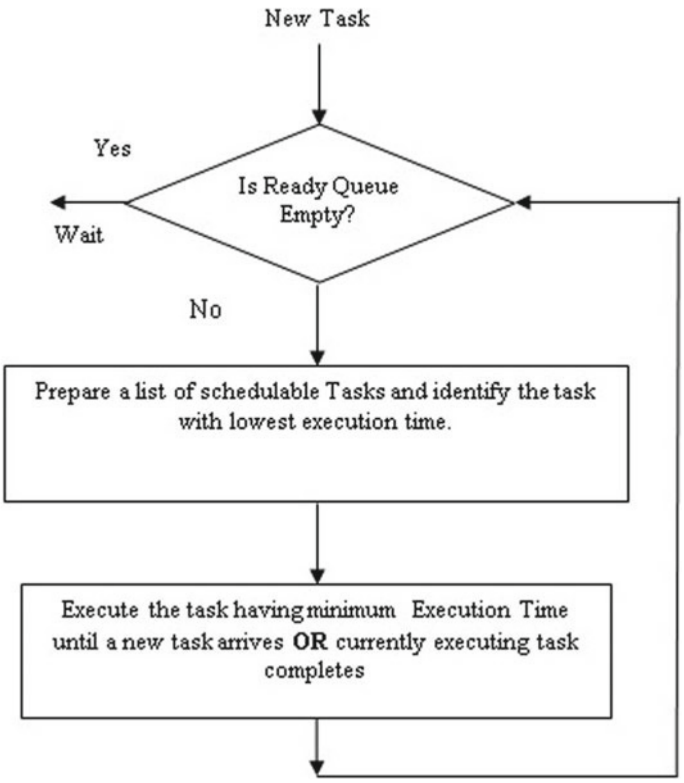
New Task
Is Ready Queue Empty?
Yes
Wait
No
Prepare a list of schedulable Tasks and identify the task with lowest execution time.
Execute the task having minimum Execution Time until a new task arrives OR currently executing task completes

Fig. 4 SJF

4 Background Work

For specific periodic process set, it is possible to identify that can we schedule process set or not. Periodic process has specific parameters like period (T), execution time (C), and deadline (D). The relation $U = C/T$ is named as utilization factor of process set and characterizes processor time required by the process set to complete all process. At a given time, for a set of N processes, utilization factor can be considered by the following equation.

$$U_p = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{2}$$

U_p is named utilization factor and signifies the CPU time required by the periodic process set. If $U_p > 1$, no feasible algorithm exists for the process set and it is overload situation. If $U_p < 1$, then feasible algorithm exists which can schedule the process set [6]. The dynamic scheduling algorithms like EDF and LST are better compared with the static scheduling algorithm in under load situation, and it can schedule the whole process set when $U_p < 1$ [1]. The static scheduling algorithm like RM and SJF decides priority grounded on its rate and required CPU time to complete the task, respectively. The scheduler selects the waiting task with the smallest period and the smallest execution time to execute the next task, respectively [10]. SJF as a single algorithm for RTOS is not observed in any experimental setup. A hybrid approach of SJF and EDF has been followed to decide the group priority of process set [9]. In this paper, it has also been experimented by considering SJF as a single scheduling algorithm with soft real-time system.

5 The Process Set and System Consideration

We considered that process deadline, its rate, and other necessary information are available with the system when the process is released. The process set is preemptive and considered that all the required resources for execution of the process are available. In soft RTOS, each task has a positive value. The system aims to gain maximum benefit. If the process meets its deadline, then the system will get its value. If the process misses its deadline, then the system will gain less value. [11]. Firm RTOS is a kind of real-time system where if the process missed the deadline, then value gain for the given process is zero. But, it is also not considered as a complete failure of the system. This paper includes the implementation of dynamic and static scheduling algorithms which is considered for soft RTOS [12].

6 Practical Environment and Measuring Parameter

Dynamic and static scheduling algorithms have been implemented using C programming language. The algorithm will be executed when a new process is generated or current process completes its execution. Algorithms are tested with the periodic process set for authenticating their performance. Load of the system is calculated based on Eq. (1). If the load is less than one system, it is considered as underload, and if it is more than one system, it is considered as overload scenario. Processes set have been generated with all possible combination. The software module has generated the 7000+ process set, and each process set has one to nine processes. A load of process set varies between 0.5 and 5. It has been tried on 500-time unit to approve the rightness everything being equal. Performance of these (EDF, LST, RM, and SJF) algorithms has been measured based on SR and ECU.

1. **SR**—Success ratio with real-time systems is defined as the ratio of a set of the process which meets their deadline and a total number of process. Success ratio is determined with the following Eq. (3) [13].

$$\text{SR} = \frac{\text{Number of Task successfully scheduled}}{\text{Total Number of Task arrived}} \tag{3}$$

2. **ECU**—Effective CPU utilization is defined as how much CPU time has been utilized for the processes which can meet their deadline. ECU is determined with the following Eq. (4) [13].

$$\text{ECU} = \sum_{i \in R} \frac{V_i}{T} \tag{4}$$

where

- V represents process value and
  - process value = time required to complete the process, if the process meets its deadline.
  - Process value = 0 if the process does not meet the deadline.
- R is a set of process, which is scheduled successfully, i.e., completed within their deadline.
- T is the total time of scheduling.

7 Result and Analysis

In this paper, EDF, LST, RM, and SJF algorithms are implemented and evaluated with SR and ECU parameters, and the results are given in Tables 1 and 2. Table 1 contains the underload scenario, and Table 2 includes the result of an overload situation where in underload it is $U_p \leq 1$ and in overload it is $U_p > 1$. Observation with these results indicates that ECU values persist nearly the same for dynamic and static algorithms, but SR values are not 100% with the static scheduling algorithms. When $U_p \leq 1$, it indicates that scheduling of given task set is possible, but static scheduling algorithms are failing to schedule all process, whereas dynamic scheduling algorithm can schedule this process set. Dynamic scheduling algorithms give optimum result in underload scenario, and it is advisable to use the dynamic schedulers with underload condition. Table 2 contains the results of overload situation,

Table 1 Dynamic and static algorithms performance in underload

Load	ECU%				SR%			
	Dynamic		Static		Dynamic		Static	
	EDF	LST	RM	SJF	EDF	LST	RM	SJF
0.50	49.49	49.49	49.49	49.49	100.00	100.00	100.00	100.00
0.55	54.66	54.40	54.40	54.31	100.00	100.00	100.00	100.00
0.60	59.39	59.39	59.39	59.39	100.00	100.00	100.00	100.00
0.65	64.35	64.35	64.35	64.35	100.00	100.00	100.00	100.00
0.70	69.35	69.35	69.35	69.35	100.00	100.00	100.00	100.00
0.75	74.31	74.31	74.31	74.31	100.00	100.00	100.00	100.00
0.80	79.22	79.22	79.22	79.22	100.00	100.00	100.00	100.00
0.85	84.16	84.16	84.16	84.15	100.00	100.00	100.00	99.99
0.90	89.16	89.16	89.15	89.00	100.00	100.00	99.99	99.84
0.95	94.17	94.17	94.08	93.89	100.00	99.99	99.93	99.78
1.00	99.10	99.10	97.78	96.74	100.00	100.00	98.92	98.74

and the observation indicates that dynamic algorithms performance reduces quickly, whereas static algorithms like RM and SJF are still able to meet few of their deadlines for given process set. This observation can conclude that in underload scenario, EDF and LST give optimal results, whereas in overload scenario, RM and SJF performed well (Figs. 5 and 6).

8 Conclusion

The dynamic and static algorithms are evaluated in this paper for soft RTOS and considering it for a single processor and preemptive process sets. It is also believed that process set is periodic. All four algorithms are evaluated in a similar environment, and the results have been observed and equated. EDF and LST are dynamic algorithms, and they do well in underload scenario and schedule all process in a given process set. LST and SJF are static algorithms, and they do well in overload scenario and try to schedule maximum process in given process set. The ideal algorithm can be designed which uses the features of dynamic and static algorithm, and it performs well in underload as well as overload scenario.

Table 2 Dynamic and static algorithms performance in overload

Load	ECU%				SR%			
	Dynamic		Static		Dynamic		Static	
	EDF	LST	RM	SJF	EDF	LST	RM	SJF
1.05	17.45	16.09	70.85	56.63	18.27	15.84	78.49	73.49
1.10	9.21	8.33	75.82	63.60	9.31	7.90	80.49	75.98
1.15	6.29	5.58	73.20	62.66	6.19	5.06	75.88	73.66
1.20	4.62	4.21	83.50	70.08	4.22	3.67	79.47	73.06
1.25	4.06	3.56	79.05	73.20	3.67	3.06	77.58	77.47
1.30	3.63	3.09	75.66	72.24	3.19	2.53	73.81	75.34
1.35	3.12	2.63	74.65	70.99	2.65	2.09	70.77	71.55
1.40	2.66	2.20	83.55	76.57	2.24	1.71	75.47	73.80
1.45	2.50	2.01	79.75	74.45	2.00	1.52	69.03	68.76
1.50	2.21	1.83	85.27	80.07	1.71	1.33	70.33	69.96
1.60	2.17	1.77	85.61	77.26	1.61	1.29	69.52	67.20
1.70	2.03	1.58	86.26	79.16	1.42	1.07	65.99	64.60
1.80	1.93	1.45	86.12	77.28	1.30	0.95	65.98	63.04
1.90	1.90	1.31	85.83	77.53	1.29	0.85	63.51	62.21
2.00	1.84	1.19	85.78	78.10	1.20	0.76	62.88	61.00
2.25	1.76	1.13	84.27	76.95	1.04	0.65	56.16	55.91
2.50	1.55	0.98	87.06	74.97	0.89	0.54	53.82	49.92
2.75	1.46	0.91	89.21	74.42	0.78	0.47	52.07	46.83
3.00	1.32	0.86	94.46	77.23	0.63	0.40	48.36	41.67
3.50	1.27	0.75	93.48	73.37	0.57	0.33	44.50	36.76
4.00	1.11	0.73	95.04	79.57	0.43	0.27	39.52	34.09
4.50	1.08	0.71	96.77	71.58	0.38	0.24	36.45	27.74
5.00	0.97	0.66	98.13	78.22	0.31	0.20	31.72	25.71

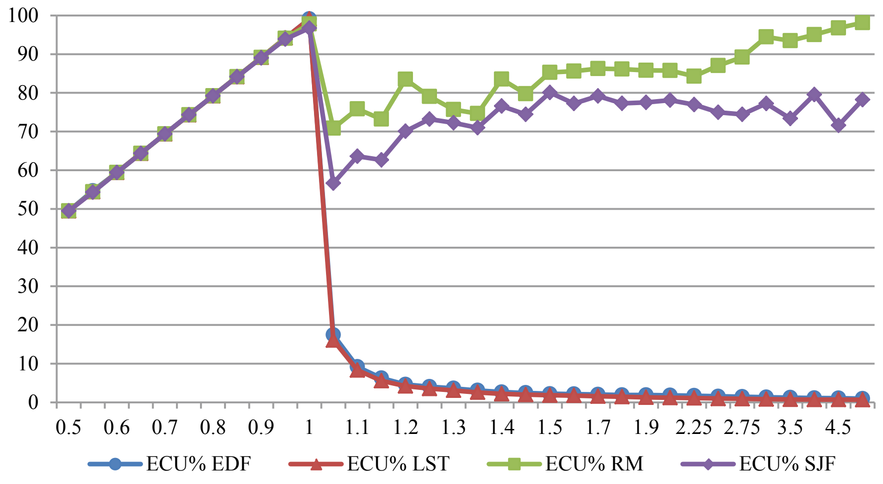

Fig. 5 ECU% versus load

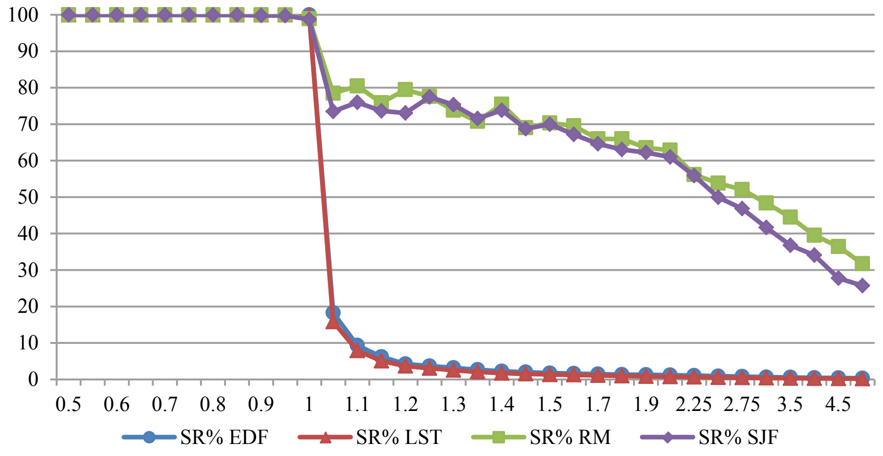

Fig. 6 SR% versus load

References

1. Belagali, R., Kulkarni, S., Hegde, V., Mishra, G.: Implementation and validation of dynamic scheduler based on LST on FreeRTOS. In: Electrical, Electronics, Communication, Computer and Optimization Techniques (ICEECCOT), Mysore, India, Dec 2016, pp. 325–330
2. Mohammadi, A., Akl, S.G.: Scheduling Algorithms for Real-Time Systems. School of Computing, Queen's University, Kingston, Ontario (2005)
3. Thakor, D., Shah, A.: D_EDF: an efficient scheduling algorithm for real-time multiprocessor system. In: Information and Communication Technologies (WICT), Mumbai, India, Dec 2011, pp. 1044–1049
4. Harkut, D.G.: Comparison of different task scheduling algorithms in RTOS: a survey. Int. J. Adv. Res. Comput. Sci. Softw. Eng. **4**(7), 1236–1240 (2014)
5. Koren, G., Shasha, D.: D_over: an optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. SIAM J. Comput. **24**(2), 318–339 (1995)
6. Buttazzo, G.C.: Rate monotonic vs. EDF: judgment day. Real-Time Syst. **29**(1), 5–26 (2005)
7. Patel, M., Oza, B.: An improved LLF_DM scheduling algorithm for periodic tasks by reducing context switches. Int. J. Adv. Eng. Res. **2**, 248–254 (2015)
8. Ramamritham, K., Stankovic, J.A., Shiah, P.F.: Efficient scheduling algorithms for real-time multiprocessor systems. IEEE Trans. Parallel Distrib. Syst. **1**(2), 184–194 (1990)
9. Li, W., Kavi, K., Akl, R.: A non-preemptive scheduling algorithm for soft real-time systems. Comput. Electr. Eng. **33**(1), 12–29 (2007)
10. Chen, G., Xie, W.: On a laxity-based real-time scheduling policy for fixed-priority tasks and its non-utilization bound. In: Information Science and Technology (ICIST), Tebessa, Algeria, pp. 7–10 (2011)
11. Locke, C.D.: Best-effort decision making for real-time scheduling. Ph.D. Thesis, Computer Science Department, CMU (1986)
12. Shah, A.: Adaptive scheduling algorithm for real-time distributed systems. In: Biologically-Inspired Techniques for Knowledge Discovery and Data Mining, pp. 236–248 (2014)
13. Shah, A., Kotecha, K.: Scheduling algorithm for real-time operating systems using ACO. In: Computational Intelligence and Communication Networks (CICN), Bhopal, India, Nov 2010, pp. 617–621

