4 CRITICAL ANALYSIS OF STATIC AND DYNAMIC SCHEDULING ALGORITHMS

In Soft Real-Time System, a suitable scheduling algorithm needs to select grounded on the characteristics of the System and the process type. Its task set classifies as a Periodic, Aperiodic, and Sporadic task. Task sets can be divided into preemptive and non-preemptive task sets. The task has different characteristics like its execution time, arrival time, deadline, and resource requirements. The scheduling algorithm can be divided into two categories, static and dynamic, which depend on the priority they follow. The Earliest Deadline First (EDF) and Least Slack Time first (LST) are dynamic scheduling algorithms in real-time systems. It chooses the priority of the processes grounded on deadline and slack time correspondingly. The process, which has the shortest deadline and shortest slack time, will have more priority in EDF and LST. EDF and LST are more appropriate for scheduling processes in the soft real-time operating systems (RTOS) [3][26][45]. The rate monotonic (RM) and shortest job first (SJF) are static scheduling algorithms in real-time systems. It chooses the priority of the processes grounded on its occurrence and time required to execute for a given process correspondingly. The process which has the smallest period and shortest time needed to execute will be considered as more priority in RM and SJF [46] [47].

In this section, two dynamic scheduling algorithms (EDF and LST) and two static algorithms (RM and SJF) have been evaluated for the soft Real-Time System and did a critical analysis of these algorithms. Algorithms are tested with a periodic task set (describe in section 3), and

results are collected. It has observed the success ratio (SR) and effective CPU utilization (ECU) for all algorithms in a similar environment (describe in section 3) [48].

4.1 The Static Scheduling Algorithms

The static scheduling algorithm can calculate the order of execution before runtime as well. The static scheduling algorithm also decides the sequence of tasks based on priority, but the priority value will not change during runtime [30]. In this section, two static scheduling algorithms RM and SJF have been explained.

4.1.1 The Rate Monotonic (RM) Algorithm

The Rate Monotonic is a static preemptive scheduling algorithm. It gives priority to the task based on its Rate (task occurrence period). The task with the smallest Rate will get high priority [31][27][49]. The period of any task is predefined in Real-Time System and defined as the task occur again in a given duration. Figure 4.1 shows the flow of this algorithm. When the currently executing task is completed; or a new task comes, the scheduling algorithm will run and check the lowest rate of each active task. The task which has the lowest rate will be selected for the subsequent execution [10][50]. Due to this nature task having smallest rate value will get chance to execute first and have more chances to meet their deadline before the new occurrence of the same task come again.

