5 HYBRID SCHEDULING ALGORITHM (S_LST)

In the Soft Real-Time System, scheduling task with the processor is a critical activity. The system schedules the tasks on a processor in a time interval, and hence the tasks get a chance to execute on the processor. Priority-driven scheduling algorithms are sub-categorized into mainly two categories called Static Priority and Dynamic Priority. Critical analysis of static and dynamic priority scheduling algorithms has been discussed in the previous section. The previous section 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 algorithms have been analyzed with preemptive task sets, and it has been considered that all the task sets are periodic [52].

This section proposes 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 processes meet their deadline. However, in an overload and highly overload condition, the performance of dynamic priority algorithms reduces quickly, and most of the tasks will miss their deadline. In contrast, static priority scheduling algorithms are still able to meet few deadlines. This section proposes one Hybrid algorithm called 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. Success Ratio (SR) & Effective CPU Utilization (ECU) have been observed and compared all algorithms in the same conditions. It is noted that instead of using LST and SJF as an independent algorithm, the Hybrid algorithm S_LST performs well in an underload, overload, and highly overload scenario. The

hybrid algorithm called S_LST, which uses the LST and SJF scheduling algorithm, is described below [46][15].

5.1 S_LST Algorithm

S_LST algorithm uses the characteristics of LST and SJF. In underload, task priority will be given based on slack time, and in an overload and highly overload scenario, task priority will be assigned based on the shortest execution time. It has been considered that the execution time of the task, its arrival time, its period, and total CPU load are 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: Task Set

Output: MITask to be executed

1: **if** (Underload Scenario)

2: **for** each task in a task set

3: Calculate Slack time for each task in task Set

4: Select MITask with the lowest slack time

5: **end for**

6: **else**

7: **for** each task in a task set

8: Calculate Shortest Execution Time for each task

9: Select MITask with the lowest Execution time

10: **end for**

11: **end if**

12: **return** MITask

As shown in the Algorithm, when scheduling algorithm invokes; first, it observed the CPU load, based on the current task set, and available tasks 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 the shortest execution time (Static Scheduling algorithm). The static scheduling algorithm aims to gain maximum profit from the given task set. So, the SJF algorithm gets more tasks that meet their deadline in an overload and highly overload situation where the dynamic scheduling algorithm performs poorly [53].

5.2 Performance Analysis and Result Comparison

S_LST, LST, and SJF 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 $1.5 < U_p \leq 5.0$ 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.

5.2.1 Underload Scenario

LST, SJF, and S_LST 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 5.1 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 identical for all these algorithms, whereas, for SJF, it is slightly less. Results also indicate SR is not 100% in the case of the SJF scheduling algorithm. 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, the SJF algorithm cannot schedule all tasks, whereas LST and S_LST scheduling algorithms can successfully schedule these task sets. In an underload situation, the S_LST algorithm performs equally to the LST algorithm and performs better than the SJF algorithm. Figure 5.1 and 5.2 provides the performance comparison of LST, SJF and S_LST scheduling algorithms in underload scenario concerning ECU and SR parameters.

Table 5.1 – LST, SJF, and S_LST scheduling algorithms performance in Underload Scenario

CPU Load	ECU%			SR%		
	LST	SJF	S_LST	LST	SJF	S_LST
0.50	49.49	49.49	49.49	100.00	100.00	100.00
0.55	54.40	54.31	54.41	100.00	100.00	100.00
0.60	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
0.70	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
0.80	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
0.90	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
1.00	99.10	96.74	99.10	100.00	98.74	100.00

