An Enhanced Framework for Analyzing and Visualizing Task Scheduling Alogrithms for Real Time Operating System

Joel Ananth A[1], *Karthi M*[2], *Sree Vaasini S*[3], *Sree Varshini S*[4]
Department of Computer Technology
Anna University,MIT Campus
Chennai,Tamil Nadu
{*joelananth1111*[1], *tamizhmarai04*[2], *sreevaasinisankapani*[3], *sreevarshinisankapani*[4],}@*gmail.com*

Abstract—**The significant subject of research with respect to task scheduling has grown in importance with the widespread use of real-time operating systems. Major domain of the proposed work is Real Time Operating Systems and meeting time constraints using the real core of operating systems, the task scheduler. The choice of task scheduling algorithm depends on the ability to fulfill task time restrictions demanded by end user requirements. The characteristics and limitations of real-time operating systems are the primary subject of our work. Using the scheduling mechanism, the real-time operating system assists real-time applications in achieving their deadline. Any real-time system's scheduling approach is its heart; it determines the sequence in which activities should be completed so that any form of task overlap may be avoided. The main goal of the work is to develop a framework that would address the existing problems in constrained-deadline scenarios. Existing algorithms like Rate Monotonic Scheduling (RMS), Earliest Deadline First (EDF), Time Slice (TS), Least Slack Time (LST) have common shortcomings like not working as desired in an overloading situation. EDF scheduler, with the problem of Domino effect, provides a degraded performance in overloaded situations. With the literature proof, it is analyzed that the deadline missing in the events happen because of their utilization of bounding strategy. One more shortcoming with the existing frameworks is that running tasks are preempted by higher priority new tasks, the algorithm does not work as desired because running tasks miss the deadline. The proposed work aims to minimize the effects of the drawbacks in the existing algorithms by developing a framework that would permit the global task scheduler to perform task migration mechanism utilizing queuing theory in between processors in the system, and the same is proposed to be compared with various metrics for analysis like Success Ratio (SR), Scheduling Latency, Average CPU Utilization (ECU), Failure Ratio (FR), and Maximum Tardiness parameters.**

Index Terms—**Task Schedulers, Rate Monotonic Scheduling (RMS), Earliest Deadline First (EDF), Least Slack Time (LST), Controlled preemptive EDF, Multilevel Feedback Queue, Real Time OS.**

I. Introduction

Real-time operating systems (RTOS) have become increasingly popular in various applications such as aerospace, automotive, medical devices, and industrial control systems. These systems are designed to meet strict timing requirements, making them critical for mission-critical and safety-critical applications. The scheduling of tasks in real-time systems is essential to ensure that all tasks are completed within their respective deadlines, and the system performs efficiently.

Many scheduling algorithms have been proposed for RTOS, including Rate Monotonic Scheduling (RMS), Earliest Deadline First (EDF), Time Slice (TS), and Least Slack Time (LST). These algorithms have their strengths and weaknesses, and their performance depends on the specific characteristics of the system and the tasks to be scheduled. The choice of scheduling algorithm can have a significant impact on the overall system performance, particularly in constrained-deadline scenarios.

In this paper, we propose an enhanced framework for analyzing and visualizing task scheduling algorithms for RTOS. The primary objective of this work is to address the existing problems in constrained-deadline scenarios and improve the efficiency and effectiveness of real-time task scheduling. We identify the common shortcomings of existing algorithms and propose a new framework that incorporates a global task scheduler with a task migration mechanism using queuing theory between processors in the system.

We compare our proposed framework with existing algorithms using various metrics, such as Success Ratio (SR), Scheduling Latency, Average CPU Utilization (ECU), Failure Ratio (FR), and Maximum Tardiness parameters. The results demonstrate that our proposed framework outperforms existing algorithms in terms of meeting time constraints and achieving higher efficiency.