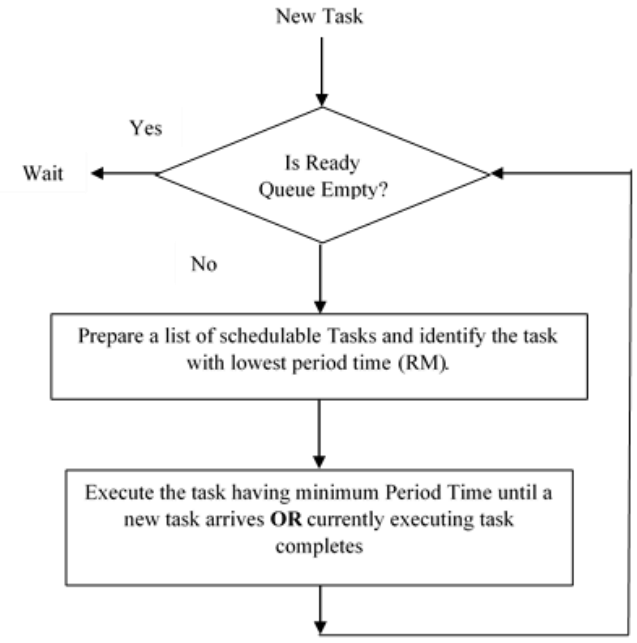

Figure 4.1 - Flow of the RM Algorithm

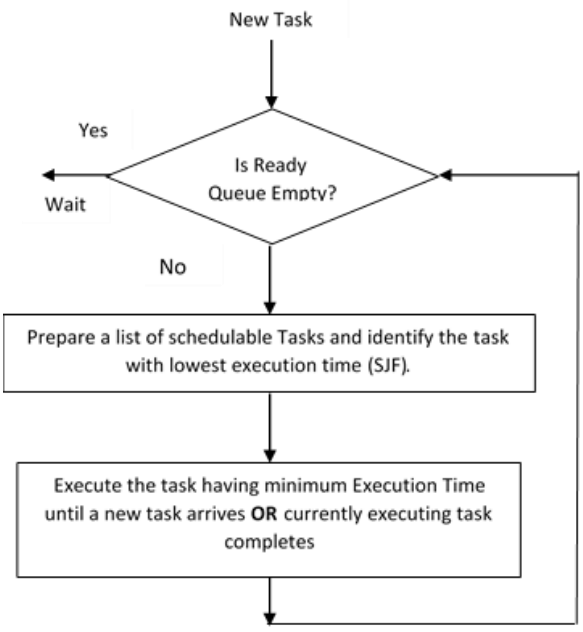

Figure 4.2 - Flow of the SJF Algorithm

4.1.2 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 a short execution time [31]. The execution time of a task is already known in the real-time system and is defined as the CPU time required for completing the task. The scheduling algorithm is necessary to execute when a currently running task completes or a new task arrives. The flowchart of the algorithm has been shown in Figure 4.2. 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 has a minimum execution time. Due to this nature task having smallest execution time will get chance to execute first and it make sure that small task will always meet their deadline and increase the overall benefits of the given system.

4.2 The Dynamic Scheduling Algorithms

Dynamic scheduling algorithms make decisions at the runtime. It allows to not only design a more flexible system but also associate calculation overhead with it. The dynamic scheduling algorithms decide what task to execute depending on the importance of the task, called priority. The task priority may change during the runtime [26][25]. In this section, two dynamic scheduling algorithms EDF and LST have been explained.

4.2.1 The Earliest Deadline First (EDF) Algorithm

The Earliest Deadline First (EDF) algorithm is a dynamic preemptive scheduling algorithm. 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 4.3 shows the flow of the EDF algorithm. When the currently executing task is completed; or a new task comes, the scheduling algorithm will run and check the absolute deadline of each active task. The task which has the earliest deadline will be selected for the subsequent execution.

4.2.2 The Least Slack Time First (LST) Algorithm

The Least Slack Time First algorithm is a dynamic preemptive scheduling algorithm. The highest priority is assigned to the task having the short slack time. The slack time l is defined as per the following equation [29][51].

$$l = d - c - t \qquad (4)$$

where,

t = current time

d = deadline

c = remaining execution time

The scheduling algorithm is necessary to execute when a currently running task completes or a new task arrives. The flowchart of the algorithm has been shown in Figure 4.4. 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 4. The new task selected for execution has minimum slack time.