Hybrid Scheduler (S_LST) for Soft Real-Time System based on Static and Dynamic Algorithm

Jay Teraiya, Apurva Shah

Abstract*: In the Soft Real-Time System scheduling process with the processor is a critical task. The system schedules the processes on a processor in a time interval, and hence the processes get chance to executes on the processor. Priority-driven scheduling algorithms are sub-categorized into mainly two categories called Static Priority and Dynamic Priority Scheduler. Critical Analysis of more static and dynamic priority scheduling algorithms have been discussed in this paper. This paper has covered the static priority algorithms like Rate Monotonic (RM) and Shortest Job First (SJF) and the dynamic priority algorithms like Earliest Deadline First (EDF) and Least Slack Time First (LST). These all algorithms have been analyzed with preemptive process set and this paper has considered all the process set are periodic. This paper has also proposed a hybrid approach for efficient scheduling. In a critical analysis, it has been observed that while scheduling in underload situation dynamic priority algorithms perform well and even EDF also make sure that all process will meet their deadline. However, in an overload situation, the performance of dynamic priority algorithms reduce quickly, and most of the task will miss its deadline, whereas static priority scheduling algorithms miss a few deadlines, even it is possible to schedule all processes in underload situation, whereas in an overload situation, the static algorithms perform well compared to the dynamic scheduler. This paper is proposing one Hybrid algorithm call S_LST which uses the concept of LST and SJF scheduling algorithm. This algorithm has been applied to the periodic task set, and observations are registered. We have observed the Success Ratio (SR) & Effective CPU Utilization (ECU) and compared all algorithms in the same conditions. It is noted that instead of using LST and SJF as an independent algorithm, Hybrid algorithm S_LST performs well in underload and overload scenario. Practical investigations have been led on a huge dataset. Data Set consists of the 7000+ process set, and each process set has one to nine processes and load varies between 0.5 to 5. It has been tried on 500-time unit to approve the rightness everything being equal.

Keywords: Soft Real-Time System, RTOS, RM, SJF, LST, EDF, S_LST

I. INTRODUCTION

Real-Time Systems has to complete its work and deliver its services on a timely basis. It makes sure that its task will be completed before its deadline. Example of a Real-Time system is vehicle control, flight control, healthcare equipment, and many more. Typical PC run nonreal-time applications such as a browser, editor, different user applications. When the real-time system works correctly, and well, they make us forget their existence [1].

The real-time system is sub-categorized into mainly two types: hard and soft. There are many definitions of hard and soft real-time systems. Real-Time system is considered as Hard if the process fails to meet its deadline, then it will be a fatal fault. In Hard Real-Time, if the process missed its deadline, then result produced by the job after the deadline may have disastrous consequences. A few examples of Hard Real-Time Systems are Metro Train and its signal system, Missile technology, Flight control system. The real-time system is considered as Soft if the late completion of the process is undesirable. However, a few misses of soft deadlines do no serious harm; only the system's performance becomes poor. A few examples of Soft Real-Time systems include ATM System, Mobile application and telephone switches [7].

The real-time system has three kinds of task model call Periodic, Aperiodic and Sporadic tasks. In the periodic task, each task generated at regular time intervals. The Real-Time system is invariably required to respond to external events and to respond; it executes aperiodic or sporadic tasks whose release times are not known to the system in advance. We call the task is aperiodic if the process in it have soft deadlines. Each unit of work is scheduled and executed by the system as a process. Each process has a different characteristic like release time, deadline, period and execution time. The release time of a process is the instant of time at which the job become available for execution. The process can be scheduled and executed at any time after its release. The deadline for a process is the instant of time by which its execution needs to be completed. The deadline for a process sometimes called absolute deadline, which is equal to its release time plus its relative deadline. The execution time of any process is considered as the unit amount of time required for the process to execute it on the processor. If the process is periodic, then the period of the process indicates the occurrence interval of the given process.

In RTOS, selecting the scheduling algorithm is a critical task, and it will be decided based on the characteristics of the RTOS and the process type [2]. The scheduler can be divided into two categories, static and dynamic, which depends on the priority they follow in selecting the process for execution.

Revised Manuscript Received on December 15, 2019.

* Correspondence Author
Jay Teraiya*, Department of Computer Engineering, Marwadi University, Rajkot, India. Email: jay.teraiya@gamil.com
Apurva Shah, Department of Computer Science and Engineering The Maharaja Sayajirao University of Baroda, Baroda, India. Email: apurva.shah-cse@msubaroda.ac.in

The static algorithm uses a unique priority to each process throughout the scheduling. Rate Monotonic (RM) and Deadline Monotonic (DM) are an example of static priority algorithms. Dynamic algorithm priority changes during the scheduling process. Earliest Deadline First (EDF) and Least Slack Time First (LST) are an example of dynamic priority algorithms [10][11].

In this paper, we have compared all dominant dynamic and static scheduling algorithms and did their critical analysis. All algorithm has been compared based on Success Ratio (SR) and Effective CPU Utilization (ECU) parameters. This paper also proposed an effective scheduling algorithm call S_LST, which is using characteristics of LST and SJF. The new algorithm also compared with the rest of all algorithms based on SR and ECU parameters. This paper explains the Static and Dynamic Scheduling algorithm in section II. Their critical analysis based on SR and ECU has been described in section III, and a new efficient algorithm call S_LST has been proposed in section IV, and performance of a new algorithm has been compared and discussed in section V, and paper is ended with a brief conclusion in section VI.

II. THE STATIC AND DYNAMIC SCHEDULING ALGORITHMS

Priority-driven scheduling algorithms are online schedulers that schedule the process according to some priority. It does not pre-decide the process; instead of that, it assigns priorities to process when it is ready to execute. The scheduling algorithm will be executed whenever a new process is released, or currently, running process completes its execution. Priority-driven schedulers categorize based on how priority assigned to each process. Priority-driven algorithms are classified in to two categories: Static Priority and Dynamic Priority. A Static Priority algorithm assigns the same priority to all the periodic processes, and it will remain fixed relative to other processes. Whereas dynamic-priority algorithm changes the priority of the process based on the new process arrives or currently running process completes [12][22].

A. Static Scheduling Algorithms

The Rate Monotonic (RM) and the Shortest Job First (SJF) are well known static priority algorithms. The RM assigns the priority to the process based on their period (the frequency of the task when it occurs). The Rate of the process is already known in RTOS for the periodic task. The rate of a process is the inverse of its period, so higher the rate, the priority of the process will be high [6][13][14]. The Shortest Job First (SJF) assigns the priority to the process based on their required execution time. The required execution time of the process is also known in RTOS and process with the shortest execution time will have the highest priority in SJF [13]. By looking at the approach of both algorithms, its ultimate aim is to gain maximum profit or try to meet the maximum deadline of the given processes.

B. Dynamic Scheduling Algorithms

The Earliest Deadline First (EDF) and the Least Slack Time First (LST) are well known dynamic priority algorithms. The EDF assigns the priority to the process based on the absolute deadline. The absolute deadline for each process is already known in RTOS, and the process which has the smallest absolute deadline will consider as highest priority process [8][14]. The LST is another well-known dynamic priority algorithm, and it assigns priority based on the slack time of the given process. The slack value of the process is equal to absolute deadline minus given time t minus remaining execution time x (slack=d-t-x). The algorithm checks the slacks of all the ready process each time a new process is released, or the existing process completes. The process with the smallest slack value will have the highest priority [9][15][16][17]. By looking at the approach of both algorithms, its ultimate aim is to meet the deadline of the given process.

For any set of periodic processes, we can verify its stimulability is possible or not using its occurrence period(T), its execution time(C), and its deadline(D). This ratio U_p is called the utilization factor of the task set as shown in equation 1.

$$U_p = \sum_{i=1}^{n} \frac{C_i}{T_i} \qquad (1)$$

U_p is called the total processor utilization factor and represents the fraction of processor time used by the periodic task set. If U_p>1 no feasible schedule exists for the task set with an algorithm, and it is overload condition.

III. CRITICAL ANALYSIS OF STATIC AND DYNAMIC SCHEDULING ALGORITHM

A. System Consideration and Task Model

In Soft Real Time System, system is already aware with task deadline, its period and the other required data to compute the required time by the task when task is dispatch. The process set is considered pre-emptive. This paper has believed that the system is not having a resource clash problem. Each task in soft real-time systems has a positive value and ultimate goal is to gain maximum value. If a process succeeds, then the system considers its value. If a process fails, then the system gets less benefit from it [18] [19]. In this paper, we have implemented Dynamic and Static scheduling algorithms that apply to the soft real-time system. The value of the task has been considered the same as its computation time required [20].

B. Experimental Environment and Evaluating Parameters

1) Success Ratio (SR):

Success Ratio with real-time systems defined as the ratio of a set of the process which meets their deadline and a total number of process. Success Ration determined with the following equation 2 [21].

$$SR = \frac{\textit{Number of Task successfully scheduled}}{\textit{Total Number of Task arrived}} \qquad (2)$$

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

2) Effective CPU Utilization (ECU):

Effective CPU Utilization defined as how much CPU time has been utilizing for the processes which can meet their deadline. ECU determined with the following equation 3 [21].

$$ECU = \sum_{i \in R} \frac{V_i}{T} \qquad (3)$$

Where,

• V represents process value and,
Process Value = time required to complete the process if the process meets its deadline.
Process Value = 0 if the process does not meet the deadline.
• R is a set of processes, which are scheduled successfully, i.e., completed within their deadline.
• T is the total time of scheduling.

C. Analysis and Observation

RM, SJF, EDF, and LST algorithms are implemented and evaluated with SR and ECU parameters (explained in section 3), and results have been observed. Observation with these results indicates that ECU values persist nearly the same for Dynamic and Static algorithms, but SR values are not 100% with the Static scheduling algorithms. When $U\_p \leq 1$, it indicates that scheduling of a given task set is possible, but Static scheduling algorithms are failing to schedule all processes, whereas Dynamic scheduling algorithm can schedule this process set. Dynamic scheduling algorithms give optimum results in underload scenario, and it is advisable to use the Dynamic schedulers with underload conditions. In overload situation when $U\_p > 1$, observation indicates that Dynamic algorithms performance reduce quickly whereas Static algorithms like RM and SJF are still able to meet a few of their deadline for a given process set. This observation can conclude that in underload EDF and LST give optimal results whereas in overload RM and SJF performed well. Fig. 1 and Fig. 2 provides a graphical representation of results.
The Static and Dynamic algorithms are evaluated here for Soft – RTOS and considering it for a single processor, and pre-emptive process sets and all process set is periodic. All algorithms are evaluated in a similar environment and results have been observed and equated. EDF and LST are dynamic algorithms, and they do well in underload scenario and schedule all processes in a given process set. LST and SJF are static algorithms, and they do well in an overload scenario and try to schedule the maximum process in a given process set. The ideal algorithm can be designed, which uses the features of Dynamic and Static algorithm, and it performs well in underload as well as overload scenario [3][4][5].

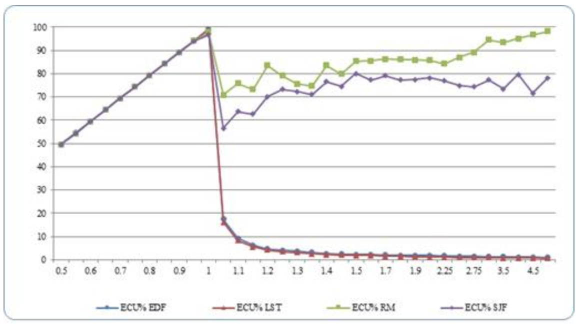

Fig. 1 Load Vs. ECU%

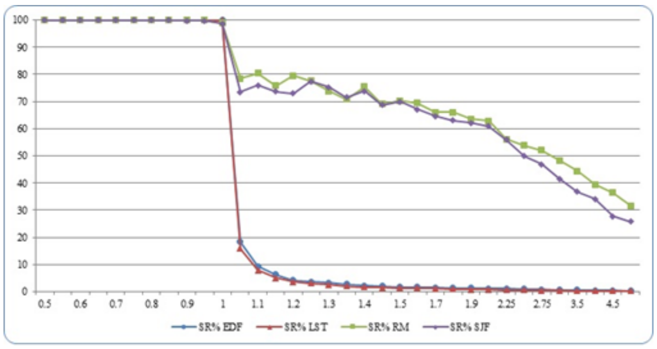

Fig. 2 Load Vs. SR%

IV. THE HYBRID APPROACH FOR EFFICIENT SCHEDULING – S_LST ALGORITHM

S_LST algorithm uses the characteristics of LST and SJF. In underload, situation task priority will be given based on slack time, and in an overload situation, task priority will be assigned based on the shortest execution time. We are considering that the execution time of the task, its arrival time, its period and total CPU load is available with Soft Real-Time System. The scheduling algorithm executes when a currently running task completes or a new task arrives. The algorithm has been described as follows.

_

S_LST Algorithm for Scheduling

_

```
Input: Process Set
Output: MIProcess
1:  if (Underload Scenario)
2:          for each process in process set
3:              Calculate Slack time for each Process in Process Set
4:              Select MIProcess with lowest slack time
5:          end for
6:  else
7:          for each process in process set
8:              Calculate Shortest Execution Time for each process
9:              Select MIProcess with lowest Execution time
10:     end for
11: end if
12: return MIProcess
```

_

As shown in Algorithm, when scheduling algorithm invokes; first it observed the CPU load, based on the current process set and available processes are
ready for scheduling.