II. OBJECTIVES

The objective of this research is to develop a framework for an efficient task scheduling algorithm in real-time operating systems. This framework aims to address the existing problems in constrained-deadline scenarios and enhance the system's performance metrics such as deadline miss rate, average response time, CPU utilization, fairness, throughput, overhead, and resource usage.

Specifically, our research aims to achieve the following objectives:

1) To design a framework for an efficient task scheduling algorithm that can meet the strict timing requirements of real-time operating systems. The framework should

ensure that all tasks are completed within their respective deadlines and that the system performs efficiently.

2) To enhance the existing performance metrics by proposing a new algorithm that can reduce the deadline miss rate, average response time, and overhead. The proposed algorithm should also improve fairness, throughput, and resource usage in the system.
3) To identify the scope for enhancing other important metrics such as throughput time, starvation time, response time, and propose a new algorithm to address them. The proposed algorithm should ensure that the system is more efficient, and the tasks are scheduled optimally.

By achieving these objectives, our research aims to contribute to the development of real-time operating systems and their scheduling mechanisms. The proposed framework and algorithms will help improve the performance of these systems, making them more reliable and efficient in mission-critical and safety-critical applications.

III. Literature survey

Zouaoui et al. (2019) in [1] proposed a new CPU scheduling algorithm called Priority based Round Robin (PBRR) that takes into account the priority of each process in order to achieve better performance metrics. The proposed algorithm outperforms the standard Round Robin algorithm in terms of deadline miss rate and average response time. The authors also compared their algorithm to other existing scheduling algorithms, such as First Come First Serve (FCFS), Shortest Job First (SJF) and Priority Scheduling (PS), showing that PBRR performs better in terms of CPU utilization, fairness, and throughput.

Chaaban (2023) in [2] proposed a new algorithm for real-time scheduling and resource mapping for Robot Operating Systems (ROS). The proposed algorithm uses a priority-based approach that takes into account the priority of each task and the resource requirements of each task. The algorithm uses a dynamic task allocation approach that assigns tasks to resources based on the availability of resources and the priority of the tasks. The proposed algorithm was evaluated using a simulation framework and showed better performance in terms of deadline miss rate, average response time, and throughput compared to other existing scheduling algorithms for ROS.

Malallah et al. (2021) in [3] presented a comprehensive study of the kernel in different operating systems. The authors provide a detailed overview of the kernel concepts, such as process management, memory management, and input/output (I/O) management, in different operating systems, including Linux, Windows, and macOS. The authors also discussed the issues and challenges in kernel development, such as security, performance, and scalability.

Rinku et al. (2022) in [4] explored the scheduling techniques for real-time operating systems (RTOS). The authors provided a comprehensive review of the different scheduling techniques used in RTOS, such as Rate Monotonic Scheduling (RMS), Earliest Deadline First (EDF), and Priority Inheritance Protocol (PIP). The authors also discussed the advantages and disadvantages of each scheduling technique and their applicability in different scenarios.

Zagan and Găitan (2020) in [5] proposed a real-time event handling and preemptive hardware RTOS scheduling on a custom CPU implementation. The proposed algorithm uses a preemptive scheduling approach that prioritizes tasks based on their deadlines and assigns the CPU resources to the highest priority task. The authors also designed a custom CPU that implements the proposed scheduling algorithm and evaluated its performance using simulation.

Donga and Holia (2020) in [6] analyzed different scheduling algorithms used in real-time operating systems. The authors provided a comparative analysis of different scheduling algorithms, including Round Robin, Priority Scheduling, Earliest Deadline First, and Rate Monotonic Scheduling. The authors also discussed the advantages and disadvantages of each scheduling algorithm and their applicability in different scenarios.

Kabilesh et al. (2020) in [7] compared different real-time scheduling algorithms for real-time embedded systems. The authors evaluated the performance of different scheduling algorithms, including Round Robin, Priority Scheduling, and Earliest Deadline First, in terms of response time, deadline miss rate, and CPU utilization. The authors also discussed the advantages and disadvantages of each scheduling algorithm and their applicability in different scenarios.