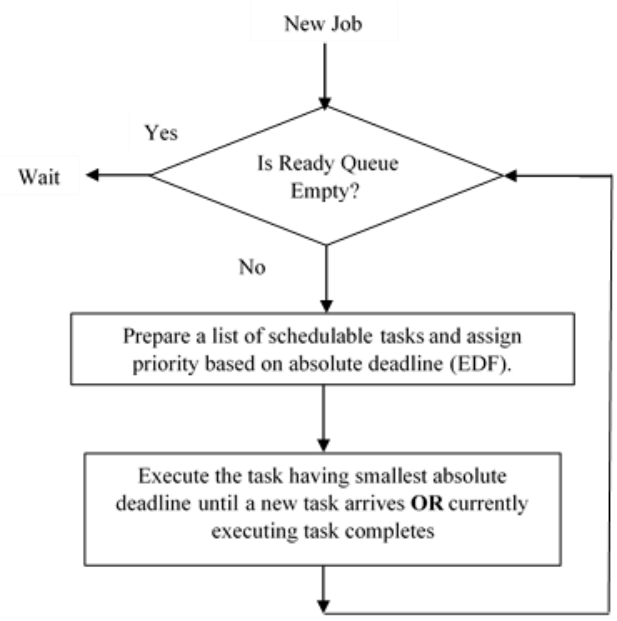

Figure 4.3 - Flow of the EDF Algorithm

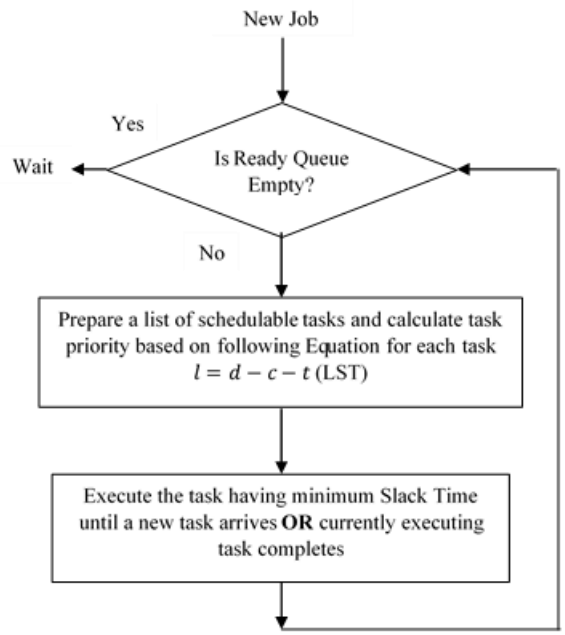

Figure 4.4 - Flow of the LST Algorithm

4.3 Performance Analysis and Result Comparison

All Dynamic and static scheduling algorithms have been implemented using the simulator, explained in Section 3.3. The algorithm executes when a new task arrives, or the current task completes its execution. All algorithms are tested with the periodic task set (Data Set) described in Section 3.2 to evaluate their performance. Load of the system (U_p) is calculated based on Equation 3 (Described in Section 3.2). All algorithms have been assessed with three major categories of U_p values which consider as underload, overload, and highly overload scenarios. If the value of $U_p \leq 1$, it is considered as underload scenario, if it is $1.0 < U_p \leq 1.5$, it considered an overload scenario, and if it is $1.5 < U_p \leq 5.0$, it is considered a highly overload scenario. The value of U_p varies between 0.5 to 5 for the entire Data Set. All algorithms have been tested on 500-time units to prove their effectiveness. Performance of all algorithms has been measured and evaluated concerning SR and ECU parameters which are explained in Section 3.1. Detailed performance analysis and result comparison has been given in this section.

4.3.1 Underload Scenario

All Static and Dynamic algorithms have been tested in the underload scenario. The scenario is considered an underload when the utilization factor for the task set is less than or equals one. Table 4.1 represents the scenario where the task set contains 1 to 9 tasks, and the utilization factor (CPU Load) is less than 1 or equal to 1 ($U_p \leq 1$). Results show that ECU values remain nearly the same for dynamic algorithms, whereas for static, it is slightly less. Results also indicate SR is not 100% in the case of static scheduling algorithms. When CPU Load is less than

one, it means that the task set is schedulable, and the scheduling algorithm can schedule all tasks, and all tasks can meet their deadline. Still, static algorithms cannot schedule all tasks, whereas dynamic scheduling algorithms can successfully schedule these task sets. It means in underload situation dynamic algorithms like EDF guarantee to schedule all task, and dynamic algorithm like LST performs batter then static scheduling algorithm like RM and SJF. So, it is advisable to use the characteristics of the dynamic scheduling algorithm in the underload scenario compare to the static scheduling algorithms. Figure 4.5 and 4.6 provides the performance comparison of static and dynamic scheduling algorithms in underload scenario concerning ECU and SR parameters.

Table 4.1 - Static and Dynamic scheduling algorithms performance in Underload Scenario

