A COMPREHENSIVE REVIEW AND SIMULATION-BASED EVALUATION OF REAL-TIME SCHEDULING ALGORITHMS IN EMBEDDED SYSTEMS

Abdusamea I. A. Omer
Department of Computer Engineering and Information Technology, Faculty of Engineering, Sabratha University, Sabratha, Libya
Email: abdusamea@gmail.com

Maryam Alaa Zaki
Department of Computer Engineering and Sciences, Faculty of Science, University of Zawia, Zawia, Libya

Haneen Jama Alnakoua
Department of Computer Engineering and Sciences, Faculty of Science, University of Zawia, Zawia, Libya

Abstract

Real-time scheduling is a fundamental requirement in embedded systems, where system correctness depends not only on logical accuracy but also on meeting strict temporal constraints. Selecting an appropriate scheduling algorithm is critical to ensuring predictability, responsiveness, and efficient resource management. This paper presents a comprehensive review and practical evaluation of four classical real-time scheduling algorithms: Rate Monotonic Scheduling (RMS), Deadline Monotonic Scheduling (DMS), Earliest Deadline First (EDF), and Least Laxity First (LLF). A custom simulation framework was developed in Python to evaluate these algorithms under various workload scenarios (low to overload) and diverse task-set sizes (5, 10, and 20 tasks). Performance was assessed using metrics including deadline miss ratio, average response time, and scheduling overhead (quantified by context switches). Results demonstrate that while all algorithms perform reliably under normal conditions, significant divergence occurs under overload. RMS consistently achieved the lowest response times and overhead, whereas dynamic-priority algorithms like LLF suffered from performance degradation due to high scheduling complexity. These findings offer practical guidance for selecting scheduling strategies in resource-constrained embedded environments.

Keywords: Real-time systems, Embedded systems, Scheduling algorithms, Response time, Context switches.

1. Introduction

1.1 Background and Importance

Embedded systems have become an integral part of modern technological infrastructures and are widely deployed in domains such as industrial automation, automotive systems, healthcare devices, aerospace applications, and Internet of Things (IoT) platforms [1], [2]. Many of these applications operate under strict timing constraints, where the correctness of

system behavior depends not only on the logical accuracy of computations but also on the timely completion of tasks [3], [1].

Real-time systems are designed to satisfy these timing requirements by ensuring that computational tasks complete before their specified deadlines. Depending on the consequences of deadline violations, real-time systems are generally classified into hard, firm, and soft real-time systems [1]. In hard real-time applications, missing a deadline may lead to severe failures, making predictable scheduling mechanisms essential [3], [1].

1.2 Real-Time Scheduling in Embedded Systems

Task scheduling is a fundamental component of real-time operating systems (RTOSs) and determines the order in which tasks access processor resources [1], [2]. Effective scheduling directly influences system predictability, responsiveness, resource utilization, and overall performance [4].

Real-time scheduling algorithms are commonly categorized into fixed-priority and dynamic-priority approaches [3], [1]. Fixed-priority techniques, such as Rate Monotonic Scheduling (RMS) and Deadline Monotonic Scheduling (DMS), assign priorities before runtime and maintain them throughout task execution. In contrast, dynamic-priority algorithms, including Earliest Deadline First (EDF) and Least Laxity First (LLF), adjust priorities during execution according to task characteristics and timing requirements [3], [1].

Each scheduling approach offers distinct advantages and limitations in terms of implementation complexity, schedulability, response time, and scheduling overhead [5], [6].

1.3 Research Problem

Although real-time scheduling has been extensively studied, selecting an optimal scheduling policy remains a non-trivial challenge due to the trade-offs between computational overhead and schedulability [5], [7]. Modern embedded systems exhibit diverse workload characteristics, increasing task complexity, and limited computational resources, making it difficult to determine which scheduling technique offers the best performance under different operating conditions [8], [9].

Furthermore, many existing studies focus primarily on theoretical schedulability analysis or application-specific scenarios, while practical comparative evaluations considering multiple workload levels and varying task-set sizes remain relatively limited [5], [6].

1.4 Research Objectives and Contributions

This study aims to provide a comprehensive review and practical evaluation of classical real-time scheduling techniques in embedded systems. Specifically, the objectives of this work are as follows:

- To review the principles and characteristics of RMS, DMS, EDF, and LLF scheduling algorithms.
- To develop a simulation-based framework for evaluating scheduling performance.
- To analyze scheduling behavior under different workload conditions, including low load, medium load, high load, and overload scenarios.

- To investigate the impact of task-set size on scheduling performance and scalability.
- To evaluate scheduling algorithms using multiple performance metrics, including deadline miss ratio, average response time, scheduling efficiency, and context switches.