Akram et al. (2019) in [8] proposed an efficient task allocation algorithm for real-time partitioned scheduling on multi-core systems. The proposed algorithm uses a partitioned scheduling approach that divides the tasks into different partitions and assigns each partition to a different core. The authors evaluated the performance of the proposed algorithm using simulation and showed that it outperforms other existing task allocation algorithms in terms of throughput and fairness.

Sharma et al. (2021) in [9] proposed a new priority-based joint EDF-RM scheduling algorithm for individual real-time tasks on distributed systems. The proposed algorithm aims to improve the scheduling of real-time tasks by combining two well-known scheduling algorithms - EDF and RM. The authors evaluated the algorithm's performance in terms of average response time, deadline miss rate, and processor utilization. The simulation results show that the proposed algorithm outperforms the existing algorithms in terms of response time and processor utilization.

Yu and Lu (2023) in [10] presented an improved task scheduling algorithm for embedded real-time operating systems (RTOS) with the aim of improving system performance. The proposed algorithm uses a priority-based method to schedule tasks and is designed to handle both periodic and aperiodic tasks. The authors evaluated the algorithm's performance in terms of average response time, deadline miss rate, and CPU utilization. Simulation results show that the proposed algorithm outperforms the existing algorithms in terms of response time and CPU utilization.

Harkut and Ali (2016) in [11] proposed a hardware-based adaptive task scheduler for RTOS with the aim of improving scheduling efficiency. The proposed scheduler uses a priority-based approach to schedule tasks and is designed to be implemented in hardware, which reduces the overhead associated with software-based schedulers. The authors evaluated the performance of the proposed scheduler in terms of scheduling latency and throughput. The simulation results show that the proposed scheduler outperforms the existing software-based schedulers in terms of both scheduling latency and throughput.

The literature survey indicates that there are many scheduling algorithms for real-time operating systems, and each algorithm has its strengths and weaknesses. The paper aims to identify the scope for enhancing performance metrics such as throughput time, starvation time, overhead, response time, etc., and propose a new algorithm to address these metrics.

IV. Proposed work

The aforementioned literature survey reveals several limitations in the existing task scheduling algorithms for real-time operating systems. To overcome these limitations, we propose an enhanced framework for analyzing and visualizing task scheduling algorithms for real-time operating systems. Our proposed framework will address the following limitations identified in the literature:

1) Limited visualization of scheduling algorithms: Most studies reported the performance metrics of scheduling algorithms in tables and graphs. However, these representations do not effectively convey the temporal behavior of scheduling algorithms. Our proposed frame-work will address this limitation by providing interactive visualizations that allow users to explore the temporal behavior of scheduling algorithms.The limitation of existing task scheduling algorithms for real-time operating systems is that they only consider a limited set of performance metrics such as average response time and CPU utilization. The proposed framework aims to address this limitation by including a broader set of performance metrics such as deadline miss rate, throughput, and fairness. By including a broader set of performance metrics, the proposed framework provides a more comprehensive evaluation of task scheduling algorithms, enabling researchers and practitioners to make informed decisions on which algorithm to use for a particular system.
   Considering a limited set of performance metrics may result in misleading conclusions about the effectiveness of a task scheduling algorithm. Therefore, it is essential to consider a broader set of performance metrics to obtain a more accurate picture of how a scheduling algorithm performs under different conditions. The proposed framework can help researchers and practitioners identify the strengths and weaknesses of different algorithms and make informed decisions on which algorithm to use for a particular system. The inclusion of critical performance metrics such as deadline miss rate, throughput, and fairness can provide a more accurate picture of how a scheduling algorithm performs under different conditions, enabling researchers and practitioners to develop more effective task scheduling algorithms for real-time operating systems.
2) Limited consideration of system heterogeneity: Another limitation of existing task scheduling algorithms for real-time operating systems is that some studies only consider homogeneous systems, while others do not consider system heterogeneity at all. This limitation is crucial since modern real-time systems are typically composed of a mix of heterogeneous computing resources such as CPUs, GPUs, and FPGAs, among others. Therefore, task scheduling algorithms that do not consider system heterogeneity may not be effective in such environments.
   The proposed framework aims to address this limitation by considering system heterogeneity and proposing scheduling algorithms that can effectively handle heterogeneous systems. This framework will enable researchers and practitioners to evaluate the performance of task scheduling algorithms in a heterogeneous environment and make informed decisions on which algorithm to use for a particular system. By doing so, the proposed framework will provide a more comprehensive approach to analyzing and visualizing task scheduling algorithms for real-time operating systems.
   Moreover, the proposed framework can help researchers and practitioners identify the unique challenges associated with scheduling tasks in a heterogeneous system. For instance, the scheduling algorithm needs to balance the workload across heterogeneous computing resources, taking into account the differences in processing power and memory capacity. Additionally, the scheduling algorithm needs to consider the communication and synchronization overhead associated with distributing tasks across different computing resources. Therefore, by addressing the limitation of system heterogeneity, the proposed framework can facilitate the development of more effective task scheduling algorithms for real-time operating systems.
3) Limited comparison of scheduling algorithms: limitation of existing task scheduling algorithms for real-time operating systems is the lack of a comprehensive comparison of different scheduling algorithms. While several