CPU 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

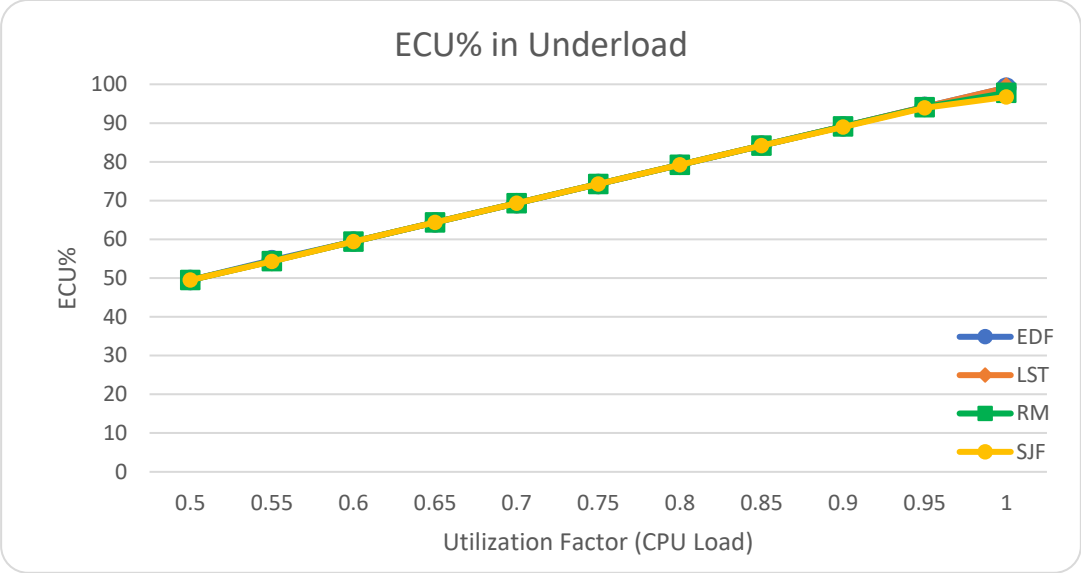

Figure 4.5 – ECU% Vs CPU Load for Static and Dynamic Scheduling Algorithms Underload Scenario

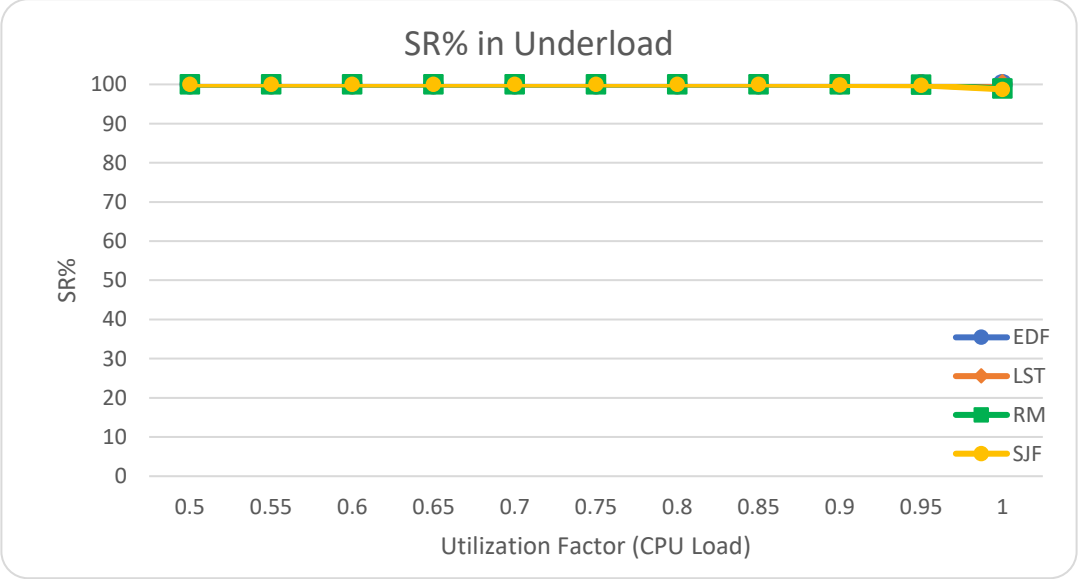

Figure 4.6 – SR% Vs CPU Load for Static and Dynamic Scheduling Algorithms Underload Scenario

4.3.2 Overload Scenario

All Static and Dynamic algorithms have been tested in the overload scenario. The scenario is considered as an overload when the utilization factor for the task set is $1.0 < U_p \leq 1.5$. Table 4.2 represents the scenario where task set contains 1 to 9 task, and utilization factor (CPU Load) vary between $1.0 < U_p \leq 1.5$. Results show a significant difference in ECU and SR values for static and dynamic scheduling algorithms. When the CPU load is greater than 1, the task set is not schedulable, and few tasks will miss their deadline. Table 4.2 observations reflect that dynamic scheduling algorithms performance degrades very poorly in a slightly overload situation, whereas static scheduling algorithms can meet the deadline for a few of their task sets. It means that in overload situations; static scheduling algorithms give better performance than dynamic scheduling algorithms. Figure 4.7 and 4.8 provides the performance comparison of static and dynamic scheduling algorithms in overload scenario concerning ECU and SR parameters.