The main contributions of this paper can be summarized as follows:

1. A comprehensive review of classical real-time scheduling algorithms for embedded systems.
2. The development of a unified simulation framework for comparative scheduling analysis.
3. An experimental evaluation of four scheduling algorithms under multiple workload conditions.
4. A scalability analysis based on varying numbers of periodic tasks.
5. Practical recommendations for selecting appropriate scheduling techniques according to application requirements.

1.5 Literature Review

1.5.1 Foundations of Real-Time Scheduling

Real-time scheduling has been a fundamental research area in embedded systems for several decades due to its critical role in ensuring temporal correctness. In real-time systems, the correctness of an application depends not only on the logical accuracy of computations but also on the completion of tasks within specified timing constraints [3], [1].

The pioneering work of Liu and Layland [3] established the theoretical foundation of real-time scheduling by introducing Rate Monotonic Scheduling (RMS) and Earliest Deadline First (EDF). Their study demonstrated that RMS is an optimal fixed-priority algorithm for independent periodic tasks with deadlines equal to their periods, while EDF is optimal among dynamic-priority scheduling algorithms in uniprocessor environments.

Subsequent studies expanded these concepts to address increasingly complex real-time applications. Buttazzo [1] provided a comprehensive analysis of real-time scheduling algorithms and discussed their theoretical properties, implementation challenges, and practical applications in embedded systems. Similarly, Leung [4] presented an extensive overview of scheduling models, performance metrics, and optimization techniques used in real-time computing.

1.5.2 Fixed-Priority Scheduling Techniques

Fixed-priority scheduling algorithms assign task priorities before runtime and maintain these priorities throughout task execution [1]. Due to their simplicity and predictability, these algorithms are widely implemented in commercial real-time operating systems [2].

Rate Monotonic Scheduling (RMS) assigns priorities according to task periods, giving higher priorities to tasks with shorter periods [3]. Deadline Monotonic Scheduling (DMS) extends this approach by assigning priorities based on relative deadlines rather than periods, making it more suitable for task sets with constrained deadlines [1].

Fixed-priority scheduling techniques offer low runtime overhead and straightforward implementation. However, their schedulability is limited by utilization bounds and task characteristics [3], [1].

1.5.3 Dynamic-Priority Scheduling Techniques

Dynamic-priority algorithms determine task priorities during execution according to runtime information and timing constraints [1]. These approaches generally achieve higher processor utilization than fixed-priority techniques but may introduce additional scheduling overhead [6].

Earliest Deadline First (EDF) dynamically assigns the highest priority to the task with the nearest absolute deadline [3]. Under ideal assumptions, EDF can fully utilize processor capacity and provides optimal schedulability for independent periodic tasks in uniprocessor systems [3], [1].

Least Laxity First (LLF) assigns priorities according to task laxity, which represents the remaining time before a deadline after accounting for the remaining execution time [1]. Although LLF can achieve high schedulability, its frequent priority updates may increase the number of context switches, resulting in higher scheduling overhead and implementation complexity [1].

1.5.4 Recent Advances in Real-Time Scheduling

Recent developments in embedded and cyber-physical systems have introduced new scheduling challenges associated with increasing system complexity, heterogeneous architectures, and dynamic workloads [5], [6].

Akesson et al. [5] conducted a comprehensive survey of industrial practices in real-time systems and identified scalability, predictability, and resource efficiency as key challenges in modern embedded applications. Nelissen et al. [6] reviewed recent developments in response-time analysis and highlighted the growing importance of considering heterogeneous computing platforms and workload variability. Baruah and Buttazzo [7] discussed current trends in schedulability analysis and emphasized the need for practical evaluation frameworks capable of assessing scheduling algorithms under diverse operating conditions. Lakshmanan et al. [8] investigated task scheduling and synchronization in distributed embedded systems, demonstrating the impact of communication and resource-sharing overhead on system performance.

Recent studies have also explored scheduling challenges in emerging domains such as Internet of Things (IoT) systems, weakly hard real-time systems, and cyber-physical systems [10], [11].

1.5.5 Research Gap

Despite significant progress in real-time scheduling research, several limitations remain. Most existing studies focus either on theoretical schedulability analysis or on application-specific scenarios [5], [6]. Moreover, many studies evaluate scheduling algorithms using a limited set of performance metrics or under a single workload condition.

Relatively few studies provide a comprehensive practical comparison of multiple scheduling algorithms under varying workload levels and different task-set sizes within a unified simulation framework [5], [7]. Furthermore, context switches, scalability, and response-time performance are often evaluated independently rather than through an integrated experimental approach [9], [12].

1.5.6 Research Motivation

To address these limitations, this study presents a comprehensive review and practical evaluation of four classical scheduling algorithms: RMS, DMS, EDF, and LLF. A simulation-based framework is developed to analyze scheduling performance under low-load, medium-load, high-load, and overload conditions.