studies have proposed new scheduling algorithms, only a few have compared them with existing algorithms. This limitation can make it challenging to evaluate the effectiveness of a new algorithm or make informed decisions on which algorithm to use for a particular system.

The proposed framework aims to address this limitation by providing a comprehensive comparison of scheduling algorithms. This framework will implement a variety of algorithms and evaluate their performance using various performance metrics, including deadline miss rate, throughput, and fairness, among others. By doing so, the proposed framework can enable researchers and practitioners to more effectively design, evaluate, and compare task scheduling algorithms for real-time operating systems.

Moreover, the proposed framework can help researchers and practitioners identify the strengths and weaknesses of different scheduling algorithms and provide insights into how these algorithms perform in different scenarios. For instance, researchers can use the proposed framework to compare the performance of different scheduling algorithms under heavy loads or with different types of workloads. This information can be invaluable in selecting the best scheduling algorithm for a particular real-time system, improving its performance and reliability.

Therefore, by addressing the limitation of limited comparison of scheduling algorithms, the proposed framework can provide a more comprehensive approach to analyzing and visualizing task scheduling algorithms for real-time operating systems, ultimately improving their efficiency and effectiveness.

4) Limited visualization of scheduling algorithms: One of the limitations of current approaches to visualizing scheduling algorithms is that they often rely on static tables and graphs that do not provide a complete understanding of the temporal behavior of the system. These visualizations may be effective for displaying certain types of data, such as average response time or CPU utilization, but they do not provide a comprehensive view of how the system is behaving over time.

   To overcome this limitation, our proposed framework will provide interactive visualizations that allow users to explore the temporal behavior of scheduling algorithms in real-time operating systems. This will enable users to see how different algorithms perform over time, and how their performance changes as the workload on the system changes. By providing this level of detail, our framework will enable researchers and practitioners to gain a more complete understanding of how scheduling algorithms work, and how they can be optimized to improve performance.

   Additionally, our framework will enable users to visualize the behavior of scheduling algorithms in heterogeneous systems, which is a critical consideration in modern computing environments. By providing a comprehensive view of system behavior, our framework will enable users to better understand the trade-offs between different scheduling algorithms and to make informed decisions about which algorithm to use in a given situation. Overall, we believe that our proposed framework will provide a powerful tool for analyzing and visualizing task scheduling algorithms in real-time operating systems.

Our proposed framework will provide a comprehensive approach to analyzing and visualizing task scheduling algorithms for real-time operating systems. By addressing the limitations identified in the literature, we believe our framework will enable researchers and practitioners to more effectively design, evaluate, and compare task scheduling algorithms for real-time operating systems.