If $U_p < 1$ it will assign the task priority based on slack time (Dynamic scheduling algorithm) and if $U_p > 1$ it will assign the task priority based on shortest execution time (Static Scheduling algorithm). The static scheduler aim is to gain maximum profit from the given process set. So, in overload situation where dynamic scheduler performs poorly, SJF algorithm gets more processes meets their deadline.

V. S_LST ALGORITHM RESULTS AND DISCUSSION

Table 1 represents the results of LST, SJF and the S_LST algorithm on the simulator. To evaluate S_LST, we are using a similar environment which we have used to evaluate all Static and Dynamic priority algorithm as per section 3. Table 1 first eleven rows represent the scenario where task set contains 1 to 9 task and Load is less than 1 or equal to 1 ($U_p \leq 1$). Results show that S_LST performs equally well in underload scenarios like LST algorithm in terms of ECU and SR parameter. S_LST uses slack time value of task to assign dynamic priority in underload situation.

Table 1 rest of rows represents the scenario where process set contains 1 to 9 process and Load is greater than 1 ($U_p > 1$). Results show a waste difference in ECU and SR values compare to a simple LST algorithm. When Load is greater than 1, it means that task set is not schedulable, and most of the process misses their deadline with LST algorithm. Table 1 observations reflect that in slightly overload situations LST performance degrades very poorly, whereas SJF able to meet the deadline for few of their process sets. It means in overload situation, SJF gives better performance than LST. That is why S_LST uses static priority in an overload situation. Fig. 3 and Fig. 4 provides a graphical representation of Table 1.

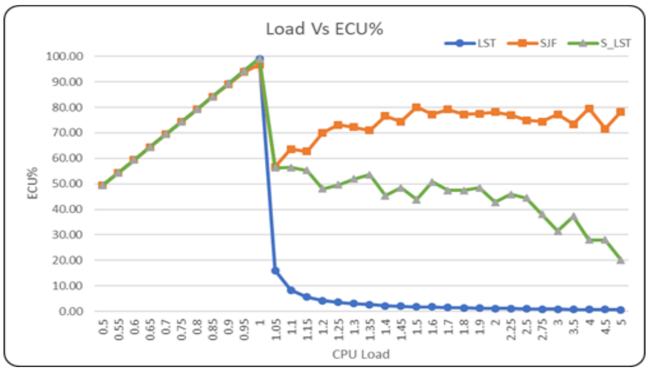

Fig. 3 Load Vs. ECU%

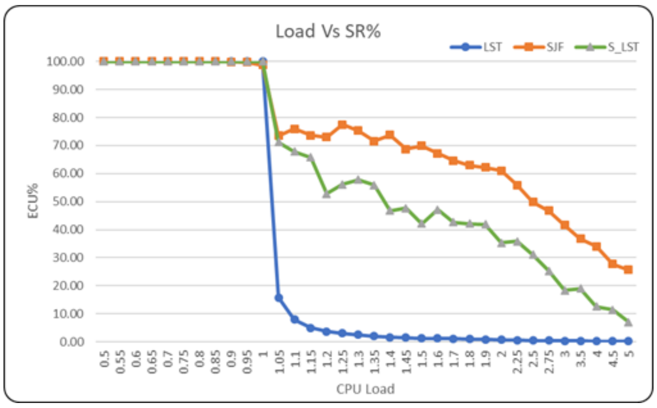

Fig. 2 Load Vs. SR%

Table- I: Comparison of LST, SJF and S_LST

	ECU			SR		
Load	**LST**	**SJF**	**S_LST**	**LST**	**SJF**	**S_LST**
0.5	49.49	49.49	49.49	100.00	100.00	100.00
0.55	54.40	54.31	54.40	100.00	100.00	100.00
0.6	59.39	59.39	59.39	100.00	100.00	100.00
0.65	64.35	64.35	64.35	100.00	100.00	100.00
0.7	69.35	69.35	69.35	100.00	100.00	100.00
0.75	74.31	74.31	74.31	100.00	100.00	100.00
0.8	79.22	79.22	79.22	100.00	100.00	100.00
0.85	84.16	84.15	84.16	100.00	99.99	100.00
0.9	89.16	89.00	89.16	100.00	99.84	100.00
0.95	94.17	93.89	94.17	99.99	99.78	99.99
1	99.10	96.74	99.10	100.00	98.74	100.00
1.05	16.09	56.63	56.63	15.84	73.49	73.49
1.1	8.33	63.60	63.60	7.90	75.98	75.98
1.15	5.58	62.66	62.66	5.06	73.66	73.66
1.2	4.21	70.08	70.08	3.67	73.06	73.06
1.25	3.56	73.20	73.20	3.06	77.47	77.47
1.3	3.09	72.24	72.24	2.53	75.34	75.34
1.35	2.63	70.99	70.99	2.09	71.55	71.55
1.4	2.20	76.57	76.57	1.71	73.80	73.80
1.45	2.01	74.45	74.45	1.52	68.76	68.76
1.5	1.83	80.07	80.07	1.33	69.96	69.96
1.6	1.77	77.26	77.26	1.29	67.20	67.20
1.7	1.58	79.16	79.16	1.07	64.60	64.60
1.8	1.45	77.28	77.28	0.95	63.04	63.04
1.9	1.31	77.53	77.53	0.85	62.21	62.21
2	1.19	78.10	78.10	0.76	61.00	61.00
2.25	1.13	76.95	76.95	0.65	55.91	55.91
2.5	0.98	74.97	74.97	0.54	49.92	49.92
2.75	0.91	74.42	74.42	0.47	46.83	46.83
3	0.86	77.23	77.23	0.40	41.67	41.67
3.5	0.75	73.37	73.37	0.33	36.76	36.76
4	0.73	79.57	79.57	0.27	34.09	34.09
4.5	0.71	71.58	71.58	0.24	27.74	27.74
5	0.66	78.22	78.22	0.20	25.71	25.71

VI. CONCLUSION

The Static Algorithms (RM and SJF) and Dynamic Algorithms (EDF and LST) are implemented for scheduling of soft real-time system with a single processor and pre-emptive task sets and done a critical analysis of these algorithms with ECU and SR parameter in this paper. These algorithms are simulated with periodic task sets; results are obtained and compared. Observation says that dynamic algorithms perform well in underload situations and gives a guarantee to meet all the deadlines of a given process set. In overload (Load is > 1) situation, dynamic algorithms performance degrades very poorly. So, in underload, dynamic algorithms are advisable but not with an overload situation.

Static algorithms miss few process deadlines even in underload situations. It has been observed that with the specific process set, even it is possible that all processes can meet their deadline, but static algorithms are failed to schedule it. So, in underload, static schedulers are not advisable, but in overload, they perform well compared to dynamic algorithms. Based on this observation we have proposed a hybrid approach for efficient scheduling in Soft Real-Time system call S_LST. S_LST algorithm assigns the static priority in overload situations will perform better in all situations compare to a single approach. Developing a scheduling algorithm using swarm (ACO) has been done for the Soft Real-Time system [21]. There is still multiple research possibility where we can use swarm intelligence techniques like Gravitational Search Algorithm (GSA) or Particle Swarm Optimization (PSO) and can design an efficient scheduling algorithm which can perform well in underload and overload situation.

REFERENCES

1. El Ghor, H., Hage, J., Hamadeh, N., & Chehade, R. H. (2018). Energy-Efficient Real-Time Scheduling Algorithm for Fault-Tolerant Autonomous Systems. Scalable Computing: Practice and Experience, 19(4), 387-400.
2. A. Magdich, Y. Hadj Kacem, M. Kerboeuf, A. Mahfoudhi, and M. Abid, "A design pattern-based approach for automatic choice of semi-partitioned and global scheduling algorithms," Inf. Softw. Technol., vol. 97, no. November 2017, pp. 83–98, 2018.
3. J. Teraiya and A. Shah, "Comparative Study of LST and SJF Scheduling Algorithm in Soft Real-Time System with its Implementation and Analysis," 2018 Int. Conf. Adv. Comput. Commun. Informatics, ICACCI 2018, pp. 706–711, 2018.
4. J. Teraiya, A. Shah, and E. Foundation, "Analysis of Dynamic and Static Scheduling Algorithms in Soft Real-Time System with its Implementation," Soft-Computing: Theories and Applications (SoCTA - 2018) Jalandhar, India 21-23 December 2018.
5. Konar, D., Bhattacharyya, S., Sharma, K., Sharma, S., & Pradhan, S. R. (2017). An improved Hybrid Quantum-Inspired Genetic Algorithm (HQIGA) for scheduling of real-time task in multiprocessor system. Applied Soft Computing, 53, 296-307.
6. Feld, T., Biondi, A., Davis, R. I., Buttazzo, G., & Slomka, F. (2018). A survey of schedulability analysis techniques for rate-dependent tasks. Journal of Systems and Software, 138, 100-107.
7. Laalaoui, Y., & Bouguila, N. (2014). Pre-run-time scheduling in real-time systems: Current researches and artificial intelligence perspectives. Expert Systems with Applications, 41(5), 2196-2210.
8. Yang, K., & Anderson, J. H. (2015, August). On the soft real-time optimality of global EDF on multiprocessors: From identical to uniform heterogeneous. In 2015 IEEE 21st International Conference on Embedded and Real-Time Computing Systems and Applications (pp. 1-10). IEEE.
9. Benhai, Z., Yuan, Y., Hongyan, M., Dapeng, Y., & Libo, X. (2016, May). Research on optimal ELSF real-time scheduling algorithm for CPS. In 2016 Chinese Control and Decision Conference (CCDC) (pp. 6867-6871). IEEE.
10. A. Mohammadi and S. G. Akl, "Scheduling Algorithms for Real-Time Systems", in School of Computing, Queen's University, Kingston, Ontario, 2005.
11. Thakor, D., & Shah, A. (2011, December). "D_EDF: An efficient scheduling algorithm for real-time multiprocessor system", in Information and Communication Technologies (WICT), Mumbai, India, pp. 1044-1049, 2011.
12. D. G. Harkut, "Comparison of Different Task Scheduling Algorithms in RTOS : A Survey," vol. 4, no. 7, pp. 1236–1240, 2014
13. Li, W., Kavi, K., &Akl, R. "A non-preemptive scheduling algorithm for soft real-time systems", in Computers & Electrical Engineering, Vol. 33(1), pp. 12-29, 2007.
14. Buttazzo, G. C. "Rate monotonic vs. EDF: judgment day", in Real-Time Systems, Vol. 29(1), pp. 5-26, 2005.
15. M. Patel and B. Oza, "An Improved LLF_ DM Scheduling Algorithm for Periodic Tasks by Reducing Context Switches," in International Journal of Advance Engineering and Research, vol. 2, pp. 248–254, 2015.
16. Belagali, R., Kulkarni, S., Hegde, V., & Mishra, G. "Implementation and validation of dynamic scheduler based on LST on Free RTOS", in Electrical, Electronics, Communication, Computer and Optimization Techniques (ICEECCOT), Mysore, India, pp. 325-330, 2016, December.
17. Chen, G., & Xie, W. "On a laxity-based real-time scheduling policy for fixed-priority tasks and its non-utilization bound", in Information Science and Technology (ICIST), 2011,Tebessa, Algeria, pp. 7-10, 2011.
18. Locke, C. D. "Best-effort decision making for real-time scheduling" (Ph. D Thesis), Computer Science Department, CMU, 1986.
19. Koren, G., & Shasha, D. "D_over: An Optimal On-Line Scheduling Algorithm for Overloaded Uniprocessor Real-Time Systems" in SIAM Journal on Computing, Vol. 24(2), pp. 318-339, 1995.
20. A Shah, "Adaptive scheduling algorithm for real-time distributed systems", in Biologically-Inspired Techniques for Knowledge Discovery and Data Mining,pp. 236-248, 2014.
21. J. Teraiya, A. Shah, & K. Kotecha, "ACO Based Scheduling Method for Soft RTOS with Simulation and Mathematical Proofs" in International Journal of Innovative Technology and Exploring Engineering, Vol. 8, Issue. 12 pp. 4736-4740, 2019.
22. D. G. Harkut, "Comparison of Different Task Scheduling Algorithms in RTOS : A Survey," vol. 4, no. 7, pp. 1236–1240, 2014.

AUTHORS PROFILE

Jay Teraiya, has completed Bachelor of Engineering from GCET – Vallabh Vidyanagar under S. P. University. He has also completed his M. S. in Software Engineering from BITS Pillani. He is pursuing in Ph. D from the M. S. University of Baroda under the guidance of Dr. Apurva Shah.

Dr. Apurva Shah, Associate Professor and Head of Department (Computer Science and Engineering) in Faculty of Technology, the M. S. University of Baroda Gujarat. He is also director of Computer Center in the University. His area of interest are Real Time System, Artificial intelligence and distributed computing. He has completed his Ph. D. from S. P. University Vallabh Vidyanagar.

ACO Based Scheduling Method for Soft RTOS with Simulation and Mathematical Proofs

Jay Teraiya, Apurva Shah, Ketan Kotecha

Abstract*: *The Ant Colony Optimization (ACO) algorithm is a mathematical model enlivened by the system searching conduct of ants. By taking a gander at the qualities of ACO, it is most suitable for scheduling of tasks in soft real-time systems. In this paper, the ACO based scheduling method for the soft real-time operating system (RTOS) has been profound with mathematical and practical proof. In Mathematical proof, three different Propositions and two Theorems have been given, which prove the correctness of the proposed algorithm. Practical experiments also support mathematical proofs. During the investigation, observations are gathered with different periodic task set. Algorithms have been observed regarding Success Ratio (SR) and Effective CPU utilization (ECU). ACO based scheduling algorithm has been compared with the Earliest Deadline First (EDF) algorithm with parameter SR and ECU. The EDF is dynamic scheduling algorithm and it is most suitable in RTOS when task set is preemptable. It is noted that the new algorithm is equally efficient during under loaded conditions when CPU load is less than one. ACO based scheduling algorithm performs superior during the overloaded conditions when CPU load is more than one where as EDF algorithm performance degraded in overload condition. Empirical study has been executed with a hefty Dataset consist of more than 7500 task set, and a set contains different one to nine processes where CPU load is dynamic for each process set and differ from 0.5 to 5. Algorithms have been executed on five-hundred-time unit for each process set to authenticate the accuracy of both algorithms.