Unlike previous studies, the proposed framework simultaneously evaluates multiple performance metrics, including deadline miss ratio, average response time, and context switches. In addition, the scalability of the evaluated algorithms is investigated by examining the impact of increasing task-set sizes on scheduling behavior. The findings of this study aim to provide practical insights into the strengths and limitations of classical real-time scheduling techniques and support the selection of appropriate scheduling strategies for embedded systems.

1.6 Paper Organization

The remainder of this paper is organized as follows. Section 2 describes the methodology, including the simulation framework, implementation details, and experimental setup. Section 3 presents the results and discussion. Finally, Section 4 concludes the paper and suggests directions for future research.

2. Methodology

2.1 Research Framework

This study adopts a simulation-based experimental approach to evaluate the performance of classical real-time scheduling algorithms in embedded systems. The proposed framework aims to provide a unified environment for comparing different scheduling techniques under varying workload conditions and task-set sizes.

Four widely used scheduling algorithms were selected for evaluation: Rate Monotonic Scheduling (RMS), Deadline Monotonic Scheduling (DMS), Earliest Deadline First (EDF), and Least Laxity First (LLF). These algorithms were chosen because they represent both fixed-priority and dynamic-priority scheduling approaches commonly employed in real-time embedded systems [3], [1]. The evaluation process consists of task generation, job creation, scheduling simulation, performance measurement, and result analysis.

Figure 1 illustrates the overall workflow of the proposed simulation-based framework. The workflow was designed by the authors based on the experimental methodology adopted in this study. The framework begins with task-set generation and hyperperiod calculation, followed by job generation and scheduling execution using the selected algorithm.

Subsequently, performance metrics are computed, and the experimental results are statistically analyzed and visualized.

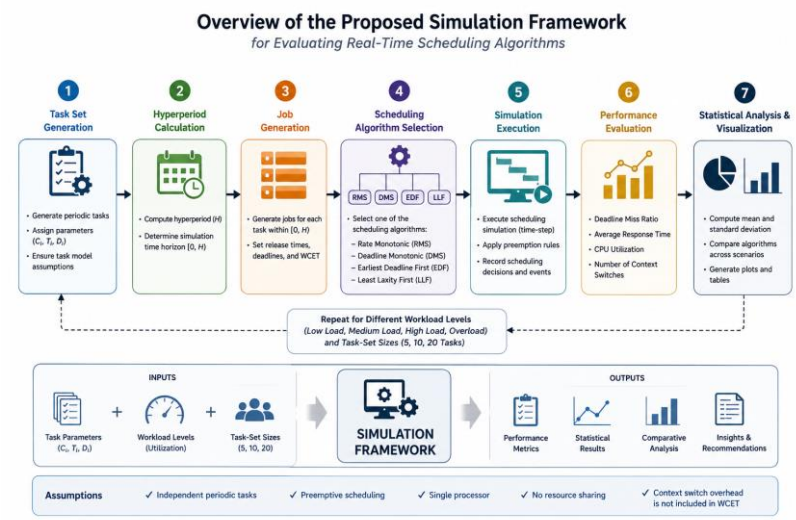


Figure 1. Overview of the proposed simulation-based framework for evaluating real-time scheduling algorithms in embedded systems

2.2 Task Model and System Assumptions

The experimental framework considers a set of independent periodic tasks executing on a single-core processor. The adopted task model follows the classical periodic task model introduced by Liu and Layland [3] and further discussed by Buttazzo [1]. Each task τ_i is characterized by the tuple:

$$\tau_i = (C_i, T_i, D_i) \quad (1)$$

where C_i represents the worst-case execution time (WCET), T_i denotes the task period, and D_i corresponds to the relative deadline.

The following assumptions were adopted throughout the experiments to ensure a controlled and reproducible evaluation:

- **Independent periodic tasks:** each task is invoked at a constant rate with no dependencies between tasks.
- **Single-processor environment:** all tasks are executed on a single processing unit.
- **Fully preemptive scheduling:** a higher-priority task can interrupt a lower-priority task at any time.
- **No shared resources:** tasks do not compete for mutual exclusion resources, eliminating the risk of priority inversion.
- **No inter-task communication:** tasks operate in isolation with no data exchange.
- **Negligible context-switch overhead:** although context switches were measured as a performance metric, their execution cost was assumed to be negligible and was not incorporated into task execution times. This assumption was adopted to isolate the intrinsic scheduling behavior of the evaluated algorithms and facilitate a fair comparison under identical conditions.

- **Deadline constraints:** relative deadlines are less than or equal to their respective task periods ($D_i \leq T_i$).