Table 4.2 - Static and Dynamic scheduling algorithms performance in Overload Scenario

CPU 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

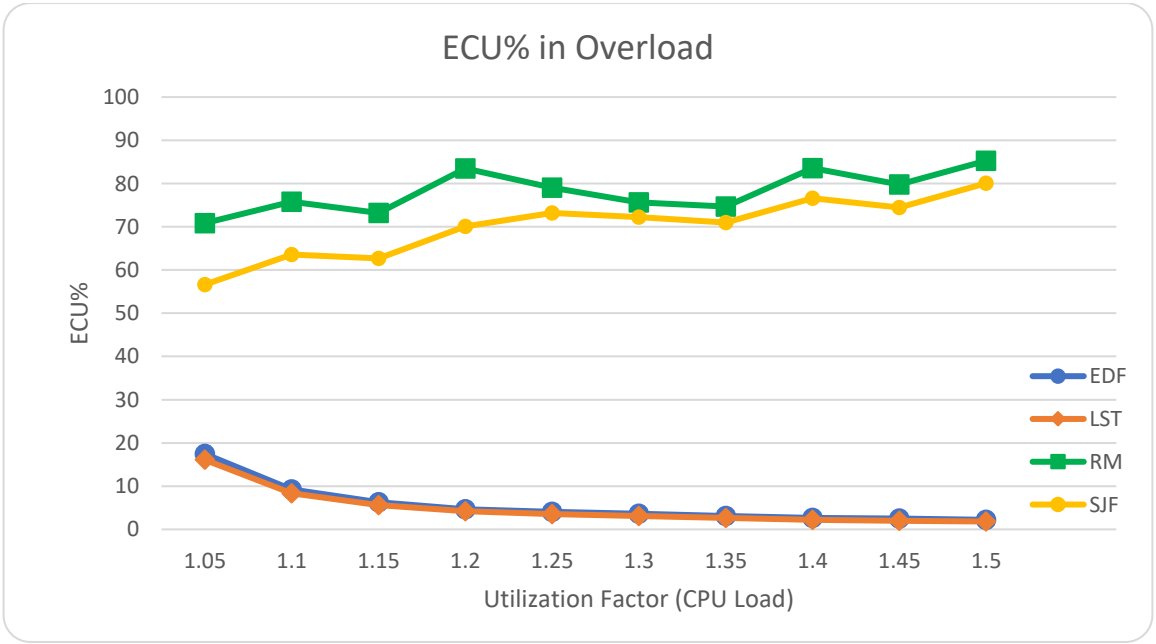

Figure 4.7 – ECU% Vs CPU Load for Static and Dynamic Scheduling Algorithms in Overload