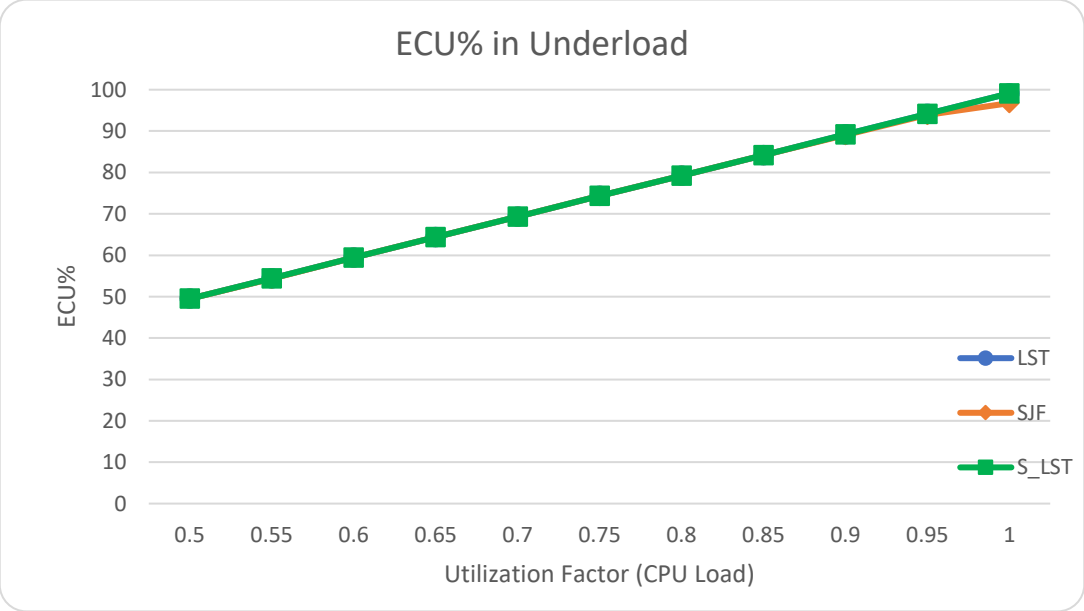

Figure 5.1 – ECU% Vs CPU Load for LST, SJF and S_LST Scheduling Algorithms in Underload Scenario

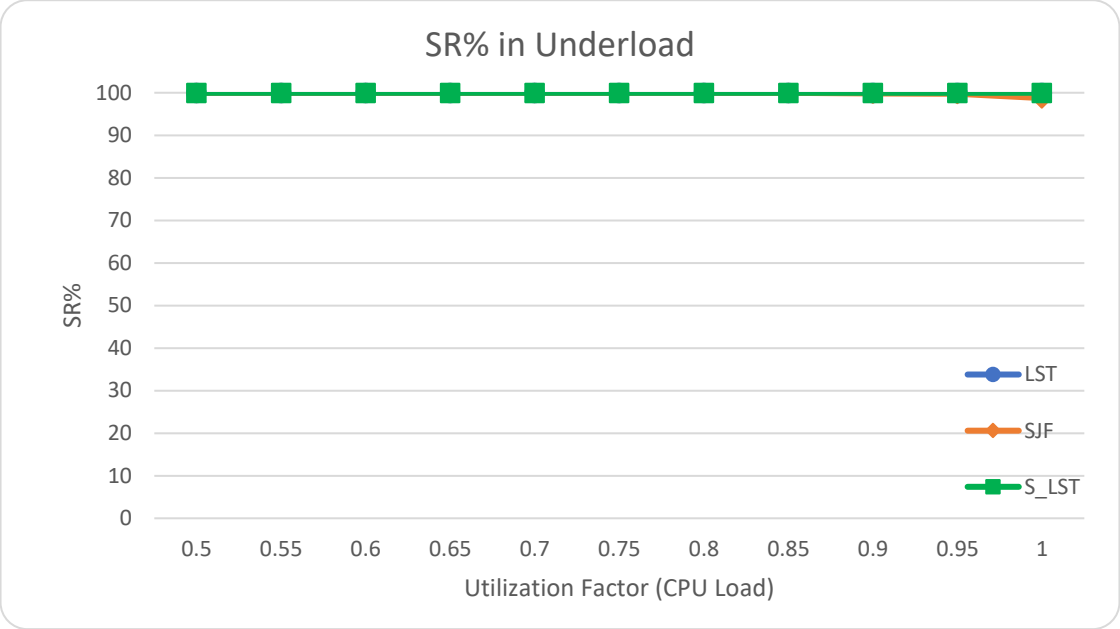

Figure 5.2 – SR% Vs CPU Load for LST, SJF and S_LST Scheduling Algorithms in Underload Scenario

5.2.2 Overload Scenario

LST, SJF, and S_LST 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 5.2 represents the scenario where the task set contains 1 to 9 tasks, 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 LST, SJF, and S_LST scheduling algorithms. When the CPU load is greater than 1, the task set is not schedulable, and few tasks will miss their deadline. Table 5.2 observations reflect that in a slightly overload situation, LST performance degrades very poorly, whereas the S_LST scheduling algorithm can meet the deadline for a few of their task set. It means in an overload situation S_LST scheduling algorithm gives better performance than the LST

scheduling algorithm. Figure 5.3 and 5.4 provides the performance comparison of LST, SJF and S_LST scheduling algorithms in overload scenario concerning ECU and SR parameters.