These assumptions are widely adopted in real-time scheduling research to facilitate algorithm comparison [3], [1], [7].

2.3 Scheduling Algorithms

2.3.1 Rate Monotonic Scheduling (RMS)

RMS is a fixed-priority scheduling algorithm that assigns higher priorities to tasks with shorter periods [3]. Task priorities remain unchanged throughout execution.

2.3.2 Deadline Monotonic Scheduling (DMS)

DMS is a fixed-priority scheduling algorithm that assigns priorities according to relative deadlines. Tasks with shorter deadlines receive higher priorities [1].

2.3.3 Earliest Deadline First (EDF)

EDF is a dynamic-priority scheduling algorithm that selects the task with the nearest absolute deadline for execution [3], [1].

2.3.4 Least Laxity First (LLF)

LLF is a dynamic-priority scheduling algorithm that assigns priorities according to task laxity, which represents the amount of time a task can be delayed without missing its deadline [1]. The laxity of task τ_i at time t is calculated as:

$$\mathbf{L_i(t) = D_i - t - R_i(t)} \quad \textbf{(2)}$$

where D_i denotes the absolute deadline of task τ_i, t represents the current time, and $R_i(t)$ is the remaining execution time. Tasks with smaller laxity values receive higher priorities. Although LLF can achieve high schedulability, its frequent priority recalculations may increase the number of context switches, resulting in higher scheduling overhead and implementation complexity [1].

2.4 Simulation Environment and Implementation

A custom simulation framework was developed in Python to evaluate the performance of the selected real-time scheduling algorithms. Python was selected due to its flexibility, extensive scientific computing ecosystem, and suitability for rapid prototyping and data analysis [1], [4]. The implementation was executed on a single-core simulation model that emulates a preemptive uniprocessor real-time environment. The framework supports the simulation of periodic task sets and enables the execution of multiple scheduling algorithms under identical operating conditions.

The framework was implemented using NumPy for numerical computations and statistical analysis, Pandas for data processing and result aggregation, and Matplotlib for data visualization and Gantt chart generation. Table 1 summarizes the software environment.

Table 1. Software environment and tools

Component	Description
Programming language	Python 3.x
Numerical library	NumPy
Data analysis library	Pandas
Visualization library	Matplotlib
Scheduling algorithms	RMS, DMS, EDF, LLF
Execution model	Single-core preemptive simulation
Task model	Independent periodic tasks

The simulation framework automatically performs the following steps:

1. Generate task sets.
2. Calculate the hyperperiod.
3. Generate jobs for each task.
4. Execute the selected scheduling algorithm.
5. Record scheduling decisions.
6. Calculate performance metrics.
7. Visualize results using Gantt charts and statistical plots.

2.5 Experimental Scenarios

Four workload scenarios were defined based on the processor utilization level U, calculated as:

$$\mathbf{U = \Sigma_{i=1}^{n} C_i / T_i} \quad \mathbf{(3)}$$

The target utilization levels were Low Load ($U \approx 0.40$), Medium Load ($U \approx 0.75$), High Load ($U \approx 0.95$), and Overload ($U \approx 1.30$). In addition to workload variation, a scalability analysis was conducted by varying the number of tasks in each task set, considering task sets containing 5, 10, and 20 tasks.

Task parameters were generated randomly for each experiment. Task periods were selected from the predefined set {20, 25, 40, 50, 80, 100}. Task execution times were generated according to the target processor utilization level using a uniform random distribution. Relative deadlines were randomly assigned within the interval $[C_i, T_i]$, ensuring that each generated task satisfied the constraint $D_i \leq T_i$.

To improve the reliability of the results and reduce the impact of random variations, ten independent experiments were performed for each workload scenario and task-set size, and each generated task set was evaluated under identical conditions using all four scheduling algorithms. This resulted in a total of 480 simulation runs (4 workload scenarios × 3 task-set sizes × 4 algorithms × 10 repetitions), providing a robust dataset for performance evaluation.

The reported results correspond to the mean and standard deviation computed across all independent experimental runs.

2.6 Performance Metrics

The performance of the evaluated algorithms was assessed using the following metrics [4], [6], [7].

2.6.1 Deadline Miss Ratio

The deadline miss ratio (DMR) measures the percentage of jobs that fail to complete before their specified deadlines and is calculated as:

$$\mathbf{DMR = (N_m / N_t) \times 100} \quad \mathbf{(4)}$$

where N_m represents the number of jobs that missed their deadlines, and N_t denotes the total number of generated jobs.

2.6.2 Average Response Time

Response time is defined as the duration between job release and job completion. The average response time (ART) is calculated as:

$$\mathbf{ART = (1 / N) \times \Sigma (F_i - R_i)} \quad \mathbf{(5)}$$

where N is the total number of completed jobs, F_i denotes the finish time of job i, and R_i represents its release time.