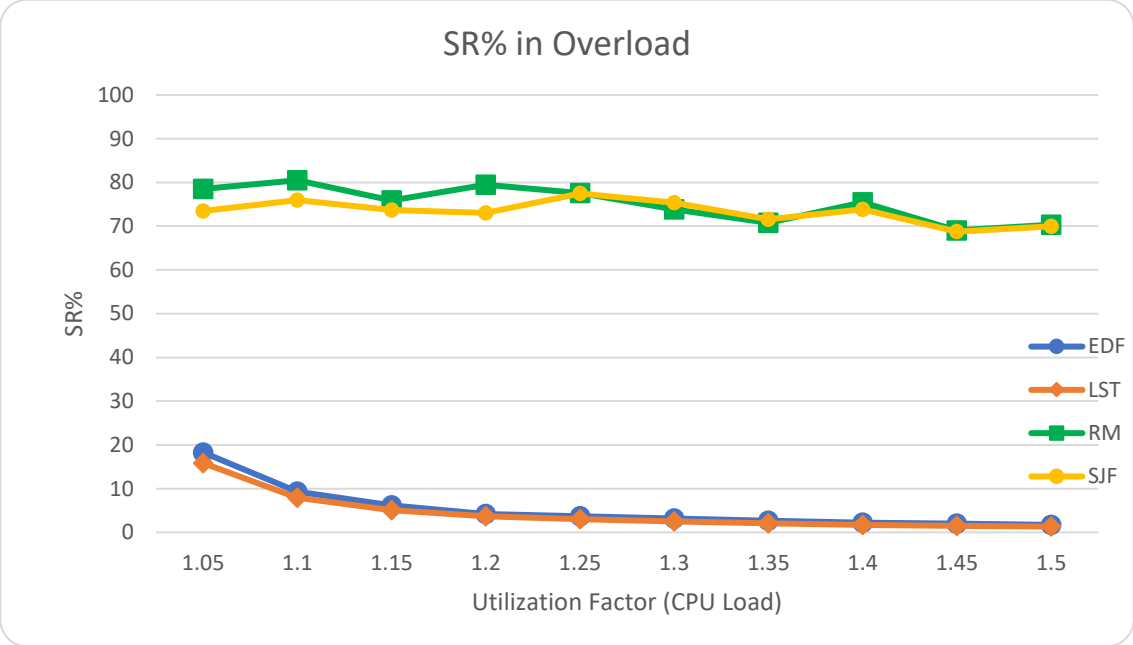

Figure 4.8 – SR% Vs CPU Load for Static and Dynamic Scheduling Algorithms in Overload Scenario

4.3.3 Highly Overload Scenario

All Static and Dynamic algorithms were also tested in the highly overload scenario. The scenario is considered a Highly Overload when the utilization factor for the task set is more than 1.5. Table 4.3 represents the scenario where the task set contains 1 to 9 tasks, and the utilization factor (CPU Load) is between 1.5 and 5 ($1.5 < U_p \leq 5.0$). Results show that ECU and SR values are very low for the dynamic scheduling algorithms because most of the task sets are missing their deadlines. When the CPU load is more than one, the task set is not schedulable, and no scheduling algorithm can schedule all tasks. There will be tasks in the task set that will miss

their deadline, but static algorithms can still schedule some of the tasks that meet their deadline, whereas dynamic scheduling algorithms fail. In highly overload situations, static algorithms like RM and SJF perform well compared to dynamic scheduling algorithms like EDF and LST. So, it is advisable to use the characteristics of static scheduling algorithms in highly overload scenarios compare to the dynamic scheduling algorithms. Figure 4.9 and 4.10 provides the performance comparison of static and dynamic scheduling algorithms in highly overload scenario concerning ECU and SR parameters.

Table 4.3 - Static and Dynamic scheduling algorithms performance in Highly Overload Scenario

CPU Load	**ECU%**				**SR%**			
	Dynamic		**Static**		**Dynamic**		**Static**	
	EDF	**LST**	**RM**	**SJF**	**EDF**	**LST**	**RM**	**SJF**
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