***Keywords*: ACO, EDF, ACO, Real-Time Systems, RTOS**

I. INTRODUCTION

Real-time system is the systems in which the accuracy of the system not only defined by the logical accuracy but also with the time it takes to produce the result. Real-Time systems have decisive, unchanging time restrictions, i.e., the task must be ended within the specified duration; otherwise, the system fails. One can find the existence of two types of real-time systems: Hard and Soft Real-Time System. Hard Real-Time System needs that task deadlines must be met; otherwise, the disastrous situation will arise whereas in Soft Real-Time System, lost an occasional deadline is unwanted but reasonable. Real-time task manager aims to make sure that it meets the deadline for scheduled tasks in the system when we consider the soft real-time system. Vast re-searches are going on real-time task scheduling in order get this desired target. In general, all the real-time systems that exist use preemption and multitasking. Real-Time scheduling methods are widely separated into two methods: Static and Dynamic Methods. Static methods allocate all priorities at design time, and it remains steady for the lifespan of a task. Dynamic methods keep changing the priority at the scheduled time, based on design parameters of any job. Dynamic methods can be endured with static or dynamic priority. Rate Monotonic (RM) and Deadline Monotonic (DM) are the examples of the dynamic scheduling method with static priority [1][2]. There are examples of dynamic scheduling with dynamic priority such as- Earliest Deadline First (EDF) and Least Slack Time First (LST). These algorithms are most the favorable where jobs are preemptable, consist of a single processor, which in turn is under-loaded [3],[4]. However, the constraint of such algorithm is its performance, which diminishes exponentially if the system becomes somewhat overloaded [5].The scheduling is treated as online if the scheduler forges scheduling outcome and doesn't know about the task that is to be released in the future. It is stated that, in an overloaded situation, no other online scheduling algorithm can attain a competitive factor prominent than 0.25. Certainly, many researchers have identified that for any system whose loading factor is nearly equal to 1, the competitive factor of an online scheduling algorithm is nearly equivalent to 0.385 [6],[7]. Certain features make ACO based algorithm an exclusive method: it is effective, population-based metaheuristic that feeds an indirect form of memory of an earlier performance [8][9]. That is one reason why we have considered the same approach for RTOS scheduling.This paper has aimed to formulate as follows: In Section II, the projected algorithm is described and explained. Section III contains mathematical proofs for this algorithm, which includes three Propositions and two Theorems. Section IV illustrates the Simulation method, System and Task Model. Section V represents Results and Discussion and the paper ends with a concise decision in Section VI.

Revised Manuscript Received on October 05, 2019.

***** Correspondence Author

Jay Teraiya*, Computer Engineering Department, Marwadi University, Rajkot, India.

Apurva Shah, Computer Science and Engineering Department, The Maharaja Sayajirao University of Baroda, Baroda, India.

Ketan Kotecha, The Symbiosis Institute of Technology, Pune, India.

II. THE PROPOUND METHOD

The scheduling method is required to plan when a directly running task completes or any new task gets generated. The main steps of the method are shown in subsequent sections, and the consecutive algorithm has been described.

1. Design a journey of distinct ants to yield the better execution sequence of the task.
2. Evaluate the sequences of the task for the given processor.
3. Modify pheromone value.
4. Calculate the probability of all tasks and chose the best task for execution.

A. Creation of Tour

One is required to find the probability of each task using equation in the initial phase. (1) In addition to that, all schedulable tasks are considered as a node and using pheromone τ, and heuristic value η, the probability of all nodes are selected for execution,

$$P_i(t) = \frac{(\tau_i(t))^{\alpha} \times (\eta_i(t))^{\beta}}{\sum_{l \in R_1} (\tau_i(t))^{\alpha} \times (\eta_i(t))^{\beta}} \qquad (1)$$

Where,

$P_i(t)$ is the probability of ith fork at time t; where i ∈ N_1 and N_1 is a set of the node (schedulable tasks) at time t.

- $\tau_i(t)$ is the value of pheromone of ith node at time t.
- η_i is the value of heuristic of ith node at time t, which can be regulated as,

$$\eta_i = \frac{K}{D_i - t} \qquad (2)$$

Here, t is the current time, K is constant (scale 5 - 10) and D_i is the absolute deadline of ith fork.

- α and β are the constants that decide the significance of τ and η.

Ants form their journey based on value p for each fork, as per the following,

- Ant-1: 1st maximum p(t) → 2nd maximum p(t) → 3rd maximum p(t) →
- Ant-2: 2nd maximum p(t) → 1st maximum p(t) → 3rd maximum p(t) →
- Ant-3: 3rd maximum p(t) → 1st maximum p(t) → 2nd maximum p(t) →

Consider on-time t; there are four tasks schedulable shown in Algorithm 1. Each task will be served as a fork, and from another fork, an ant will start its tour. Let's assume the preference of all the forks is in descending order such as T1, T2, T3, T4; ants will pass over different forks as per the following paths.

- Ant-1: T1→ T2→ T3→ T4
- Ant-2: T2→ T1→ T3→ T4
- Ant-3: T3→ T1→ T2→ T4
- Ant-4: T4→ T1→ T2→ T3

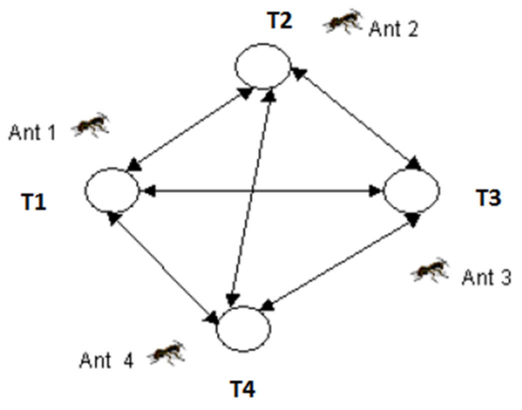

Fig. 1.Ants Journey.

B. ACO Based Algorithm

Once ants have finished their respective journeys, calculate the progress of all ant's journey is calculated. We studied this foundation based on relative number of successful tasks and missed tasks. After that, consider the two leading trips of ants and modify the pheromone cost consequently.

Algorithm 1: ACO Based Scheduling

```
Input: A set of Processes, Pheromone (τ), Heuristic Value (η), (α, β, ρ) are constants.

Output: Executes the Most Important Process.

for each New Process Arrives or Currently running process complete do
    if Is Ready Queue is Empty then,
        Wait;
    /* this step identifies the most suitable process for execution */
    Compute Most_important_Process() ;

    Analyze the Ant's Journey using two tasks:
        Success Task = {Successfully Scheduled: Total Task Arrived};
        Missed Task = {Unsuccessfully Scheduled: Total Task Arrived};
    /* Update of Pheromone is needed to forget wicked journey of ants */
    Compute Pheromone_update() to satisfy the Most_Important_Process()

    Determine the Probability of each process using Most_Important_Process and execute the process having the highest probability.
end
```

Most_Important_Process (P_i(t)) (*Set of Process P*) for the ith node at time t.

/* Probebility of each task will be calculated based on following equation 1. */

Calculate $P_i(t) = \frac{(\tau_i(t))^{\alpha} \times (\eta_i(t))^{\beta}}{\sum_{l \in R_1} (\tau_i(t))^{\alpha} \times (\eta_i(t))^{\beta}}$

```
Pheromone_Update (τ)

        Calculate Evaporation $(\tau_i) = (1-\rho)\,\tau_i$ to ignore the
        lousy path and support new paths.
        Calculate Best of two paths to get the Best Path
        $(\tau_i) = \tau_i + \Delta\tau_i$
```

C. Update Pheromone Value

Pheromone update on every node will be done via two different operations:

1. Evaporation Value of Pheromone: Pheromone evaporation is needed to for-get the lousy journey of ants and to support new paths. Value of τ is updated using,

$$\tau_i = (1-\rho)\tau_i \qquad (3)$$

Here,

- ρ is constant (suitable value is 0.2 to 0.4).
- i∈ N_1; N_1 is set of all (schedulable and non-schedulable) task.

2. Value of Pheromone Laying: Pheromone will be adjoined only for two ultimate journeys of ants. Select the most favorable journey and add pheromone to it, based on their order of travelling node. The quantity of pheromone (Δτ) added will be different and vary from node to node, i.e., the possible nearby node will get the highest quantity of pheromone, and the farthest node will get the smallest quantity.

$$\tau_i = \tau_i + \Delta\tau_i \qquad (4)$$

Where,

- i∈ N_2, N_2 is set of nodes travel by the ants.
- $\Delta\tau = \frac{ph}{s}$ (5)

Here,

- $ph = C * \frac{Number\ of\ success\ tasks}{Number\ of\ Missed\ tasks+1}$ (6)
- S is the sequence number of any fork that is hit by the ant during its leading journey.
- C is a constant (near to 0.1).

D. Selection of Execution Task

After modifying the pheromone value, one needs to compute the possibility of every node bye Eq. 1, then chose the new task for further enactment that has the outrageous value of probability.

E. Algorithm Key Points

- All schedulable tasks are considered as a node, they store τ values, and it is pheromone. The pheromone τ is initialized with value 1 for each node.
- α and β values are decided for the weightage of τ and η. In the experiment, both constants have given equivale weightage which is 1.
- A number of ants which construct the tour is essential in design criteria. During the test, the system is having the same time, and the number of ants decided based on the number of executable tasks.

III. MATHEMATICAL PROOF FOR THE PROJECTED ALGORITHM

The probability of each node will be calculated based on Eq. 1. It will decide which task one should execute to get an optimal result in the proposed algorithm. Following mathematical propositions and theorems have been given with its proof.

Proposition 1: After analyzing journey pheromone will be increased at the rate of $\Delta\tau_i$ (Eq. 4), where $\Delta\tau_i > \Delta\tau_{i+1}$, i∈$N_2$, N_2 is a set of nodes travel by the ants.

Proof - Possible amount of pheromone added to any node after analyzing the journey is $\Delta\tau_i$, Where $\Delta\tau = \frac{ph}{s}$ (Eq. 5) , s is the sequence number of nodes visited by ant during the tour and ph value will be identified based on Eq. 6. Clearly, at first node maximum, possible pheromone is $\frac{ph}{1}$, for the second node it is $\frac{ph}{2}$ and so on. It means the nearest node will get the highest amount of pheromone and far most will get least.

Proposition 2: Pheromone will be decreased at the rate of $(1-\rho)\tau_i$ (Eq. 3) $\forall_i \in N_1$, where ρ is constant and N_1 is the set of schedule and non-schedule task at that time.

Proof - Pheromone evaporation is required to forget the lousy journey of ant and to encourage new paths. Possible amount of pheromone decreases to any node after analyzing the journey is$(1-\rho)\tau_i$.

Theorem 1: Let P be the probability that the algorithm finds an optimal solution within the first analyzing journey, then for an arbitrary small $\epsilon > 0$, $P \geq 1-\epsilon$. By definition$P_{max} = 1$.

Proof - For best two journeys, $i \in N_1 \cap N_2$, where i is the task which is part of both ant journey then pheromone lying will be done on i is $\Delta\tau_i$ as per proposition-1 and according to Eq. 1, the probability P_i will increase.

If $i \notin N_2$ and $i \in N_1$ then pheromone value τ_i will continuously decreasing and it will help us to ***forget*** a bad journey. Due to pheromone trail limits τ_{min} and τ_{max} ***one*** can guarantee that any feasible choice in Eq. 1, for any solution is made with a probability$P_{min} > 0$ [15]. At trivial lower bound for

$$P_{min} \geq \frac{\tau_{min}^{\alpha}}{(N_1 - 1)\tau_{max}^{\alpha} + \tau_{min}^{\alpha}} \qquad (7)$$

Proposition 3: Once an optimal solution has been found for any task such that $\notin N_1$, it holds that $\tau_i = 0$.

Proof - After the execution of the task, the task will not belong to the optimal solution and do not receive pheromone anymore.

Theorem 2: The probabilistic decision taken by ant will be biased when incorporating heuristic information into an ACO based solution.

Proof - Prior available information on the schedulable task can be used to derive heuristic information that biases the probabilistic decision taken by the ant (Eq.2). When assimilating such heuristic information into ACO solution, the favorable choice is $F_i(\tau_i) = (\tau_i)^{\alpha} \times (\eta_i)^{\beta}$. Based on Eq. 1 and Eq. 2 η_i measures the heuristic desirability of choosing a solution as a task i. Infect, Theorem-1 are not going to affected by the heuristic information, η is limited to some (instant specific) interval $[\eta_{\min}, \eta_{max}]$ with $\eta_{min} > 0$ and $\eta_{max} < +\infty$. Then the heuristic information only affected of changing the lower bound of the probability P_{min} of making a specific decision.

IV. SIMULATION METHOD, SYSTEM AND TASK MODEL

When the task is released, we pretend that the system already knows about the task deadline and imperative data to figure out the required time to complete the task. The task set is considered to be preemptive and pretending that the system doesn't have any constraint of resource clash. Furthermore, it also considered that scheduling and preemption do not have any other overhead. In Soft RTOS, each task possesses a positive value. The simple ideology is to yield as much benefit as possible. If a particular task succeeds, then the system contemplates its benefits; otherwise, the system attains less benefit from the task [10]. In a distinct case of firm real-time system, that suggest if any task missed its deadline, then no value will be mediated, but there is no collapse as well [11]. With this work, we propound an algorithm which affixes to the firm real-time system, and the value of the task has been treated very similar to that of its required computation time.[12].