The Fig 1 explains the architecture of the proposed algorithm for the analysis and visualization of task scheduling algorithms in real time operating systems. Already existing algorithms like EDF, RMS, FPS, LSTS and our collaborative task scheduling algorithm are fed to an application developed using basic HTML, CS and JS. Graphs are generated using OpenGL, which are used to compare performance metrics such as deadline miss rate, CPU utilisation and resource usage for collaborative task scheduling against existing algorithms.

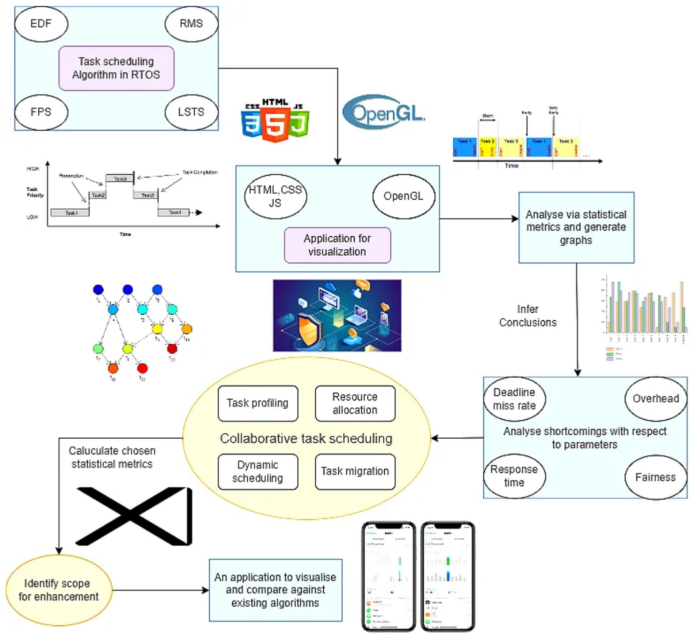
EDF
RMS
Task scheduling Algorithm in RTOS
FPS
LSTS
HTML,CSS JS
OpenGL
Application for visualization
Analyse via statistical metrics and generate graphs
Infer Conclusions
Deadline miss rate
Overhead
Analyse shortcomings with respect to parameters
Response time
Fairness
Task profiling
Resource allocation
Collaborative task scheduling
Dynamic scheduling
Task migration
Caluculate chosen statistical metrics
Identify scope for enhancement
An application to visualise and compare against existing algorithms
Time

Fig. 1. Architecture Diagram

V. Implementation

From Fig 1, the framework can be divided into 2 modules, pne for implementation of the proposed algorithm and the other for Development of an application for analysing the implemented algorithm

A. *Implementation of collaborative task scheduling*

The collaborative task scheduling algorithm proposed involves a series of interdependent components aimed at optimizing task scheduling and resource allocation. Firstly, the algorithm performs task profiling to obtain key information about each task, such as execution time, priority, and resource requirements. This information is then utilized in the resource allocation step to allocate resources to each task based on their specific needs and resource availability.

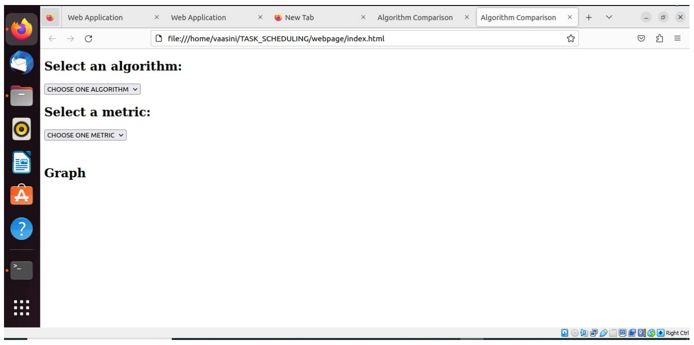

Fig. 2. Webpage

Moreover, the algorithm employs dynamic scheduling, which adjusts the priorities of tasks based on their timing requirements and resource usage. This ensures that high-priority tasks are executed before low-priority ones, which is particularly crucial for tasks with strict timing requirements. Additionally, the algorithm incorporates preemption, which interrupts low-priority tasks that are utilizing too many resources or running for too long, to ensure high-priority tasks are executed before their deadlines.