2.6.3 CPU Utilization

CPU utilization represents the proportion of simulation time during which the processor is actively executing tasks.

2.6.4 Context Switches

This metric measures the number of task preemptions and scheduling changes that occur during execution. A higher number of context switches generally indicates greater scheduling overhead [1], [6].

2.7 Statistical Analysis

For each experiment, the simulation framework generated execution traces and calculated the corresponding performance metrics. For each metric, the mean and standard deviation were calculated across all experiments to evaluate both average behavior and result variability. Comparative analysis was performed across different workload scenarios and task-set sizes to evaluate the impact of scheduling policies on system performance, and the findings were visualized using bar charts and line plots. The results are presented and discussed in Section 3.

3. Results and Discussion

3.1 Experimental Overview

The developed simulation framework was used to evaluate the performance of four real-time scheduling algorithms: RMS, DMS, EDF, and LLF. The evaluation encompassed four

distinct workload scenarios: low-load, medium-load, high-load, and overload conditions. Furthermore, a scalability analysis was conducted to assess system robustness by varying the task-set size (5, 10, and 20 tasks). The performance of each algorithm was measured using deadline miss ratio, average response time, CPU utilization, and context switches.

3.2 Performance under Different Workload Levels

Table 2 summarizes the average performance of the evaluated scheduling algorithms under different workload conditions.

Table 2. Performance comparison of scheduling algorithms across different workload scenarios

Workload Scenario	Algorithm	Average Response Time (Mean ± SD)	Context Switches (Mean ± SD)	Deadline Miss Ratio (%)
Low Load (40%)	RMS	6.46 ± 2.19	28.60 ± 9.65	6.43 ± 10.28
	DMS	6.98 ± 2.40	30.40 ± 10.56	0.87 ± 2.75
	EDF	6.97 ± 2.36	30.20 ± 10.62	0.87 ± 2.75
	LLF	7.45 ± 2.94	47.00 ± 31.58	0.87 ± 2.75
Medium Load (75%)	RMS	11.87 ± 4.92	24.60 ± 14.20	11.22 ± 16.17
	DMS	13.19 ± 5.07	24.80 ± 13.79	13.19 ± 5.07
	EDF	14.02 ± 4.36	24.50 ± 13.38	3.22 ± 7.38
	LLF	16.02 ± 4.74	64.00 ± 44.33	7.44 ± 16.06
High Load (95%)	RMS	10.19 ± 3.63	35.50 ± 15.34	24.84 ± 10.98
	DMS	10.15 ± 4.51	38.40 ± 16.75	23.84 ± 8.48
	EDF	12.18 ± 4.59	37.90 ± 15.85	21.41 ± 8.44
	LLF	14.42 ± 5.09	97.10 ± 51.60	37.57 ± 13.15
Overload (130%)	RMS	10.30 ± 3.96	41.30 ± 20.78	30.50 ± 9.42
	DMS	10.74 ± 4.57	41.80 ± 20.56	29.61 ± 10.22
	EDF	16.33 ± 4.75	41.20 ± 22.32	35.57 ± 14.19
	LLF	19.16 ± 11.37	134.80 ± 51.73	79.28 ± 4.46

As shown in Table 2, RMS achieved the lowest average response time while maintaining a relatively low number of context switches across all evaluated workload scenarios. DMS demonstrated performance comparable to RMS and achieved the lowest deadline miss ratios under medium-load (13.19%), high-load (23.84%), and overload (29.61%) conditions.

EDF exhibited moderate performance, with higher response times and deadline miss ratios than the fixed-priority approaches in several workload scenarios. In contrast, LLF generated

substantially more context switches due to frequent laxity recalculations, resulting in the highest scheduling overhead among the evaluated algorithms.

The results further indicate that increasing processor utilization generally led to higher deadline miss ratios and increased context-switch activity. However, the exact performance trends varied depending on workload characteristics and task-set configurations. Under overload conditions (130% utilization), significant performance differences emerged among the evaluated algorithms. LLF experienced a substantial degradation in schedulability, recording the highest deadline miss ratio (79.28%) and the largest number of context switches.

RMS consistently maintained the lowest average response times across most workload levels, followed closely by DMS. This behavior can be attributed to the lower runtime complexity and reduced scheduling overhead associated with fixed-priority scheduling approaches. Conversely, the frequent priority updates required by LLF led to excessive context switching, increased scheduling overhead, and longer response times, particularly under high-load and overload conditions.