Table 5.2 – LST, SJF, and S_LST scheduling algorithms performance in Overload Scenario

CPU Load	ECU%			SR%		
	LST	**SJF**	**S_LST**	**LST**	**SJF**	**S_LST**
1.05	16.09	56.63	56.29	15.84	73.49	71.22
1.10	8.33	63.60	56.45	7.90	75.98	67.82
1.15	5.58	62.66	55.32	5.06	73.66	65.79
1.20	4.21	70.08	48.09	3.67	73.06	52.84
1.25	3.56	73.20	49.65	3.06	77.47	56.21
1.30	3.09	72.24	51.96	2.53	75.34	57.82
1.35	2.63	70.99	53.55	2.09	71.55	55.93
1.40	2.20	76.57	45.37	1.71	73.80	46.76
1.45	2.01	74.45	48.39	1.52	68.76	47.74
1.50	1.83	80.07	43.77	1.33	69.96	42.10

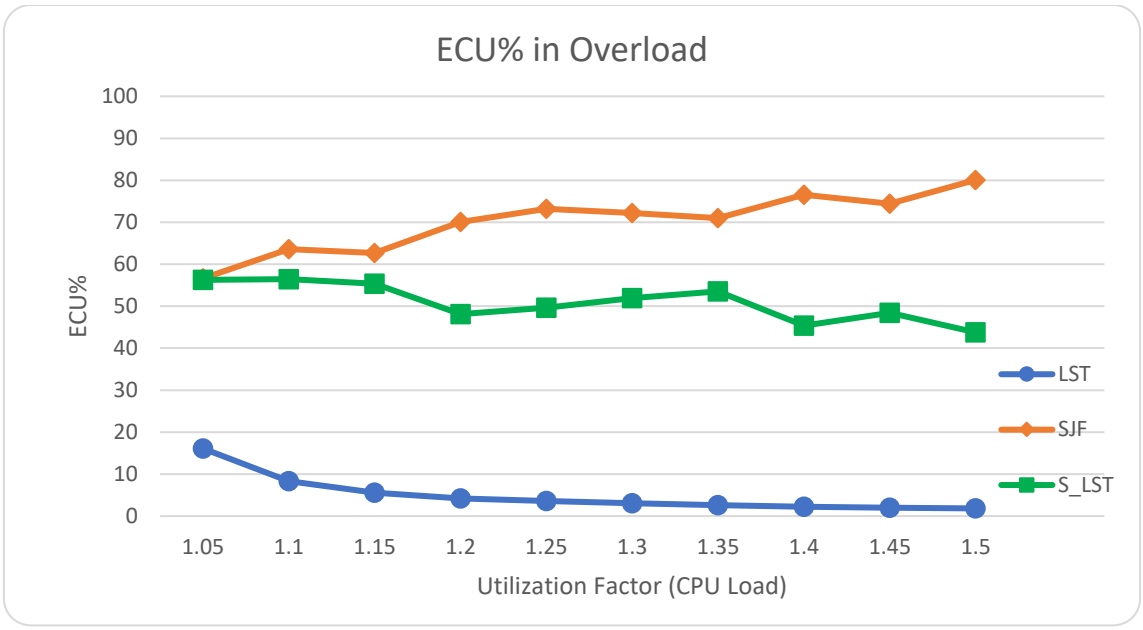

Figure 5.3 – ECU% Vs CPU Load for LST, SJF and S_LST Scheduling Algorithms in Overload Scenario