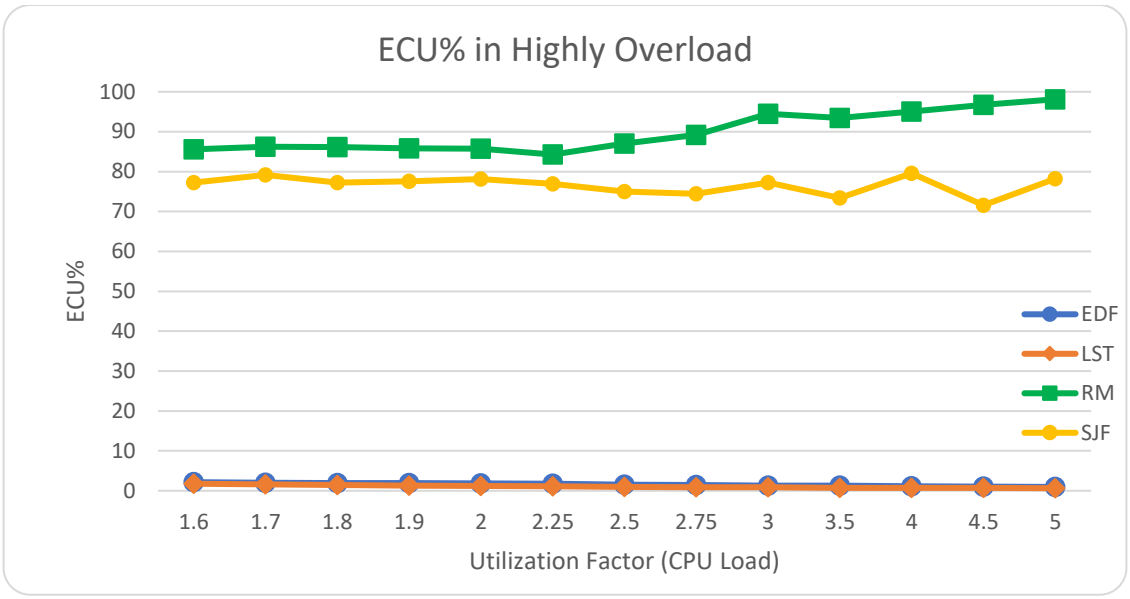

Figure 4.9 – ECU% Vs CPU Load for Static and Dynamic Scheduling Algorithms in Highly Overload Scenario

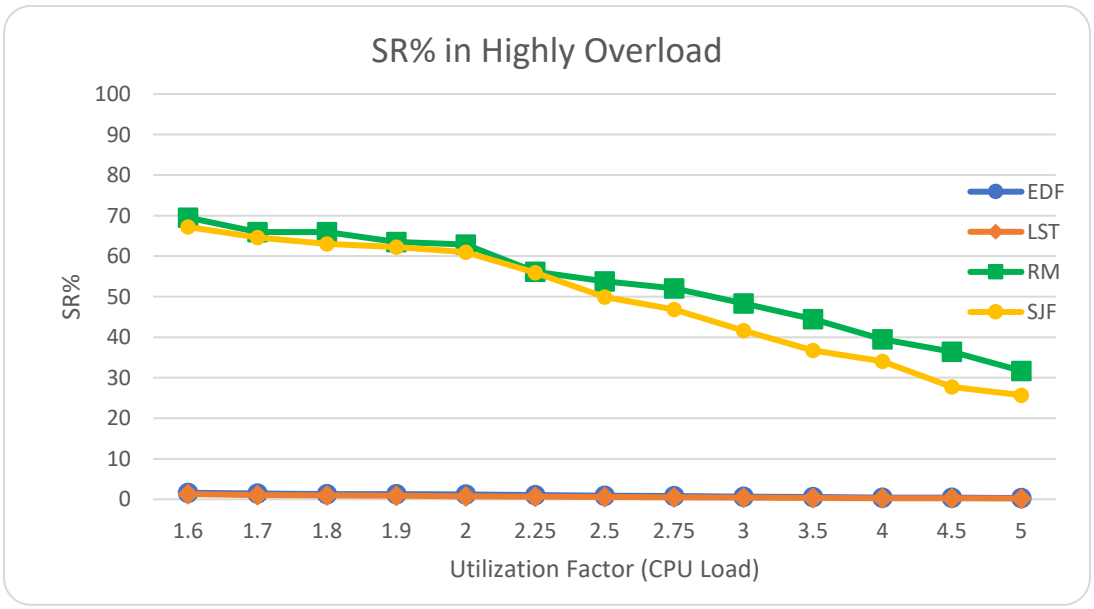

Figure 4.10 – SR% Vs CPU Load for Static and Dynamic Scheduling Algorithms in Highly Overload Scenario

4.4 Conclusion

The Dynamic and Static algorithms are evaluated for the Soft Real-Time System and considering it for a single processor and pre-emptive task set in this research. It is also believed that the process set is periodic. All algorithms are evaluated in a similar environment, and results have been observed and equated. Observation suggests that dynamic algorithms like EDF and LST performs well in underload situation and able to schedule most of all task when $U_p \leq 1$. In overload and highly overload scenarios performance of the dynamic scheduling algorithms starts decreasing rapidly. So, in an underload scenario, dynamic scheduling algorithms are advisable but not with an overload situation. Static algorithms perform moderately to underload situations. It has been observed that with a specific task set, even it is possible that all tasks can be scheduled, but static scheduling algorithms are failed to schedule it. So, in an underload, static scheduling algorithms are not advisable. But in an overload and highly overload scenario, it performs well compared to dynamic scheduling. This happens because static algorithms select the task so that more tasks meet their deadline in overload and highly overload scenario, and the system gets maximum profit. Because of that, ECU% and SR% are good compared to dynamic scheduling algorithms in an overload and highly overload scenario. situation. The ideal algorithm can be designed, which uses Dynamic and Static algorithm features, and it performs well in underload and overload scenarios.