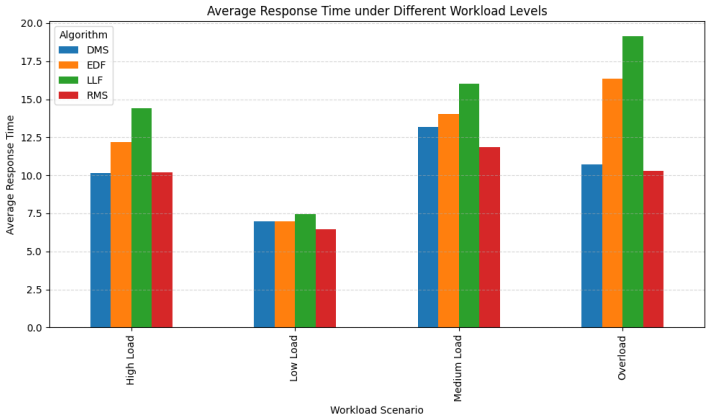


Figure 2. Average response time under different workload levels

The results presented in Figure 2 demonstrate a general upward trend in average response times as workload intensity increases. Notably, RMS consistently maintained the lowest average response times across the evaluated scenarios. Conversely, LLF exhibited the highest response times, a trend that became particularly pronounced under overload conditions due to excessive scheduling overhead.

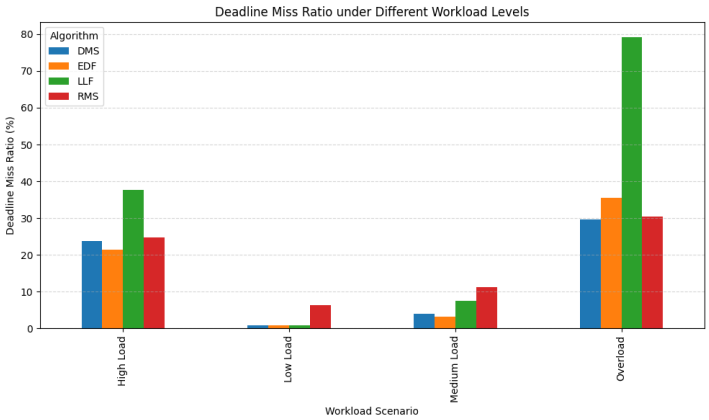


Figure 3. Deadline miss ratio under different workload levels

As illustrated in Figure 3, deadline miss ratios progressively increased with workload intensity across all algorithms. While EDF maintained the lowest miss ratio under medium and high-load conditions, a significant shift occurred under overload. In this extreme scenario, dynamic-priority algorithms (EDF and LLF) exhibited much higher miss ratios than fixed-priority ones (RMS and DMS), with LLF experiencing a severe performance collapse due to scheduling overhead.

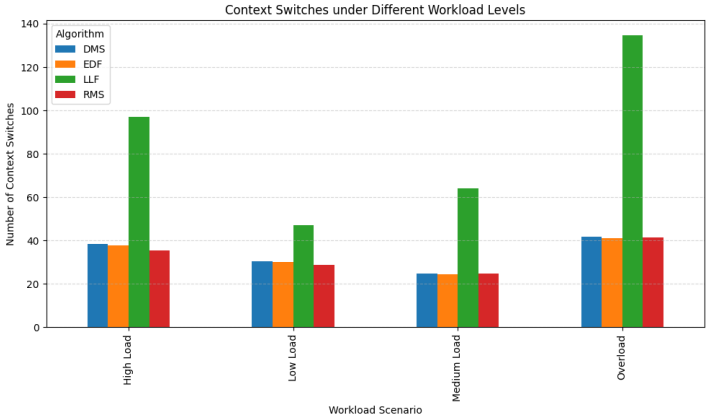


Figure 4. Context switches under different workload levels

As shown in Figure 4, LLF generated a substantially higher number of context switches compared to all other evaluated algorithms. This excessive overhead is directly attributed to its dynamic nature, which requires continuous priority recalculations based on real-time task laxity values. In contrast, fixed-priority algorithms, particularly RMS, maintained a significantly lower and more stable number of context switches.

3.3 Scalability Analysis

To investigate scalability, additional experiments were conducted using task sets of varying sizes (5, 10, and 20 tasks). The results indicate that increasing the number of tasks directly impacts scheduling overhead and response times, though the effect varies by algorithm.