This paper analyses proposed algorithm with the EDF algorithm and execute the simulations to gather the experimental outcomes; also, we considered periodic tasks in order to get effective results. For that, a system load can be described as the aggregate of the ratio of executable time and the time of each task. In order to achieve effective results, at every load value we have produced 7500 task sets and every load contains utmost 1 to 9 tasks. The outcomes from this experiment contain different values of load (ranges from 0.5 - 5), and it examined on 35,000+ task. Moreover, the results of this phenomena are revealed in Table 3 and Figure 3 [16]. Higher the amount of work ***is*** scheduled, the better and competent the algorithm is. For this reason, we have measured the two of our main performance metrics:

1. In RTOS, meeting the deadline is utmost significant and crucial, and therefore, we are more concerned towards result, whether the task is meeting the deadline or not. Based on that, the most reliable metric that we get is Success Ratio (SR), and is defined as [13],

$$SR = \frac{Number\ of\ Task\ successfully\ scheduled}{Total\ Number\ of\ Task\ arrived} \quad (8)$$

2. It is potentially important to know how effectively the scheduler exploits the processes, peculiarly during heavy load condition. Therefore, we also considered other performance metrics such as Effective CPU utilization (ECU) and is defined in [16],

$$ECU = \sum_{i \in R} \frac{V_i}{T} \quad (9)$$

Where,

- V is the task value and,
    - ○ Task Value = Estimated time of the task if the task accomplishes its work within its deadline.
    - ○ Task Value = 0 if the task fails in order to meet its deadline.
- R is a task set, which is scheduled profitably, i.e., executed within its deadline.
- T is the scheduling total time.

Table- I: Result Obtained with Load <= 1

Load	%ECU		%SR	
	EDF Algorithm	*ACO Based Algorithm*	*EDF Algorithm*	*ACO Based Algorithm*
0.50	49.96	49.97	100	100
0.55	55.04	55.04	100	100
0.60	59.88	59.88	100	100
0.65	64.99	64.99	100	100
0.70	69.92	69.92	100	100
0.75	74.87	74.87	100	100
0.80	79.87	79.87	100	100
0.85	84.71	84.72	100	100
0.90	89.61	89.61	100	100
0.95	94.54	94.54	100	100
1.00	99.36	99.36	100	100

An online scheduler has a competitive factor C_f that exist if and only if the value of the schedule of any finite sequence of tasks formed by the algorithm is at least C_f times the value of the schedule of the tasks formed by an optimal clairvoyant algorithm [7]. Since maximum value, seized by a clairvoyant scheduling algorithm is a hard problem, therefore we have instead used a rather condensed upper bound on this maximum value, which can be obtained by summation of the value of all tasks [14]. Hence, for the clairvoyant scheduler, we have considered the value of ECU as 100%.

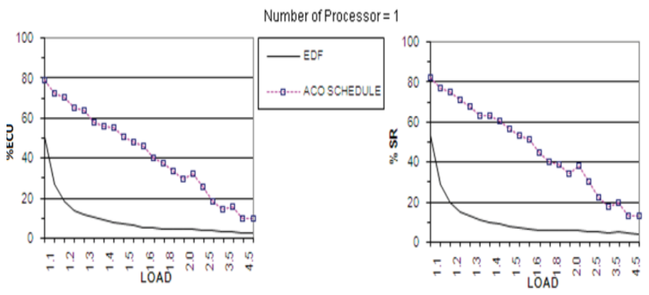

Fig. 2. CPU Load Vs. %ECU and CPU Load Vs.

V. RESULTS AND DISCUSSION

From the empirical study, it is perceived that when the system is not heavily loaded, our projected algorithm gives an ideal result for a unified processor and the preemptive conditions. Table 1 displays the outcomes achieved by our algorithm and the EDF algorithm under loaded conditions. In addition to that, Fig. 2. specifies the results of an overloaded condition. Furthermore, presumed %SR and %ECU of EDF drop quickly; however, our algorithm works prominently and gives efficient progress. The values of %ECU and the maximum value of the clairvoyant scheduler, we notice that the competitive factor of our algorithm is greater than 0.595 and 0.425 when loads are 1.25 and 1.50. Furthermore, in under loaded conditions, the competitive factor of our scheduling has been found to 1.00 and up to load ≤ 1.

VI. CONCLUSION

In this work, an algorithm specifically for the scheduling of a soft real-time system with a unified processor and the preemptive task have been introduced. In addition to that, for scheduling, ACO has been motivated and introduced. The projected method is implemented with a periodic task, and cumulative outcomes are gathered and collate it with EDF. From the mathematical proof, shown in this work and the results of the experiment, this paper concluded that the projected method accomplishes equally best for a single processor, preemptive conditions when the system is heavily load-ed. This paper has also monitor and analyze the performance of EDF that significantly diminished, during maximum loaded conditions; however, the profound algorithm works in a much better way. So, for real time scheduling it is possible to use swarm techniques for batter performance in underload as well as in overload scenario. In future more Swarm Intelligence methods like PSO, GA etc... can be explored to implement Soft Real Time Schedulers.

ACKNOWLEDGMENT

We would like to acknowledge and give our sincere thanks to leading India ai group which is a nationwide initiative on “AI and deep learning Skilling and Research”. It is funded by Royal Academy of Engineering, UK under Newton Bhabha.

REFERENCES

1. J. Teraiya and A. Shah, “Comparative Study of LST and SJF Scheduling Algorithm in Soft Real-Time System with its Implementation and Analysis”, in International Conference on Advances in Computing, Communications and Informatics (ICACCI), Banglor, India, Proceeding in the IEEE Xplore, pp. 706-711, 2018.
2. C. Liu and J. Layland, “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment”, in Journal of the ACM, vol. 20, no. 1, pp. 46-61, 1973.
3. M.Dertouzos and K.Ogata, “Control robotics: The procedural control of physical process,” in Proceedings IFIP Congress (IFIP'74), pp. 807-813, 1974.
4. A. Mok and A. Ka-Lau, “Fundamental design problems of distributed systems for the hard-real-time environment”, Thesis (Ph. D.), Massachusetts Institute of Technology, Cambridge, 1983.
5. G. Saini, “Application of fuzzy logic to real-time scheduling”, in 14th IEEE-NPSS Real Time Conference, Stockholm, Sweden, pp. 113-116, 2005.
6. S. Baruah, G. Koren and B. Mishra, “On the competitiveness of on-line real-time task scheduling”, in IEEE Proceedings 12th Real-Time Systems Symposium, San Antonio, TX, USA, pp. 106-115, 1991.
7. J. Liu, Real-time systems. Upper Saddle River, N.J.: Prentice Hall, 2009.
8. M. Dorigo and G. Caro, “The Ant Colony Optimization Metaheuristic” In D.Corne, M. Dorigo and F.Glover(eds)”, New Ideas in Optimization, McGraw Hill, pp-13-49, 1999.
9. V.Ramos, F.Muge, and P.Pina, “Self-organized data and image retrieval as a consequence of inter-dynamic synergistic relationships in artificial ant colonies,” Soft Computing Systems – Design, Management and Applications, inProceedings of the 2nd International Conference on Hybrid Intelligent System, IOS Press, Santiago, 2002.
10. Carey, Douglass, Locke “Best-effort decision-making for real-time scheduling”, Doctoral Dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 1986.
11. G.Koren and D.Shasha, “Dover: An optimal on-line scheduling algorithm for overloaded real-time systems,” in SIAM Journal of Computing, Vol. 24(2), pp. 318-339, April 1995.
12. A Shah , “Adaptive scheduling algorithm for real-time distributed systems ”, in Biologically-Inspired Techniques for Knowledge Discovery and Data Mining pp. 236-248, 2014.
13. K. Ramamritham, J. Stankovic and P. Shiah, “Efficient scheduling algorithms for real-time multiprocessor systems”, in IEEE Transactions on Parallel and Distributed Systems, vol. 1, no. 2, pp. 184-194, 1990.
14. S. Baruah, G. Koren, B. Mishra, A. Raghunath, L. Roiser and D. Shasha, “On-line scheduling in the presence of overload”, in Proceedings 32nd Annual Symposium of Foundations of Computer Science, San Juan, Puerto Rico, USA, pp. 100-110, 1991.
15. M. Dorigo and T. Stützle, “Ant colony optimization”. Cambridge, Mass.: MIT Press, pp. 131-132, 2004.
16. K. Kotecha and A. Shah, “Scheduling Algorithm for Real-Time Operating Systems Using ACO.”, in International Conference on Computational Intelligence and Communication Networks, Bhopal, India, pp.617-621, 2010.

AUTHORS PROFILE

Jay Teraiya has completed Bachelor of Engineering from GCET – Vallabh Vidyanagar under S. P. University. He has also completed his M. S. in Software Engineering from BITS Pillani. He is pursuing in Ph. D from the M. S. University of Baroda under the guidance of Dr. Apurva Shah.

Dr. Apurva Shah, Associate Professor and Head of Department (Computer Science and Engineering) in Faculty of Technology, the M. S. University of Baroda Gujarat. He is also director of Computer Center in the University. His area of interest are Real Time System, Artificial intelligence and distributed computing. He has completed his Ph. D. from S. P. University Vallabh Vidyanagar.

Dr. Ketan Kotecha, Professor, Computer Science & Engineering, Head, Symbiosis Centre for Applied Artificial Intelligence (SCAAI) Dean, Faculty of Engineering, Symbiosis International (Deemed University) Director, Symbiosis Institute of Technology Chief Executive Officer (CEO), Symbiosis Centre for Entrepreneurship and Innovation TEDx speaker 2015 | Author – Introduction to Critical Thinking (Macmillan) Recipient of Erasmus + faculty mobility grants from European Union.

Evolutionary Intelligence
https://doi.org/10.1007/s12065-021-00599-6

RESEARCH PAPER

Optimized scheduling algorithm for soft Real-Time System using particle swarm optimization technique

Jay Teraiya[1] · Apurva Shah[2]

Received: 5 May 2020 / Revised: 16 March 2021 / Accepted: 21 March 2021

Abstract
Scheduling of tasks in Real-Time Systems is based on static or dynamic priority like earliest deadline first (EDF) and rate monotonic, respectively. The static scheduler does not give assurance of scheduling all tasks during the underload scenario, whereas dynamic scheduler performs poorly during an overload scenario. This paper has proposed a swarm intelligence-based scheduling algorithm that can overcome both the situations. This paper has used particle swarm optimization (PSO) based swarm technique to design the new scheduling approach. It considers each task as a particle and applied modified PSO technique to identify the most critical task to execute. The efficiency of the newly proposed method has been compared with existing EDF and ACO based scheduling algorithms considering two significant parameters, the success ratio and the effective CPU utilization. All three algorithms have been tested on the simulator with a Soft Real-time periodic task set on 500 timelines. It has been observed that during the underload scenario, the proposed algorithm performs equally to EDF and ACO based algorithms. During overload and highly overload situations, the proposed algorithm performs batter compared to EDF and ACO based algorithms.

Keywords PSO · ACO · EDF · Real-Time System · Scheduling

1 Introduction

Real-Time Systems have become part of human life to complete their day to day needs. Real-Time System has lots of applications surrounding us like digital control systems, flight control, vehicle control, healthcare devices, IoT devices, and many more. In the twenty-first century, usage of Real-Time Systems has increased widely. Like a conventional operating system, we also use Real-Time Systems in our day to day life, but when Real-Time Systems work well, and they make us forget their existence. Real-Time System focuses on completion of the task before its deadline, whereas the conventional operating system tries to give minimum response time for any given time. There is always a specific deadline associated with Real-Time Task, whereas typical task does not have any particular timeframe. Text Editor, Browser, music players are examples of such typical application, whereas Smart Watch, aircraft control, and missile control systems are the example of Real-Time applications [1].

Real-Time System is divided into mainly three categories like Hard Real-Time, Soft Real-Time, and Firm Real-Time System based on their timing constraints. Real-Time Systems will be considered as Hard Real-Time System if the failure to meet its deadline is deemed to be a fatal fault. In Contrast, the Soft Real-Time System with few misses of the deadline does not cause serious harm; only the system's overall performance becomes poorer when such more jobs miss their deadline. In Firm Real-Time System if a task misses its deadline, then the result of the given task will be ignored. In Real-Time Systems, considering that each unit of work is scheduled and executed by the system, a job and a set of related jobs which provide some system function a task. Tasks again are divided into three different categories, like Periodic, Aperiodic, and Sporadic Tasks. The periodic task model, each computation that is executed repeatedly at regular time intervals to provide a functionality of the

✉ Jay Teraiya
jay.teraiya@gmail.com

Apurva Shah
apurva.shah-cse@msubaroda.ac.in

1 Marwadi University, Rajkot, Gujarat, India

2 The Maharaja Sayajirao University of Baroda, Vadodara, Gujarat, India

system regularly [2]. Aperiodic and Sporadic task is a set of aperiodic or sporadic jobs, respectively. The interarrival times between consecutive jobs in such a task may vary widely, and, in particular, it can be arbitrarily small.

This paper is organized as follows. Section 2 represents related work carried out till now. The proposed algorithm has been described in Sect. 3, where it shows fundamental of PSO, proposed scheduling algorithm with its detail steps, and parameter selection and one case study for instance of the task set. Section 4 describes the simulation scenario, dataset, and performance parameter. Critical analysis of the proposed scheduling algorithm has been done with EDF and ACO based scheduling algorithm in Sect. 5. Finally, Sect. 6 states the conclusion of this paper.

2 Related work

Deciding the scheduling algorithm for Real-Time System is a crucial task. The decision is taken based on the type of Real-Time System; task type and task are pre-empted or not. Scheduling of tasks is the process of identifying which task should be executed at each instant of time. Priority driven scheduling algorithms are implemented based on specific priority parameters. At runtime, the scheduling algorithm assigns priority to each active task and allocates the processor based on the highest-priority task. Based on the way priority assign to the task, Priority driven schedulers are divided into two significant categories call Static Priority and Dynamic Priority Scheduler. A static priority scheduler also referred to as a fixed priority scheduler where each periodic task is assigned a unique priority [3]. The Rate Monotonic (RM) and Shortest Job First (SJF) are examples of the Static priority scheduler. RM assigned priority to the task based on its period parameter, and the task with the smallest period assigned the highest priority. The SJF assigned priority to the task based on its execution time, and the task with the shortest execution time assigned the most top priority. The dynamic priority scheduler does not put restrictions upon how priorities are attached to the task. The priority of a task may change arbitrarily often between its release time and its completion time. The Earliest Deadline First (EDF) and Least Slack Time First (LST) are examples of dynamic priority scheduler. The EDF, in which the priority of task depends on its deadline, a task with the earlier deadline has the highest priority. The LST, in which the priority of task depends on its slack time, a task with the shortest slack time has the most top priority [4, 5].