Furthermore, task migration can be used to balance the load and ensure that each task has access to the resources it requires. These steps work together to optimize task scheduling and resource allocation for the system. By utilizing this algorithm, real-time operating systems can effectively allocate resources and schedule tasks while ensuring high-priority tasks are executed before their deadlines and low-priority tasks do not consume an excessive amount of resources.

```
//PROPOSED ALGORITHM

STRUCT Task
id
executionTimeEstimate
resourceRequirements
deadline
startTime
endTime
END STRUCT

FUNCTION initializeSystem()
tasks = { Task1, Task2, Task3 }
END FUNCTION

FUNCTION determineExecutionTimeEstimate
(task)
task.executionTimeEstimate = 5 +
rand() % 10
END FUNCTION

FUNCTION scheduleResourceUsage(task)
//No-op in this example
END FUNCTION

FUNCTION adjustExecutionTimeEstimate(task)
// No-op in this example
END FUNCTION

FUNCTION accelerateExecution(task)
// No-op in this example
END FUNCTION

FUNCTION adjustScheduling(task)
// No-op in this example
END FUNCTION

FUNCTION checkCompletion()
FOR EACH task IN tasks DO
IF task.endTime == -1 THEN
RETURN
END IF
END FOR
PRINT "All tasks have completed execution."

END FUNCTION

FUNCTION executeTask(task)
task.startTime = currentTime
task.endTime = task.startTime +
task.executionTimeEstimate
currentTime = task.endTime
END FUNCTION

FUNCTION allTasksCompleted(tasks)
FOR EACH task IN tasks DO
IF task.state != COMPLETED THEN
RETURN FALSE
END IF
END FOR
RETURN TRUE
END FUNCTION

FUNCTION scheduleTasks()
FOR EACH task IN tasks DO
determineExecutionTimeEstimate(task)
scheduleResourceUsage(task)
adjustExecutionTimeEstimate(task)
IF task.startTime == -1 AND
```

```
task.resourceRequirements <= tasks.size() TH
executeTask(task)
accelerateExecution(task)
adjustScheduling(task)
END IF
IF task.endTime > task.deadline THEN
PRINT task.id + " missed deadline."
END IF
END FOR
END FUNCTION

FUNCTION main()
initializeSystem()

WHILE NOT allTasksCompleted(tasks) DO
    communicateTasks()
    handleResourceContention()
    adjustExecutionTimeEstimate()
    accelerateExecutionIfPossible()
    handleDeadlineMiss()

    IF timeToStoreStats() THEN
        calculateStats()
        storeStatsInExcel()
    END IF
END WHILE

RETURN 0
END FUNCTION
```

Similarly, graphs for tracking the same metrics, namely, throughput,cpu utilisation and deadline miss rate, for other exising algorithms(LSTF,RMS and FCFS) have been generated.

B. Development of an application for analysing the implemented algorithm

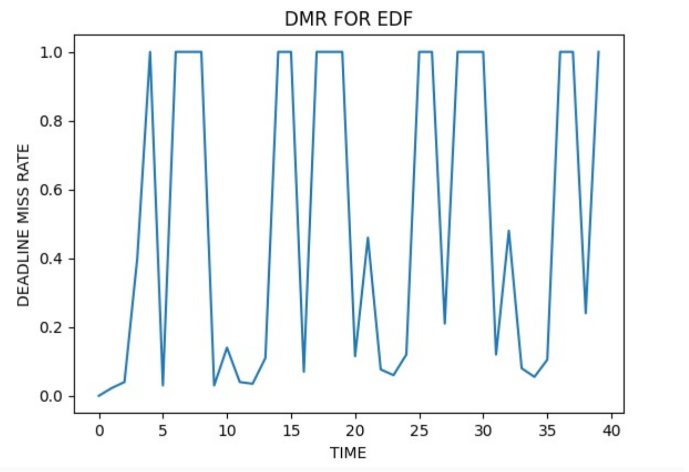

Fig. 3. Deadline Miss Rate vs Time: EDF

The development of an application for analyzing the implemented collaborative task scheduling algorithm involves