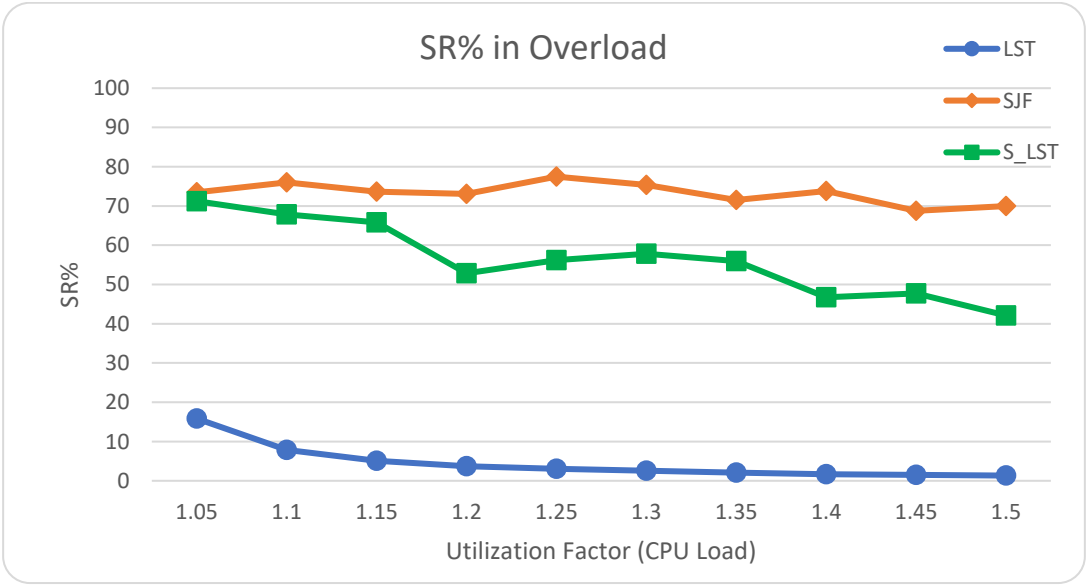

Figure 5.4 – SR% Vs CPU Load for LST, SJF and S_LST Scheduling Algorithms in Overload Scenario

5.2.3 Highly Overload Scenario

LST, SJF, and S_LST algorithms were also tested in the highly overload scenario. The scenario is considered as a Highly Overload when the utilization factor for the task set is more than 1.5. Table 5.3 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 LST scheduling algorithm because most task sets miss their deadlines. When the CPU load is more than one, it means that 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 S_LST uses the characteristics of the SJF algorithm during overload scenarios and can schedule some of the tasks that meet their deadline. It means that in highly overload situations S_LST algorithm performs well compared to the LST scheduling algorithm. Figure 5.5 and 5.6 provides the performance comparison of LST, SJF and S_LST scheduling algorithms in highly overload scenario concerning ECU and SR parameters.

Table 5.3 – LST, SJF, and S_LST scheduling algorithms performance in Highly Overload Scenario

CPU Load	ECU%			SR%		
	LST	SJF	S_LST	LST	SJF	S_LST
1.60	1.77	77.26	50.71	1.29	67.20	47.22
1.70	1.58	79.16	47.45	1.07	64.60	42.72
1.80	1.45	77.28	47.46	0.95	63.04	42.14
1.90	1.31	77.53	48.39	0.85	62.21	41.83
2.00	1.19	78.10	42.82	0.76	61.00	35.30
2.25	1.13	76.95	45.85	0.65	55.91	35.86
2.50	0.98	74.97	44.51	0.54	49.92	31.05
2.75	0.91	74.42	38.06	0.47	46.83	25.28
3.00	0.86	77.23	31.51	0.40	41.67	18.37
3.50	0.75	73.37	37.33	0.33	36.76	19.05
4.00	0.73	79.57	28.03	0.27	34.09	12.59
4.50	0.71	71.58	27.99	0.24	27.74	11.47
5.00	0.66	78.22	20.16	0.20	25.71	07.09

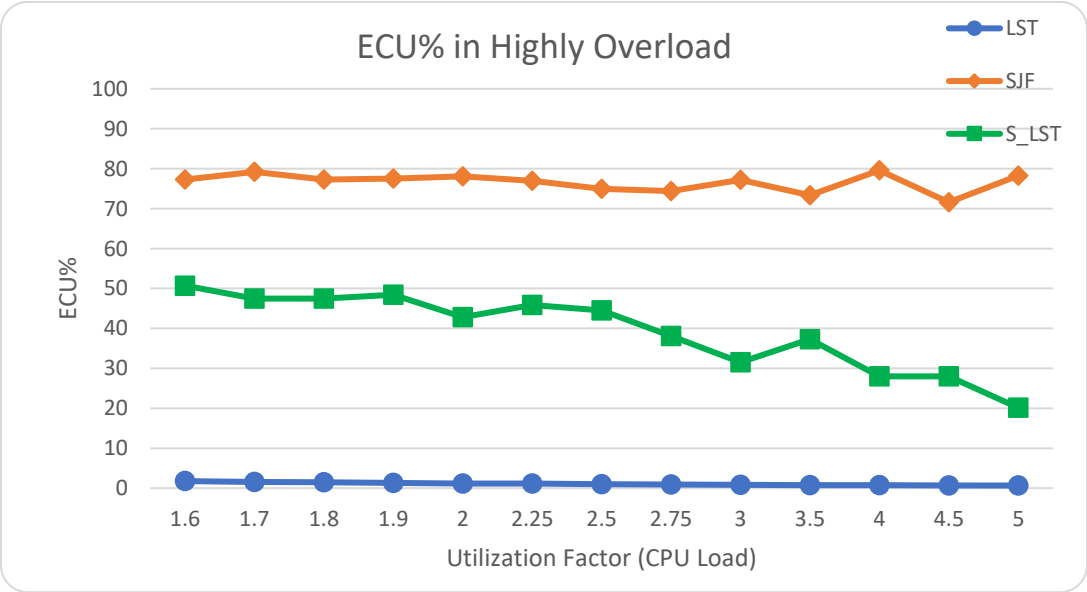

Figure 5.5 – ECU% Vs CPU Load for LST, SJF and S_LST Scheduling Algorithms in Highly Overload Scenario