Different Static (like RM, SJF) and Dynamic (like EDF, LST) scheduling algorithms have been evaluated and compared by various researchers for Soft Real-Time System. The performance of Static and Dynamic scheduler varies based on the CPU Load. It has been observed that EDF and LST perform well in the underload scenario where CPU load is less than or equal to one. Even EDF is one of the optimal scheduling algorithms, and it makes sure that in the underload scenario, all task will meet their deadline. Static algorithms like RM and SJF also perform well in the underload scenario. Still, it has been observed that in some instances where there is possible to schedule all task by EDF, but static algorithms fail to schedule those tasks and few tasks missed their deadline. In a slightly overload situation, when CPU load is higher than one at that time performance of Dynamic scheduler degrades very fast. In contrast, static schedulers are still able to schedule a few tasks and able to meet their deadline. Thus, Dynamic Priority Scheduler performs well in the underload scenario, and Static Priority Scheduler performs decently in the overload scenario [6].

The Dynamic Priority schedulers are more responsive to the average cases, but their worst-case real-time performance may be more unsatisfactory than the Static Priority scheduler. Still, there is no single priority scheduling algorithm exist which perform well in underload and overload scenario. Researchers have developed few hybrid priorities driven scheduling algorithms which are using characteristics of both type of algorithm, like D_EDF and S_LST which is using features of dynamic scheduling algorithm during underload scenario and static scheduling algorithm during overload scenario [7–9]. The problem with this kind of hybrid algorithm is scheduler needs to keep checking with the status of CPU load, and based on that, it will assign the priority to the task. Researchers have given an entirely new direction for scheduling tasks using Artificial Intelligence and Swarm techniques. Swarm intelligence is the study of computational systems inspired by collective intelligence. Collective Intelligence emerges through the cooperation of large numbers of homogeneous agents in the environment. Examples include schools of fish, flocks of birds, and colonies of ants. Such intelligence is decentralized, self-organizing, and distributed throughout an environment. Using Swarm intelligence, it is possible to find optimal solutions for problems like scheduling of the task [10, 11]. The researcher has proposed ACO based scheduling algorithm, and it has been shown that the swarm-based scheduling algorithm performs equally well like Dynamic scheduler. It gives batter performance in an overload scenario as well [12, 13].

Particle Swarm Optimization has been widely used in scheduling for the Cloud Computing environment. Researcher A. S. Ajeena Beegom and M. S. Rajasree proposed the Integer-PSO algorithm for task scheduling in a cloud computing system in 2019 [14]. A two-level particle swarm optimization algorithm created for the flexible job-shop scheduling problem [15] and PSO based scheduling also applied in workflow applications in Cloud Computing Environments by researchers [16]. An Adaptive PSO-Based Real-Time Workflow Scheduling Algorithm has been introduced by researcher for Cloud

Systems. Researchers have targeted reducing execution time and reducing execution cost which are two conflicting objectives and has been address in paper [17]. Medhat Awadalla and Abdullah Elewi has proposed Enhanced PSO Approach for energy-aware static partitioning of periodic real time tasks on heterogeneous multiprocessor platforms [18]. PSO based approach also used with GA approach to solve Real-Time Order Acceptance and Scheduling Problems in a Flow Shop Environment [19]. Although, the PSO is integrated to scheduling in all concerned fields including real time system, there is still there is sufficient scop for exploration. In this paper, we are addressing few of such gaps in this area. Particle Swarm Optimization (PSO) investigates probabilistic algorithms inspired by the flocking. Swarm intelligence algorithms strategies are considered adaptive strategy and are typically applied to search and optimization domains. This paper is selecting PSO because it is the right approach when the problem size is between 20 and 40 [20, 21]. This paper has considered that scheduling task problem in a soft real-time system. Static and Dynamic schedulers have their advantages and disadvantages, and both are not able to perform well in overload and underload scenario. The researcher has developed an ACO based scheduling algorithm. This paper is proposing a PSO based scheduling algorithm that will overcome the disadvantages of the Static and Dynamic scheduling algorithm with retaining its advantages and trying to introduce scheduler which performance is batter then ACO based scheduler as well.

3 Proposed algorithm

3.1 Particle swarm optimization

The particle swarm optimization algorithm is comprised of a collection of particles that move around the search space influenced by their own best past location and the best past location of the whole swarm or a close neighbour. Each iteration, a particle's velocity is updated using following Eq. 1 [21–24].

$$v_{i,d}(t+1) = v_{i,d}(t) + \left(c_1 \times r_1 \times \left(p_{i,d}^{best} - p_{i,d}(t)\right)\right) + \left(c_2 \times r_2 \times \left(p_{gbest,d} - p_{i,d}(t)\right)\right) \tag{1}$$

$$p_{i,d}(t+1) = p_{i,d}(t) + v_{i,d}(t+1) \tag{2}$$

where $v_{i,d}(t+1)$ and $v_{i,d}(t)$ represent the current and previous velocity in the dth dimension of particle i, respectively. c_1 and c_2 are acceleration coefficient for the personal best and global best positions respectively. $p_{i,d}(t+1)$ and $p_{i,d}(t)$ are the current and previous position of particle i. $p_{i,d}^{best}$ and $p_{gbest,d}$ are the best position found by particle i so far and the best position found by the whole swarm so far, respectively. r_1 *and* r_2 are the randomly generated numbers in the range of [0, 1]. $d \in D$ is the dimension d in the search space.

Variants on this update equation consider the best positions within a particle's local neighbourhood at time t. A particle's position is updated using the Eq. 2 [21].

Figure 1 shows the graphical representation of the particle swarm optimisation. After each iteration the particle moves in a new direction and most of the time it is optimal, and that decision will be based on the personal best position and global best position.

Heuristics for this approach are [20–22]:

- The number of particles should be low, around 20–40,
- The speed a particle can move should be bounded,
- The learning factors should be between 0 and 4, typically 2.0,
- Particles may leave the boundary of the problem space and maybe penalized, be reflected in the domain, or biased to return toward a position in the problem domain. Alternatively, a wrapping strategy may be used at the edge of the domain, creating a loop, or related geometrical structures at the chosen dimensionality.
- An inertia or momentum coefficient can be introduced to limit the change in velocity.

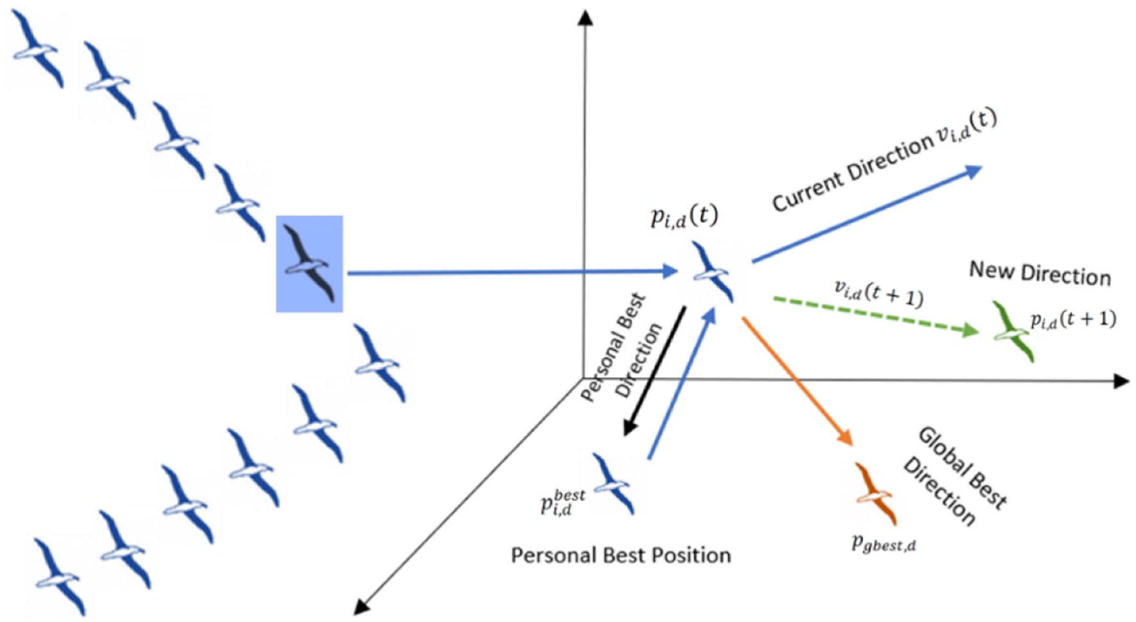

Fig. 1 Graphical representation of PSO

3.2 PSO based scheduling algorithm

Algorithm 1: PSO based scheduling approach for Soft Real-Time System

```
Input  : Task Set $N = \{T_1, T_2, T_3, \dots T_n\}$
Output : $T_{most\_imp\_task\_to\_execute}$
 1. $T_{most\_imp\_task\_to\_execute} = -1$;
    /* Initializing each task $T_i$ ($T_i \in N$)with its initial velocity($v_i$), position ($p_i$) and local
    best position ($p_i^{best}$) value. İt also initializing global best ($p_{gbest}$) value. */
 2. for each task $T_i$ in set N do
 3. |     $v_i = T_{i\,(Deadline)}$;
 4. |     $p_i = T_{i\,(Execution\,Time)} + T_{i\,(Period)} - T_{i\,(Elapsed\,Time)}$;
 5. |     $p_i^{best} = p_i$;
 6. |     if $p_i^{best} < p_{gbest}$ then
 7. |     |    $p_{gbest} = p_i^{best}$;
 8. |     |    $T_{most\_imp\_task\_to\_execute} = T_i$;
 9. |     end if
10. end for
    /* Updating velocity($v_i$), position ($p_i$) and local best position ($p_i^{best}$) values of each task
    $T_i$ ($T_i \in N$). İt also update global best ($p_{gbest}$) values after each iteration if condition
    satisfied. At the end of stopping condition $T_{most\_imp\_task\_to\_execute}$ will contain the most
    important task which needs to execute by the processor. */
11. while stopingcondition() do
12. |     for each $T_i \in N$ do
13. |     |    $v_i(t+1) = v_i(t) + \left(c_1 r_1 \left(p_i^{best} - p_i(t)\right)\right) + \left(c_2 r_2 \left(p_{gbest} - p_i(t)\right)\right)$;
14. |     |    if $v_i(t+1) < 0.0$ do
15. |     |    |    $v_i(t+1) = 0.0$;
16. |     |    end if
17. |     |    $p_i(t+1) = p_i(t) + v_i(t+1)$;
18. |     |    if $p_i(t+1) < p_i^{best}$ do
19. |     |    |    $p_i^{best} = p_i(t+1)$;
20. |     |    |    if $p_i(t+1) < p_{gbest}$ do
21. |     |    |    |    $p_{gbest} = p_i(t+1)$;
22. |     |    |    |    $T_{most\_imp\_task\_to\_execute} = T_i$;
23. |     |    |    end if
24. |     |    end if
25. |     end for
26. end while
27. return $T_{most\_imp\_task\_to\_execute}$;
```

The Proposed Scheduling Algorithm Selecting the scheduling algorithm for Soft Real-Time System is a crucial decision, as discussed in Sect. 2. This paper is introducing the scheduler, which is based on PSO techniques. The algorithm considering each given task as a particle, and all tasks which are eligible for scheduling are viewed as a set of particles. The ultimate goal of the scheduler is to choose a task at a given point of time in such a way that the task can meet its deadline [16]. In the Soft Real-Time system, it is intended to make sure that all task will meet their deadline in the underload condition, and the maximum task will meet their deadline in the overload scenario.

The scheduling algorithm will be executed when a new task arrives, or the currently performing task is completed. When there is more than one task is ready to run at that time scheduler needs to select the task effectively. This paper proposed the PSO based scheduler (Algorithm 1), which has the following significant steps.

Step 1: Initialization of Task as a Particle
Step 2: Compute the velocity and position of each task
Step 3: Analyse the position and velocity of each task
Step 4: Selection of Task for execution

Step 1 *Initialization of Task as a Particle* At given point of time all schedulable task is considered as a set of $N = \{T_1, T_2, T_3, \dots T_n\}$. Each task (particle) $T_i \in N$, needs to initialize with its initial position and velocity. Each Periodic task T_i in task set N has essential characteristics associated with it, like execution time of task (E_i), deadline of the task (D_i) and rate (period) of the task (R_i).

These characteristics are already known in Soft Real-Time System before the scheduler is going to select the task for scheduling. Each task (particle)$T_i \in N$, needs to initialize with its initial position (P_i) and initial velocity (V_i). P is the set of the initial position of each task and $P = \{P_1, P_2, P_3, \dots P_n\}$. V is set of the initial velocity of each task and $V = \{V_1, V_2, V_3, \dots V_n\}$. Initial value of v_i and p_i is going to calculate for each task $T_i \in N$ based on the following Eqs. 3 and 4.

$$v_i = T_{i(Deadline)} \tag{3}$$

$$p_i = T_{i(Execution\ Time)} + T_{i(Period)} - T_{i(Elapsed\ Time)} \tag{4}$$

It is also necessary to initialize individual task best position p_i^{best} and global best position p_{gbest}. Initially for each task $p_i^{best} = p_i$ and initial value of p_{gbest} for the whole task set is chosen from a minimum of the set P. Figure 2 represents the task set with its parameters like Execution Time, Deadline, and Rate. For each task, the algorithm initializes its position, velocity, and best position using Eqs. 3 and 4, as described above.

Step 2 *Compute PSO values for each task* Algorithm calculates the velocity (v) for each task which is ready to execute and part of task set N. To calculate the velocity (v) value for each task this algorithm has considered Eq. 1 as a base equation and proposed Eq. 5 and it is an optimal equation for scheduling problem of Soft Real-Time System.

$$v_{i,d}(t+1) = v_{i,d}(t) + \left(c_1 r_1 \left(p_{i,d}^{best} - p_{i,d}(t)\right)\right) + \left(c_2 r_2 \left(p_{gbest,d} - p_{i,d}(t)\right)\right) \tag{5}$$

where $v_{i,d}(t+1)$ is the new velocity of task T_i in the dth dimension, $v_{i,d}(t)$ is the current velocity of task T_i in the dth dimension, $c_1 = (T_{i(Execution\ Time)})^{-1}$, where $T_{i(Execution\ Time)}$ is the execution time of the task T_i, which is required on the processor to complete the task, $c_2 = (T_{i(Deadline)})^{-1}$, where $T_{i(Deadline)}$ is the deadline of the task T_i, r_1 and r_2 are generated uniformly between 0 and 1, $p_{i,d}(t)$ is the T_i task's position at time t in the dth dimension, $p_{i,d}^{best}$ is the T_i task's best-known position in the $d$thth dimension, $p_{gbest,d}$ is the best position known to the entire task set in the dth dimension, $d \in D$ is the dimension d in the search space.