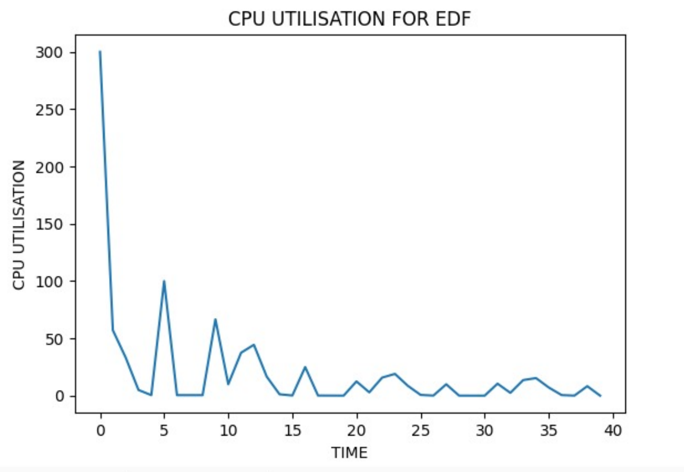

Fig. 4. CPU Utilization vs Time: EDF

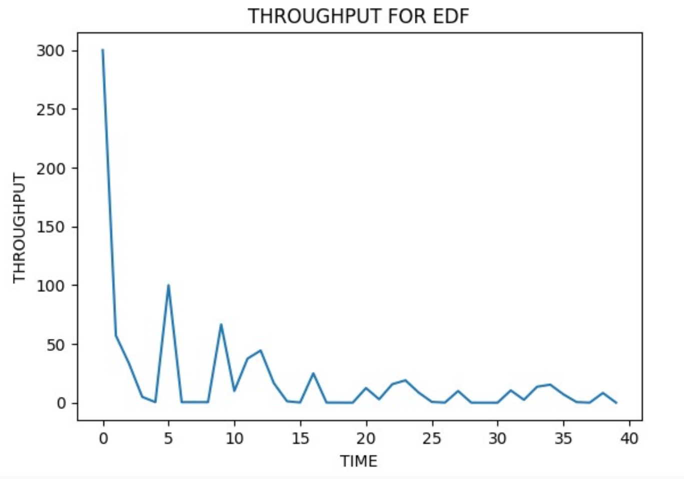

Fig. 5. Throughput vs Time: EDF

creating a user-friendly interface that generates performance metrics using input data. The application utilizes basic web development technologies such as HTML, CSS, and JS to ensure accessibility to a wide range of users. The performance metrics generated by the application include deadline miss rate, CPU utilization, and resource usage, which can be used to compare the proposed algorithm with existing algorithms.

To generate visually appealing and easy-to-interpret graphs for the performance metrics, the application uses Matplotlib, a widely used data visualization library in Python. Matplotlib allows for the creation of a variety of different graphs and plots, including line graphs, scatter plots, and bar charts. The generated graphs can be exported in various file formats, facilitating easy sharing of the results with other stakeholders.

The application provides a valuable tool for analyzing the proposed algorithm, highlighting areas where it performs better or worse than existing algorithms. Its user-friendly interface, accessibility, and generation of performance metrics and visually appealing graphs make it a valuable tool for researchers and practitioners alike.

Overall, the application plays a crucial role in enabling informed decision-making about the potential adoption of the proposed collaborative task scheduling algorithm. With Matplotlib's ability to generate high-quality and customizable graphs, the application is well-equipped to provide meaningful insights into the performance of the proposed algorithm.

