International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2022): 7.942

Comparative Analysis of Scheduling Algorithms for Latency Optimization in Real-Time Applications

Jatin Pal Singh

Abstract: ***Real-time applications, from video conferencing to financial trading, hinge on the timely delivery of data streams. However, latency can disrupt this flow, compromising user experience and operational efficiency. This paper explores scheduling and resource allocation algorithms as weapons against latency in data streaming. We delve into Earliest Deadline First (EDF), Weighted Fair Queuing (WFQ), and Rate Monotonic Scheduling (RMS), dissecting their mechanisms, theoretical guarantees, and practical implications for latency optimization. We examine trade-offs between these algorithms, considering factors such as deadline constraints, resource availability, and fairness requirements. Through theoretical analysis and experimental evaluation, we aim to provide actionable insights and recommendations for selecting and implementing optimal scheduling strategies in diverse data streaming scenarios, paving the way for more responsive and efficient real-time applications.***

Keywords: Data streaming, Latency optimization, Scheduling algorithms, Resource allocation, Earliest Deadline First (EDF), Weighted Fair Queuing (WFQ), Rate Monotonic Scheduling (RMS), Network Calculus

1.Introduction

In the relentless march of real-time applications, where decisions are made in milliseconds, data streams are the lifeblood. But like unruly crowds vying for a limited exit, these streams can be plagued by chaos and delay. Enter the strategic art of scheduling and resource allocation, the key to transforming this data deluge into a smooth, responsive flow.

This paper delves into the intricate mechanics of this art, dissecting scheduling algorithms as the conductors orchestrate the data symphony and resource allocation strategies as the stage managers ensuring each packet gets its moment in the spotlight. We peel back the layers of Earliest Deadline First, where milliseconds become deadlines and data packets sprint to meet them. We unveil the balancing act of Weighted Fair Queuing, where different streams claim their rightful share of the bandwidth, each note given its due prominence. And we explore the Rate Monotonic Scheduling, a static priority scheduling algorithm where tasks with shorter periods have higher priority.

Our journey is not merely theoretical. We venture into the real-time battlefield, wielding these algorithms as tools to conquer latency in diverse applications. We witness video conferences transformed from choppy stutters to seamless exchanges, financial markets dancing to the rhythm of timely data feeds, and industrial control systems operating with the precision of a well-oiled machine.

This is not just a battle against milliseconds; it's a quest for the perfect data flow, where information arrives precisely when needed, empowering real-time applications to reach their full potential. Join us as we turn the knobs of scheduling and allocation, crafting the future where data streams sing in perfect harmony, unburdened by the shackles of delay.

Overview of Scheduling and Resource Allocation methods:

The scheduling and resource allocation methods in computing systems primarily focus on optimizing the use of available resources while ensuring efficient task execution. Scheduling methods determine the order and timing of tasks, considering factors like priority and resource availability. Resource allocation involves assigning the necessary resources to each task, such as CPU time, memory, or bandwidth. Various algorithms and strategies are employed in different contexts, from operating systems to cloud computing, each with a specific focus on minimizing latency, maximizing throughput, or balancing load effectively. This involves a detailed examination of existing algorithms, their design principles, and performance metrics in various scenarios. While there are several scheduling & resource allocation methods - Least Slack Time, Max-Min Fairness Algorithm, Proportional Fair Scheduling, Banker's Algorithm, Dynamic Resource Allocation etc, this paper's scope is to compare - Earliest Deadline First (EDF), Weighted Fair Queuing (WFQ) & Rate Monotonic Scheduling (RMS) in terms of their effectiveness in reducing latency, with a focus on specific application areas like real-time systems or network traffic management. Let's get a basic overview of these three methods:

- **Earliest Deadline First (EDF):** EDF simply prioritizes tasks based on their deadlines, with the tasks having the earliest deadlines being addressed first. This method is particularly useful in real-time systems where meeting task deadlines is critical. EDF aims to minimize the number of **missed deadlines**, making it a suitable choice for systems where timely task completion is a priority.
- **Weighted Fair Queuing (WFQ):** WFQ is a network scheduling algorithm that allocates bandwidth among multiple data flows. It's used to ensure fair bandwidth distribution, particularly in situations where network traffic must be managed efficiently. WFQ assigns weights to different queues or traffic flows, based on which it schedules their packets, thus providing a fair allocation of resources based on the predefined weights. This approach is particularly beneficial for handling

diverse types of traffic with **varying bandwidth** requirements in a network.

- **Rate Monotonic Scheduling (RMS):** Rate Monotonic Scheduling (RMS) is a fixed-priority algorithm used primarily in real-time operating systems. It assigns priority to tasks based on their request rate (frequency of execution), with the task having the shortest period (or highest frequency) receiving the highest priority. This scheduling method is optimal for periodic tasks in preemptive systems where each task's execution time is less than or equal to its period. RMS is widely used due to its simplicity and effectiveness in ensuring that time-critical tasks are completed within their required deadlines.

Let's take a deeper look at each of these & compare.

I - Earliest Deadline First (EDF)

In the fast-paced world of data streaming, where milliseconds matter, Earliest Deadline First (EDF) emerges as a potent weapon against the nefarious villain of latency. This dynamic scheduling algorithm prioritizes tasks based on their **deadlines**, ensuring the timely delivery of data for real-time applications. Imagine it as a meticulous maestro, orchestrating the flow of data packets, ensuring those with imminent deadlines reach their destination first. EDF is an *optimal* scheduling algorithm on preemptive uniprocessors, in the following sense: if a collection of independent *jobs*, each characterized by an arrival time, an execution requirement and a deadline, can be scheduled (by any algorithm) in a way that ensures all the jobs complete by their deadline, the EDF will schedule this collection of jobs so they all complete by their deadline. For example, consider three tasks:

Task A: Execution Time = 2 units, Deadline = 4 units
Task B: Execution Time = 1 unit, Deadline = 6 units
Task C: Execution Time = 3 units, Deadline = 8 units

Let's assume

- All tasks arrive at time 0.
- Execution Time is the time required to complete the task.
- Deadline is the time by which the task must be completed.