The algorithm also needs to calculate the new position (p) for each task and to calculate it; it is using Eq. 2, mention in Sect. 3.1.

Step 3 *Analyse the position and velocity of each task* The goal of the algorithm is to have all the tasks locate the optima in a multi-dimensional hypervolume. This can be achieved by assigning initial velocity and position to each task as per step 1. The algorithm is executed and, in each iteration, it is calculating the new position of each task based on Eq. 2 and updating its velocity based on Eq. 5. The evolution of velocity and position is carried out for the specified number of iterations, and the number of iterations depends on the problem size. Over the period, through a combination of exploration and exploitation of known right positions in the search space, the task set cluster or converge together around an optimal task. If any task leaves the boundary of the problem space, then it will be penalized and reflected in the domain by changing its velocity [25].

Step 4 *Selection of task for execution* The algorithm calculates new velocity and the new position of the task in each iteration. The algorithm also changes the value of p_{gbest} in every iteration. p_{gbest} value will be set as the smallest p_i^{best} value. The task which has $p_{gbest} = p_i^{best}$ will be considered and will get the chance to execute on the processor.

3.3 Case study for instance of task set

The proposed algorithm in Sect. 3.2 has been tested with a set of the periodic task set. In this section, the paper has demonstrated how it operates with one case study shown in Table 1. Table 1 shows one task set with its arrival time, its deadline, and its required execution time.

As described in Sect. 3.2, each task will be initialized with its initial position (p_i) using Eqs. 3 and 4, and its initial value has been shown in Table 2. To get an optimal position for each task p_i will be calculated for N number of times. After that task set will be evaluated and identify the most important task which we need to execute. In the above task set (shown in Table 1), T5 is the most crucial task, and the scheduler will select it for execution, so it will meet the deadline, as shown in Table 2.

4 Simulation environment for proposed algorithm

4.1 Simulation scenario and dataset

The entire simulator for the proposed algorithm has been developed in C programming language, and the compiler is GNU GCC. The simulator has been executed on hardware configuration—Core i5 processor with 8 GB of RAM. Simulation of the proposed algorithm has been carried out on a 64-bit Windows 10 Enterprise operating system. The Real-Time System has three types of tasks like Periodic task, Aperiodic task, and Sporadic task. The proposed algorithm has been evaluated with a periodic task set. This paper has considered an extensive data set of periodic tasks. İt has found the 6800 tasks set, which vary in terms of CPU load and the number of tasks within the task set. CPU load ranges from 0.5 to 5.0, and the number of task set varies from 1 to 9. CPU load of task set is referred to total processor utilization factor (U_p) and represent the fraction of processor time used by the periodic task set and calculated based on Eq. 6.

$$U_p = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{6}$$

where C_i is execution time required by each task in task set and T_i is the occurrence period of each task in the task set. It has considered the underload scenario, overload scenario, and highly overload scenario. This 6800-task set contains total 28,600 processes, and it has been tested on the 500-time unit to validate the correctness of the algorithm. To confirm the above task set researcher has published the given task set on the website (http://www.processdataset.in/). Table 3 represents task set detail, and several different tasks have been considered for simulation for the proposed algorithm [26].

4.2 Performance parameter

The performance of the proposed algorithm has been tested with two primary parameters call Success Ratio (SR) and Effective CPU Utilization (ECU). These parameters have been described as follows.

SR (Success Ratio) Success Ratio with real-time systems defined as the ratio of a set of the process which meets their deadline and a total number of process. Success Ration determined with the following Eq. 7 [27].

$$SR = \frac{Number\ of\ Task\ successfully\ scheduled}{Total\ Number\ of\ Task\ arrived}. \tag{7}$$

ECU (Effective CPU utilization) Effective CPU Utilization defined as how much CPU time has been utilizing for the processes which can meet their deadline. ECU determined with the following Eq. 8 [27].

$$ECU = \sum_{i \in R} \frac{V_i}{T} \tag{8}$$

where V represents process value and, Process Value = time required to complete the process if the process meets its deadline. Process Value = 0 if the process does not meet the deadline. R is a set of processes, which are scheduled successfully, i.e., completed within their deadline. T is the total time of scheduling.

5 Critical analysis of proposed algorithm

5.1 Results and comparison with different existing algorithm

The proposed algorithm has been compared with Earliest Deadline First (EDF) and Ant Colony Optimization (ACO) based algorithm [26]. The correctness of all three algorithms has been tested under similar hardware and dataset, as described in Sect. 4.1. These algorithms have been implemented in the simulator using the C language. These algorithms have been compared with parameter SR and ECU, as described in Sect. 4.2.

Underload scenario In this scenario paper has considered all dataset where the utilization factor of the task set is less than or equal to one ($U_p \leq 1$). Table 4 shows the results and comparison of these algorithms during the underload scenario. Table 4 compares these three algorithms for SR and ECU parameters. Figures 3 and 4 represents the graphical representation of Table 4. Observation of these algorithm says that EDF and ACO can meet all the deadlines for the given task set, whereas the PSO based scheduling algorithm missed a few deadlines when the load is near to one.

Overload scenario In this scenario paper has considered all dataset where the utilization factor of the task set is

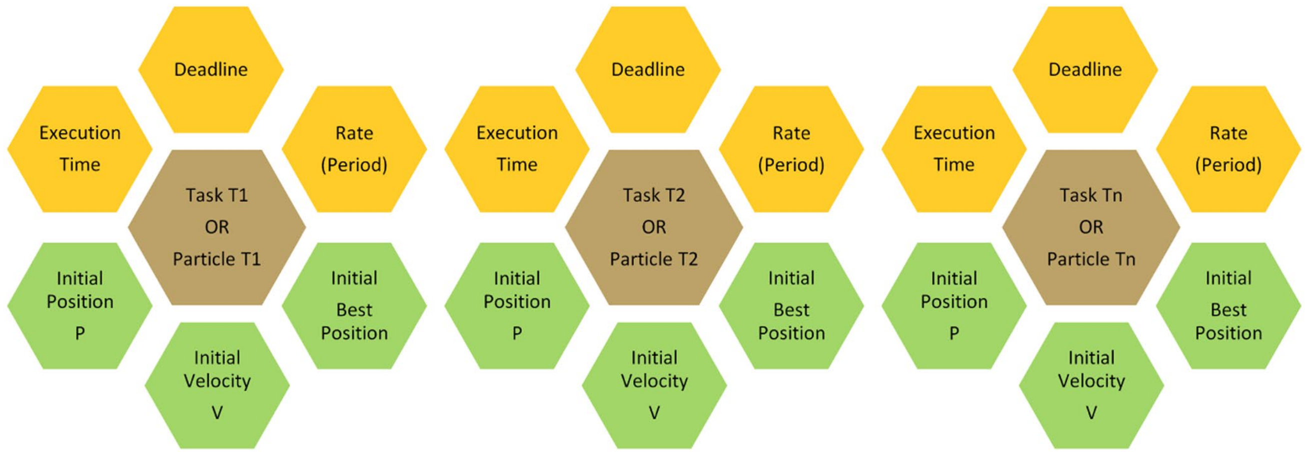

Fig. 2 Task Set for PSO Algorithm with its parameters and initial values

Table 1 An instance of task set for case study

Task	Arrival time	Absolute deadline	Required execution time
T1	0	12	1
T2	0	12	2
T3	0	3	1
T4	0	12	2
T5	0	2	1

Table 2 PSO algorithm calculation for instance of task set

Task	Initial values of p_i	After N iteration values of p_i	Selection for execution of task by PSO algorithm at t = 0 is
T1	13.00	28.25	T5
T2	14.00	32.70	
T3	04.00	07.24	
T4	14.00	32.70	
T5	03.00	05.25	

greater than 1 and less than 1.50 ($1.00 \leq U_p \leq 1.5$). Table 5 shows the results and comparison of these algorithms during an overload scenario. Table 5 compares these three algorithms for SR and ECU parameters. Figures 5 and 6 represents the graphical representation of Table 5. Observation of these algorithms says that EDF performance degrades rapidly in slightly overload situations. Whereas ACO and PSO based scheduling algorithms are still able to meet most of the deadlines of the given task set. Even the PSO based scheduling algorithm performs more batter than the ACO based scheduling algorithm.

Highly Overload Scenario In this scenario, the paper has considered all dataset where the utilization factor of the task set is higher than 1.50 and less than 5.00 ($1.50 \leq U_p \leq 5.00$). Table 6 shows results and comparison of these algorithms during highly overload scenarios. Table 6 compares these three algorithms for SR and ECU parameters. Figures 7 and 8 represents the graphical representation of Table 6. Observation of these algorithms says that EDF performance is abysmal during highly overload scenarios, and even ACO performance is also degraded. PSO based scheduling algorithm is still able to meet many of the deadlines for the given task set. Overall PSO based scheduling algorithm performs far batter compare to EDF and ACO based scheduling algorithm.

5.2 Complexity comparison with different existing algorithm

This section compares the time complexity of EDF, ACO, and PSO based scheduling algorithm. Critical analysis of these algorithms has been done in Sect. 5.1 by implementing these algorithms on the simulator. The experiment set up has been prepared for the periodic task set so, the researcher is giving time complexity comparison for a periodic task only. At a given point of time, all schedulable task is considered as a set of $N = \{T_1, T_2, T_3, \ldots T_n\}$. EDF is a dynamic scheduling algorithm and identifies the most crucial task to execute based on the absolute deadline. When the scheduler is executed to select the most critical task, EDF will have $O(N)$ time complexity [28, 29]. ACO based scheduling algorithm use concept of traversing the different path to identify the optimal route and then select the most crucial task for execution. Due to its traversing techniques, when scheduler will be executed to select the most crucial task ACO based scheduler will have $O(N^2)$ time complexity to select the most crucial task. The algorithm which proposed with this paper also calculate Velocity and Position of each task for N iteration to identify optimal positions in given task set and because of the time complexity of PSO based scheduling algorithm is also $O(N^2)$ to select the most crucial task. It is true that PSO based scheduling algorithm time complexity is higher than EDF but as discuss in Sect. 5.1 it gives an excellent performance in overload scenario and even in the modern evolution of electronics devices Real-Time system able to perform faster and able to schedule a task using any algorithm by ignoring its overhead.

6 Conclusion

The proposed PSO based scheduling algorithm has been compared with EDF and ACO based scheduling algorithm under the Soft Real-Time periodic task set. The performance parameters SR and ECU has been calculated for each algorithm for large dataset and comparison has been done. It has been observed that during the underload scenario ($U_p \leq 1$) proposed scheduling algorithm performs similar to the EDF and ACO based algorithm. In slightly overload situation when $1.00 \leq U_p \leq 1.5$, EDF performance gets degraded sharply. The proposed algorithm and ACO based scheduling algorithm perform batter compare to EDF, and even the proposed approach delivers batter than the ACO based scheduling algorithm. During highly overload scenario ($1.50 \leq U_p \leq 5.00$) EDF and ACO based algorithms

Table 3 Dataset detail for periodic Task Set

Load	Number of Task in each Task Set									Task Set load wise
	1	2	3	4	5	6	7	8	9	
0.50	50	50	50	50	00	00	00	00	00	2200
0.55	50	50	50	50	00	00	00	00	00	
0.60	50	50	50	50	00	00	00	00	00	
0.65	00	50	50	50	50	00	00	00	00	
0.70	00	50	50	50	50	00	00	00	00	
0.75	00	50	50	50	50	00	00	00	00	
0.80	00	50	50	50	50	00	00	00	00	
0.85	00	50	50	50	50	00	00	00	00	
0.90	00	50	50	50	50	00	00	00	00	
0.95	00	50	50	50	50	00	00	00	00	
1.00	00	50	50	50	50	00	00	00	00	
1.05	00	50	50	50	50	00	00	00	00	2000
1.10	00	50	50	50	50	00	00	00	00	
1.15	00	50	50	50	50	00	00	00	00	
1.20	00	50	50	50	50	00	00	00	00	
1.25	00	50	50	50	50	00	00	00	00	
1.30	00	50	50	50	50	00	00	00	00	
1.35	00	50	50	50	50	00	00	00	00	
1.40	00	50	50	50	50	00	00	00	00	
1.45	00	50	50	50	50	00	00	00	00	
1.50	00	50	50	50	50	00	00	00	00	
1.60	00	00	50	50	50	50	00	00	00	2600
1.70	00	00	50	50	50	50	00	00	00	
1.80	00	00	50	50	50	50	00	00	00	
1.90	00	00	50	50	50	50	00	00	00	
2.00	00	00	50	50	50	50	00	00	00	
2.25	00	00	00	50	50	50	50	00	00	
2.50	00	00	00	50	50	50	50	00	00	
2.75	00	00	00	50	50	50	50	00	00	
3.00	00	00	00	50	50	50	50	00	00	
3.50	00	00	00	00	50	50	50	50	00	
4.00	00	00	00	00	50	50	50	50	00	
4.50	00	00	00	00	00	50	50	50	50	
5.00	00	00	00	00	00	50	50	50	50	
Total task Set	150	1050	1300	1500	1450	650	400	200	100	6800
Total task	150	2100	3900	6000	7250	3900	2800	1600	900	28,600

perform poorly, whereas the PSO based scheduling algorithm is still able to meet a specific deadline. So, instead of static or dynamic priority, the proposed approach works well during underload, overload, and highly overload scenarios. The proposed method is tested with uniprocessor and periodic task set for Soft Real-Time System. In future work, this algorithm can be examined with Hard and Firm Real-Time System as well. By making a few changes, the modified PSO based scheduling algorithm can be implemented for the multi-processor system as well.