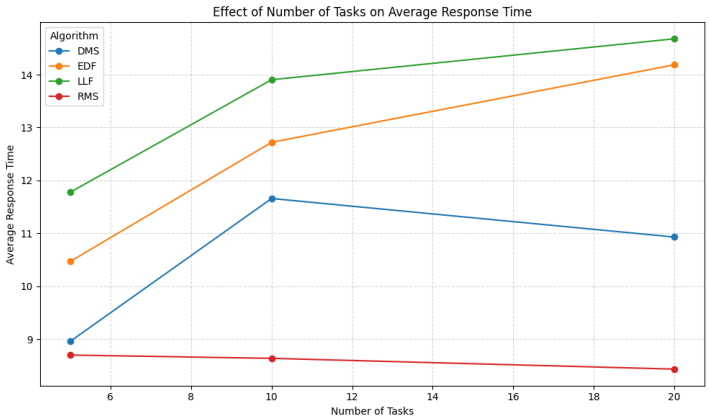


Figure 5. Effect of task-set size on average response time

As shown in Figure 5, while the average response time for dynamic-priority algorithms generally increased as the number of tasks grew, RMS exhibited remarkable stability. RMS maintained a consistently low and nearly flat response time across all evaluated task-set sizes, demonstrating highly favorable scalability. In contrast, LLF exhibited the highest response times due to frequent priority recalculations. DMS and EDF demonstrated intermediate performance.

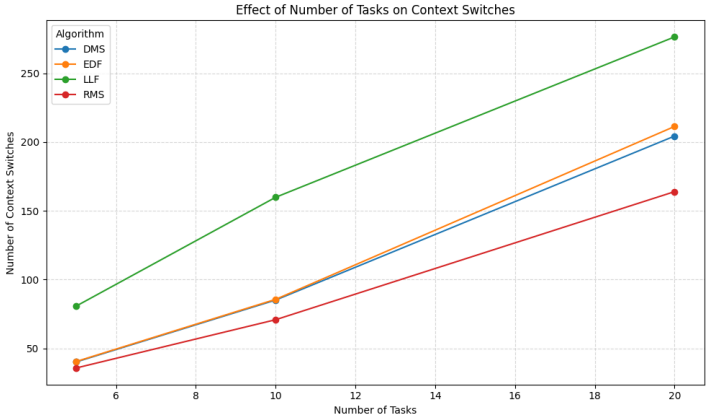


Figure 6. Impact of task-set size on context switches

Figure 6 illustrates that the number of context switches increased linearly as the number of tasks grew. LLF experienced the steepest growth rate due to its dynamic priority reassignment mechanism, whereas RMS demonstrated the lowest scheduling overhead. EDF and DMS exhibited nearly identical, moderate scalability characteristics between RMS and LLF.

3.4 Discussion

The experimental findings reveal distinct performance trade-offs among the evaluated scheduling algorithms. RMS demonstrated superior efficiency in terms of response time and context switches, confirming its suitability for embedded systems that prioritize predictable execution and low runtime complexity. DMS exhibited performance characteristics comparable to RMS while offering greater flexibility for task sets with constrained deadlines.

Although EDF is theoretically optimal for uniprocessor systems under ideal conditions [3], the simulation results indicate that fixed-priority approaches (RMS and DMS) achieved lower response times and reduced scheduling overhead. This performance advantage is attributed to the lower runtime complexity of fixed-priority scheduling, which avoids the overhead associated with dynamic priority recalculations.

LLF recorded the highest scheduling overhead, primarily driven by its continuous priority reassignment mechanism based on task laxity. Consequently, LLF suffered from significant performance degradation, particularly under overload conditions, where the high frequency of context switches led to increased response times and a surge in deadline misses. Overall, the results suggest that fixed-priority algorithms offer a favorable balance between predictability and runtime efficiency for many embedded applications, whereas dynamic-priority approaches, while theoretically superior in schedulability, may introduce excessive overhead in practical resource-constrained environments.

4. Conclusion

This paper presented a practical evaluation of classical real-time scheduling algorithms, namely Rate Monotonic Scheduling (RMS), Deadline Monotonic Scheduling (DMS), Earliest Deadline First (EDF), and Least Laxity First (LLF), using a custom-built simulation framework developed in Python. The framework enabled a rigorous comparative analysis under diverse workload scenarios, ranging from low-load to overload conditions, alongside a scalability study by varying task-set sizes.

The experimental results demonstrated that performance differences become increasingly pronounced as system utilization increases. RMS consistently achieved the lowest average response time and scheduling overhead, proving to be the most stable algorithm across varying task-set sizes. DMS performed similarly to RMS, providing a reliable alternative for constrained-deadline tasks. While EDF maintained reasonable schedulability under normal operating conditions, its performance, along with that of LLF, degraded notably under overload scenarios due to the complexity of dynamic priority management. LLF, in particular, suffered from “laxity thrashing,” resulting in the highest number of context switches and the most significant performance collapse under heavy loads.

In conclusion, the findings indicate that the selection of a scheduling algorithm should not be based solely on theoretical optimality, but also on the practical trade-offs regarding system complexity, workload variability, and scheduling overhead. Fixed-priority algorithms, specifically RMS, provide superior scalability and efficiency for systems where predictability and low overhead are paramount.