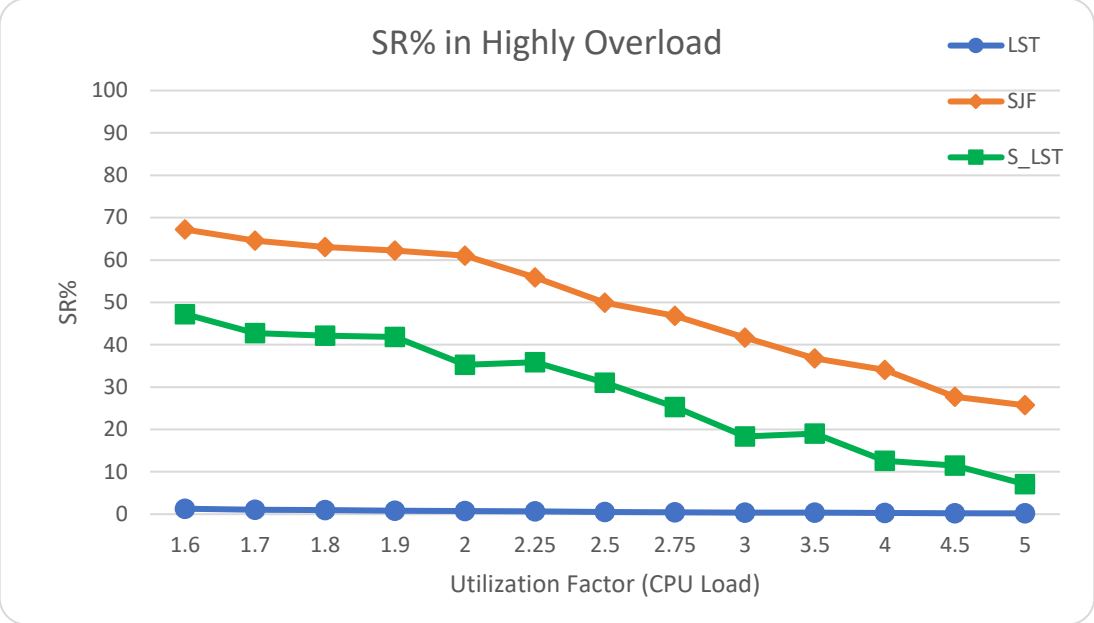

Figure 5.6 – SR% Vs CPU Load for LST, SJF and S_LST Scheduling Algorithms in Highly Overload Scenario

5.3 Conclusion

The Static and Dynamic Algorithms are implemented to schedule a soft real-time system with a single processor and pre-emptive task sets and a critical analysis of these algorithms with ECU and SR parameters in the previous section. These algorithms are simulated with periodic task sets; results are obtained and compared. Observation says that dynamic algorithms perform well in underload situations and guarantee to meet all the deadlines of a given process set. In overload and highly overload situations, dynamic algorithms performance degrades very poorly. So, in an underload, dynamic algorithms are advisable but not with an overload situation. Static algorithms miss few deadlines, even in underload situations. It has been observed that with the specific task set, even it is possible that all task can meet their deadline, but static algorithms are failed to

schedule it. So, static scheduling algorithms are not advisable in an underload, but in overload, they perform well compared to dynamic algorithms. Based on this observation, thesis proposed a hybrid approach for efficient scheduling in a Soft Real-Time system called S_LST. S_LST algorithm uses the characteristics of LST in the underload scenario, and it uses the characteristics of the SJF algorithm in overload and highly overload scenarios. Because of this, the S_LST algorithm performs better in all situations compare to a single approach. Developing a scheduling algorithm using Swarm Intelligence (ACO) has been done for the Soft Real-Time system. There are still multiple research possibilities where swarm intelligence techniques like Gravitational Search Algorithm (GSA) or Particle Swarm Optimization (PSO) can be used and can design an efficient scheduling algorithm that can perform well underload and overload situation.