Table 4 Results comparison during underload scenario

Load	ECU%			SR%		
	EDF	ACO	PSO	EDF	ACO	PSO
0.50	49.49	49.98	49.49	100.00	100.00	100.00
0.55	54.66	55.04	54.40	100.00	100.00	100.00
0.60	59.39	59.88	59.39	100.00	100.00	100.00
0.65	64.35	65.00	64.35	100.00	100.00	100.00
0.70	69.35	69.93	69.35	100.00	100.00	100.00
0.75	74.31	74.88	74.31	100.00	100.00	100.00
0.80	79.22	79.83	79.22	100.00	100.00	100.00
0.85	84.16	84.72	84.16	100.00	100.00	100.00
0.90	89.16	89.62	89.15	100.00	100.00	99.99
0.95	94.17	94.54	94.08	100.00	100.00	99.94
1.00	99.10	99.37	97.99	100.00	100.00	99.26

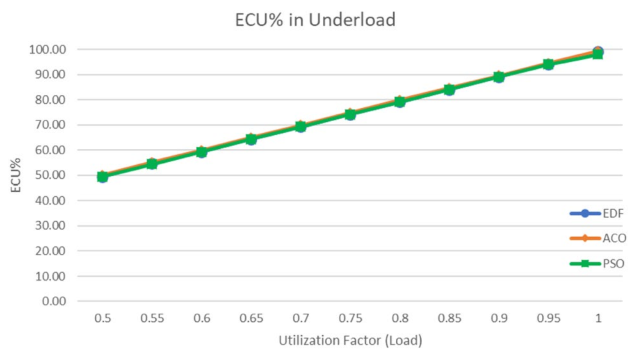

Fig. 3 ECU% comparison of EDF versus ACO versus PSO Algorithm during Underload Scenario

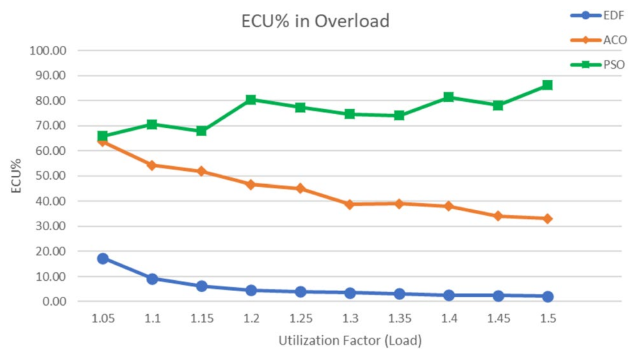

Fig. 5 ECU% comparison of EDF versus ACO versus PSO Algorithm during Overload Scenario

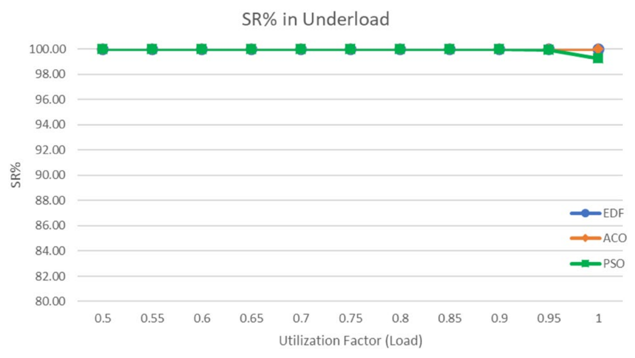

Fig. 4 SR% comparison of EDF versus ACO versus PSO Algorithm during Underload Scenario

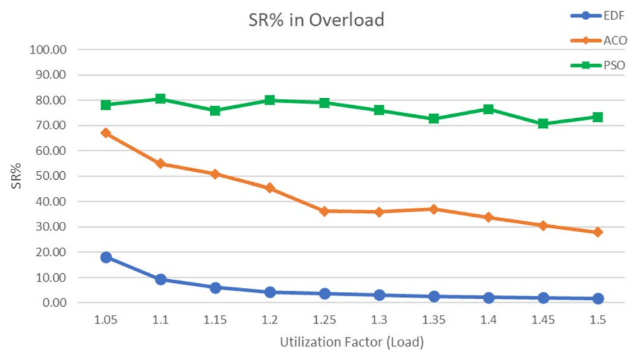

Fig. 6 SR% comparison of EDF versus ACO versus PSO Algorithm during Overload Scenario

Table 5 Results comparison during overload scenario

Load	ECU%			SR%		
	EDF	ACO	PSO	EDF	ACO	PSO
1.05	17.45	63.69	65.91	18.27	67.01	78.24
1.10	09.21	54.22	70.60	09.31	55.01	80.53
1.15	06.29	51.86	67.90	06.19	50.87	75.91
1.20	04.62	46.61	80.39	04.22	45.33	80.03
1.25	04.06	45.15	77.35	03.67	36.23	78.97
1.30	03.63	38.78	74.73	03.19	35.90	76.02
1.35	03.12	39.03	74.06	02.65	37.14	72.65
1.40	02.66	38.05	81.39	02.24	33.91	76.49
1.45	02.50	34.11	78.15	02.00	30.65	70.66
1.50	02.21	33.08	86.15	01.71	27.91	73.35

Table 6 Results comparison during highly overload scenario

Load	ECU%			SR%		
	EDF	ACO	PSO	EDF	ACO	PSO
1.60	2.17	45.98	85.02	1.61	37.25	71.25
1.70	2.03	40.45	86.52	1.42	30.24	68.70
1.80	1.93	35.52	86.34	1.30	26.39	68.27
1.90	1.90	33.56	85.17	1.29	25.35	65.52
2.00	1.84	29.56	86.90	1.20	21.45	65.23
2.25	1.76	32.51	85.89	1.04	21.24	59.81
2.50	1.55	25.54	87.86	0.89	15.39	56.23
2.75	1.46	18.31	88.82	0.78	10.16	53.75
3.00	1.32	14.66	94.25	0.63	07.11	49.21
3.50	1.27	15.80	94.46	0.57	07.69	45.52
4.00	1.11	09.67	96.20	0.43	03.79	40.47
4.50	1.08	09.86	97.69	0.38	03.37	36.99
5.00	0.97	08.74	97.88	0.31	02.41	32.15

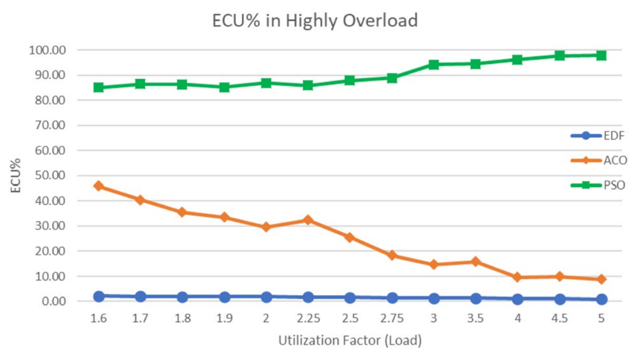

Fig. 7 ECU% comparison of EDF versus ACO versus PSO Algorithm during Highly Overload Scenario

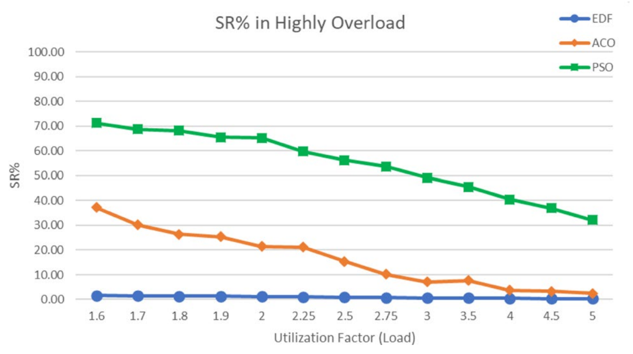

Fig. 8 SR% comparison of EDF versus ACO versus PSO Algorithm during Highly Overload Scenario

References

1. Ahmad S, Malik S, Kim DH (2018) Comparative analysis of simulation tools with visualization based on real-time task scheduling algorithms for IoT embedded applications. Int J Grid Distrib Comput. https://doi.org/10.14257/ijgdc.2018.11.2.01
2. Chatterjee K, Pavlogiannis A, Kößler A, Schmid U (2018) Automated competitive analysis of real-time scheduling with graph games. Real-Time Syst 54(1):166–207. https://doi.org/10.1007/s11241-017-9293-4
3. Wang X, Li Z, Wonham WM (2017) Optimal priority-free conditionally-preemptive real-time scheduling of periodic tasks based on des supervisory control. IEEE Trans Syst Man Cybern Syst 47(7):1082–1098. https://doi.org/10.1109/TSMC.2016.2531681
4. Teraiya J, Shah A (2018) Comparative study of LST and SJF scheduling algorithm in soft real-time system with its implementation and analysis. In: 2018 international conference on advances in computing, communications and informatics, ICACCI 2018, pp 706–711. https://doi.org/10.1109/ICACCI.2018.8554483
5. Kohutka L, Stopjakova V (2016) Improved task scheduler for dual-core real-time systems. In: Proceedings—19th Euromicro conference on digital system design, DSD 2016. Institute of Electrical and Electronics Engineers Inc., pp 471–478. https://doi.org/10.1109/DSD.2016.44
6. Teraiya J, Shah A (2020) Analysis of dynamic and static scheduling algorithms in soft real-time system with its implementation. Adv Intell Syst Comput 1053:757–768. https://doi.org/10.1007/978-981-15-0751-9_69
7. Thakor D, Shah A (2011) D_EDF: an efficient scheduling algorithm for real-time multiprocessor system. In: Information and communication technologies (WICT), 2011 World Congress on, pp 1044–1049. https://doi.org/10.1109/WICT.2011.6141392
8. Teraiya J, Shah A (2019) Hybrid Scheduler (S_LST) for soft real-time system based on static and dynamic algorithm. Int J Eng Adv Technol 9(2):2885–2889. https://doi.org/10.35940/ijeat.b3837.129219
9. Alsheikhy A, Ammar R, Elfouly R, Alharthi M, Alshegaifi A (2016) An efficient dynamic scheduling algorithm for periodic tasks in real-time systems using dynamic average estimation. In: Proceedings—IEEE symposium on computers and communications (Vol. 2016-August). https://doi.org/10.1109/ISCC.2016.7543830
10. Yu SC (2014) Elucidating multiprocessors flow shop scheduling with dependent setup times using a twin particle swarm optimization. Appl Soft Comput J 21:578–589. https://doi.org/10.1016/j.asoc.2014.04.016
11. Kazemi H, Zahedi ZM, Shokouhifar M (2016) Swarm intelligence scheduling of soft real-time tasks in heterogeneous multiprocessor systems. Electr Comput Eng Int J. https://doi.org/10.14810/ecij.2016.5101
12. Shah A (2014) Adaptive scheduling for real-time distributed systems. In: Biologically-inspired techniques for knowledge discovery and data mining, pp 236–248. https://doi.org/10.4018/978-1-4666-6078-6.ch011
13. Konar D, Bhattacharyya S, Sharma K, Sharma S, Pradhan SR (2017) An improved Hybrid Quantum-Inspired Genetic Algorithm (HQIGA) for scheduling of real-time task in multiprocessor system. Appl Soft Comput J. https://doi.org/10.1016/j.asoc.2016.12.051
14. Beegom ASA, Rajasree MS (2019) Integer-PSO: a discrete PSO algorithm for task scheduling in cloud computing systems. Evol Intel 12(2):227–239. https://doi.org/10.1007/s12065-019-00216-7
15. Zarrouk R, Bennour IE, Jemai A (2019) A two-level particle swarm optimization algorithm for the flexible job shop scheduling problem. Swarm Intell 13(2):145–168. https://doi.org/10.1007/s11721-019-00167-w
16. Pandey S, Wu L, Guru SM, Buyya R (2010) A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In: Proceedings—international conference on advanced information networking and applications, AINA, pp 400–407. https://doi.org/10.1109/AINA.2010.31
17. Guo P, Xue Z (2018) An adaptive PSO-based real-time workflow scheduling algorithm in cloud systems. In: International conference on communication technology proceedings, ICCT, 2017-October, pp 1932–1936. https://doi.org/10.1109/ICCT.2017.8359966
18. Awadalla M, Elewi A (2016) Enhanced PSO approach for real time systems scheduling. Int J Comput Theory Eng 8(4):285–289. https://doi.org/10.7763/ijcte.2016.v8.1059
19. Rahman HF, Janardhanan MN, Nielsen IE (2019) Real-time order acceptance and scheduling problems in a flow shop environment using hybrid Ga-PSO algorithm. IEEE Access 7:112742–112755. https://doi.org/10.1109/ACCESS.2019.2935375
20. Eberhart R, Kennedy J (1995) New optimizer using particle swarm theory. In: Proceedings of the international symposium on micro machine and human science. https://doi.org/10.1109/mhs.1995.494215
21. Brownlee J (2011) Clever algorithms. Search. https://doi.org/10.1017/CBO9781107415324.004
22. Elbes M, Alzubi S, Kanan T, Al-Fuqaha A, Hawashin B (2019) A survey on particle swarm optimization with emphasis on engineering and network applications. Evol Intell 12(2):113–129. https://doi.org/10.1007/s12065-019-00210-z
23. Dixit A, Mani A, Bansal R (2021) An adaptive mutation strategy for differential evolution algorithm based on particle swarm optimization. Evol Intell. https://doi.org/10.1007/s12065-021-00568-z
24. Li YL, Shao W, You L, Wang BZ (2013) An improved PSO algorithm and its application to UWB antenna design. IEEE Antennas Wirel Propag Lett 12(3):1236–1239. https://doi.org/10.1109/LAWP.2013.2283375
25. Erskine A, Joyce T, Herrmann JM (2017) Stochastic stability of particle swarm optimisation. Swarm Intell 11(3–4):295–315. https://doi.org/10.1007/s11721-017-0144-7
26. Teraiya J, Shah A, Kotecha K (2019) ACO based scheduling method for soft RTOS with simulation and mathematical proofs. Int J Innov Technol Explor Eng 8(12):4736–4740. https://doi.org/10.35940/ijitee.L3606.1081219
27. Shah A, Kotecha K (2010) Scheduling algorithm for real-time operating systems using ACO. In: Proceedings—2010 international conference on computational intelligence and communication networks, CICN 2010. https://doi.org/10.1109/CICN.2010.122
28. Lindh F, Otnes T, Wennerström J (2010) Scheduling algorithms for real-time systems. Department of Computer Engineering, Malardalens University, Sweden. Retrieved from http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Scheduling+algorithms+for+real-time+systems#0
29. Yang K, Anderson JH (2015) On the soft real-time optimality of global EDF on multiprocessors: from identical to uniform heterogeneous. In: Proceedings—IEEE 21st international conference on embedded and real-time computing systems and applications, RTCSA 2015, pp 1–10. https://doi.org/10.1109/RTCSA.2015.14