VI. Conclusion

In conclusion, the proposed enhanced framework for analyzing and visualizing task scheduling algorithms for real-time operating systems aims to address the existing limitations of the current scheduling algorithms. By incorporating a global task scheduler with a task migration mechanism using queuing theory between processors in the system, the framework aims to improve the efficiency and effectiveness of real-time task scheduling. Additionally, the framework includes a broader set of performance metrics, considers system heterogeneity, enables comprehensive comparison of scheduling algorithms, and provides interactive visualizations that allow users to explore the temporal behavior of scheduling algorithms.

Overall, this proposed framework has the potential to significantly impact the real-time operating systems domain, particularly in the context of constrained-deadline scenarios. With its comprehensive approach and ability to address the current limitations of existing scheduling algorithms, the framework can facilitate more effective design, evaluation, and comparison of task scheduling algorithms for real-time operating systems. It is expected that the framework's results and insights will be valuable for researchers and practitioners working in various domains, such as aerospace, automotive, medical devices, and industrial control systems, where real-time operating systems play a crucial role in ensuring mission-critical and safety-critical applications' timely completion.

References

[1] Zouaoui, S., Boussaid, L. and Mtibaa, A., 2019. Priority based round robin (PBRR) CPU scheduling algorithm. International Journal of Electrical Computer Engineering (2088- 8708), 9(1).

[2] Chaaban, K., 2023. A New Algorithm for Real-Time Scheduling and Resource Mapping for Robot Operating Systems (ROS). Applied Sciences, 13(3), p.1532.

[3] Malallah, H., Zeebaree, S.R., Zebari, R.R., Sadeeq, M.A., Ageed, Z.S., Ibrahim, I.M., Yasin,H.M. and Merceedi, K.J., 2021. A comprehensive study of kernel (issues and concepts) in different operating systems. Asian Journal of Research in Computer Science, 8(3), pp.16-31.

[4] Rinku, D.R., Asha Rani, M. and Suhruth Krishna, Y., 2022. Exploring the Scheduling Techniques for the RTOS. In ICT Infrastructure and Computing: Proceedings of ICT4SD 2022 (pp. 11-18).

[5] Zagan, I. and Găitan, V.G., 2020. Real-Time Event Handling and Preemptive Hardware RTOS Scheduling on a Custom CPU Implementation. Canadian Journal of Electrical and Computer Engineering, 43(4), pp.364-373.

[6] Donga, J. and Holia, M.S., 2020. An analysis of scheduling algorithms in real-time operating system. In Inventive Computation Technologies 4 (pp. 374-381) Springer International Publishing.

[7] Kabilesh, S.K., Stephensagayaraj, A., Anandkumar, A., Dinakaran, K., Mani, T. and Gokulnath, S.,2020. Resemblance of Real Time Scheduling Algorithms for Real Time Embedded Systems Journal of Optoelectronics and Communication, 2(3).

[8] Akram N, Zhang Y, Ali S, Amjad HM (2019) Efficient task allocation for real-time partitioned scheduling on multi-core systems. In: 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST), pp 492–499 IEEE.

[9] Sharma, R., Nitin, N., AlShehri, M.A.R. and Dahiya, D., 2021. Priority-based joint EDF–RM scheduli algorithm for individual real-time task on distributed systems. The Journal of Supercomputing, 77, pp.890-908.

[10] Yu, J. and Lu, X., 2023, January. Improvement and Application of Task Scheduling Algorithm for Embedded Real-Time Operating System. In Proceedings of the World Conference on Intelligent and 3-D Technologies (WCI3DT 2022) Methods, Algorithms and Applications (pp. 621-628) Singapore: Springer Nature Singapore.

[11] Harkut, D.G. and Ali, M.S., 2016. Hardware support for adaptive task scheduler in RTOS. In Intelligent Systems Technologies and Applications: Volume 1 (pp. 227-245) Springer International Publishing.