4.1 Future Work

The present study focuses on a single-core preemptive environment as a baseline for scheduling algorithm evaluation. Extending the proposed framework to multiprocessor and multicore platforms represents an important direction for future research [13], [14]. Future work may also investigate scheduling challenges in mixed-criticality systems, energy-aware

scheduling approaches, and machine learning-based adaptive scheduling techniques [11], [15].

In addition, the simulation framework can be extended to incorporate real-world factors that were not considered in the current study, including context-switch execution costs, resource-sharing mechanisms, synchronization protocols, cache-related delays, and energy consumption [16], [12], [11]. Further research may explore adaptive and hybrid scheduling approaches that dynamically adjust scheduling policies according to workload characteristics and system conditions [5], [7].

Finally, the proposed framework can be validated through implementation on real-time operating systems such as FreeRTOS or Zephyr and deployed on embedded hardware platforms to compare simulation results with practical execution behavior [1].

References

[1] Buttazzo, G. C. 2024. “Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications”, 4th ed. Cham, Switzerland: Springer. doi: 10.1007/978-3-031-45410-3.

[2] Burns, A. and Wellings, A. J. 2024. “Real-Time Systems and Programming Languages: Ada 95, Real-Time Java and Real-Time POSIX”, 5th ed. Harlow, U.K.: Pearson.

[3] Liu, C. L. and Layland, J. W., 1973, Scheduling algorithms for multiprogramming in a hard-real-time environment, *Journal of the ACM*, 20(1), pp. 46–61. doi: 10.1145/321738.321743.

[4] Leung, J. Y.-T. (Ed.). 2021. “Handbook of Scheduling: Algorithms, Models, and Performance Analysis”, 2nd ed. Boca Raton, FL, USA: CRC Press.

[5] Akesson, B., Nasri, M., Nelissen, G., Altmeyer, S. and Davis, R. I., 2022, A comprehensive survey of industry practice in real-time systems, *Real-Time Systems*, 58(3), pp. 358–398. doi: 10.1007/s11241-021-09376-1.

[6] Nelissen, G., Fonseca, J. and Nélis, V., 2021, Response-time analysis: Challenges and perspectives for the next generation of real-time systems, *Journal of Systems Architecture*, 120, Art. no. 102276. doi: 10.1016/j.sysarc.2021.102276.

[7] Baruah, S. K. and Buttazzo, G. C., 2022, Schedulability analysis of real-time systems: Current trends and challenges, *Real-Time Systems*, 58(1), pp. 1–24. doi: 10.1007/s11241-021-09379-y.

[8] Lakshmanan, K., de Niz, D. and Rajkumar, R., 2022, Coordinated task scheduling, allocation, and synchronization for distributed real-time embedded systems, *Proceedings of the IEEE*, 110(7), pp. 902–921. doi: 10.1109/JPROC.2022.3171899.

[9] Altmeyer, S. and Maiza, C., 2023, Timing analysis for real-time embedded systems: Recent developments and future directions, *ACM Transactions on Embedded Computing Systems*, 22(1), pp. 1–29. doi: 10.1145/3564739.

[10] Salamun, K., Pavić, I., Džapo, H. and Čuljak, I., 2023, Weakly hard real-time model for control systems: A survey, *Sensors*, 23(10), Art. no. 4652. doi: 10.3390/s23104652.

[11] Abolhassani Khajeh, S., Hossain, M. S. and Muhammad, G., 2023, Real-time scheduling in IoT applications: A systematic review, *Sensors*, 23(1), Art. no. 232. doi: 10.3390/s23010232.

[12] Bastoni, A., Brandenburg, B. B. and Anderson, J. H., 2022, Cache-related preemption and migration delays in real-time systems, *IEEE Transactions on Computers*, 71(4), pp. 1014–1027. doi: 10.1109/TC.2021.3119848.

[13] Kumar, A. and Gupta, S. K., 2022, A systematic survey of multiprocessor real-time scheduling and synchronization protocol, *International Journal of Sensors, Wireless Communications and Control*, 12(6), pp. 741–756. doi: 10.2174/2210327911666210616143638.

[14] Davis, R. I. and Burns, A., 2011, A survey of hard real-time scheduling for multiprocessor systems, *ACM Computing Surveys*, 43(4), Art. no. 35, pp. 35:1–35:44. doi: 10.1145/1978802.1978814.

[15] Burns, A. and Davis, R. I., 2018, A survey of research into mixed criticality systems, *ACM Computing Surveys*, 50(6), Art. no. 82, pp. 1–37. doi: 10.1145/3131347.

[16] Behnam, M. and Bertogna, M., 2021, Resource sharing in real-time embedded systems, *IEEE Embedded Systems Letters*, 13(3), pp. 81–84. doi: 10.1109/LES.2020.3047314.