Time Units	Task Scheduled
0-2	Task A
2-3	Task B
3-6	Task C

At time 0, all tasks are available. Task A has the earliest deadline at 4 units, so it is scheduled first. Task A completes at time 2. Next, Task B, with the next earliest deadline at 6 units, is scheduled. Task B completes at time 3. Finally, Task C is scheduled and runs until it completes at time 6. If you observe, EDF works with three main ideas 1) **Deadline Awareness:** Each data packet in the stream carries a pre-assigned deadline, signifying the latest acceptable time for its arrival. 2) **Priority Queue:** EDF maintains a prioritized queue, placing packets with earlier deadlines closer to the front. & 3) **Preemptive Scheduling:** If a new packet with an earlier deadline arrives while another task is being processed, EDF preempts the ongoing task and prioritizes the newcomer. This ensures that deadline-critical data receives immediate attention.

Benefits of EDF:

- **Minimized Latency:** By prioritizing tasks with tighter deadlines, EDF reduces the average time it takes for data to reach its destination, contributing to a smoother and more responsive experience for users.
- **Guaranteed Deadlines:** For streams with strictly defined deadlines, EDF provides theoretical guarantees that all packets will arrive on time, making it ideal for mission-critical applications.
- **Fairness and Efficiency:** EDF avoids starvation by eventually processing all incoming packets, even those with later deadlines. This ensures overall fairness in resource allocation and prevents lower-priority data from being indefinitely delayed.

Challenges and Considerations:

- **Deadline Accuracy:** The effectiveness of EDF hinges on reliable deadline estimation. Inaccurate deadlines can lead to suboptimal scheduling and potential deadline violations.
- **Overhead:** Maintaining a prioritized queue and preemption mechanisms can introduce some overhead, potentially impacting overall system performance.
- **Real-Time Implementation:** Implementing EDF in real-time systems demands efficient data structure and algorithm choices to minimize processing delays and prevent deadline misses.

EDF in Action (real world use cases):

Despite its challenges, EDF is a valuable tool for latency optimization in diverse data streaming applications. Consider its potential in:

- **Real-time video conferencing:** Prioritizing video frames with stricter deadlines ensures smooth playback and minimizes frustrating audio-video synchronization issues.
- **Financial market data feeds:** Timely delivery of stock prices and trading updates is crucial for making informed investment decisions. EDF provides the necessary speed and reliability.
- **Embedded Systems:** EDF is used in embedded systems for controlling machinery, robots, or other devices where tasks need to be completed in a timely manner.
- **Automotive Systems:** In automotive electronics, EDF schedules tasks like sensor data processing and actuator control to ensure timely responses.
- **Medical Devices:** For critical healthcare devices like heart monitors and ventilators, EDF ensures that tasks are executed within their deadlines for patient safety.

- **Multimedia Systems:** EDF is used to manage audio and video streaming to prevent lags and ensure smooth playback.
- **Industrial Control Systems:** In automated production lines, EDF schedules tasks to synchronize machinery operations efficiently.
- **Network Routers and Switches:** EDF can prioritize data packets to optimize network traffic flow and reduce latency.

EDF can be further fine-tuned and augmented with other latency-mitigating techniques like caching and compression to build a robust arsenal against the enemy of latency. Through careful consideration and optimization, EDF can empower a new generation of real-time applications that thrive on immediacy and precision.

2.Weighted Fair Queuing (WFQ)

Weighted Fair Queuing (WFQ) is an advanced network scheduling algorithm used in packet-switched networks. It is an extension of the Fair Queuing (FQ) algorithm but with an important distinction: WFQ allows for weighted allocation of bandwidth among different traffic flows. In WFQ, each flow of packets is assigned a weight, and bandwidth is allocated to these flows in proportion to these weights. This means that flows with higher weights are given more bandwidth, ensuring that important or priority traffic can be transmitted faster. Let's understand this with an example. Imagine a network where three different types of traffic – A, B, and C – are being transmitted. Each type of traffic has been assigned a different weight based on its priority or importance.

- Traffic Type A (High Priority): Weight = 3
- Traffic Type B (Medium Priority): Weight = 2
- Traffic Type C (Low Priority): Weight = 1

Traffic Type	Weight	Proportion of Total Bandwidth	Allocated Bandwidth
A	3	3/ (3+2+1) = 0.5	60 Mbps * 0, 5 = 30 Mbps
B	2	2/ (3+2+1) = 0.333	60 Mbps * 0.333 = 20 Mbps
C	1	1/ (3+2+1) = 0.167	60 Mbps * 0.167 = 10 Mbps

These weights determine the proportion of bandwidth each traffic type receives. If the total available bandwidth is, for instance, 60 Mbps, the distribution of this bandwidth among the different types of traffic is based on their respective weights. In the above table:

- Traffic Type A, being the highest priority, gets half of the total bandwidth (30 Mbps), as its weight is 3 out of the total weight of 6 (3+2+1).
- Traffic Type B gets one-third of the total bandwidth (20 Mbps), in line with its weight.
- Traffic Type C, as the lowest priority, receives the remaining one-sixth of the bandwidth (10 Mbps).

The WFQ algorithm dynamically adjusts the queue servicing based on the traffic flow and its assigned weight, thus managing network resources efficiently and maintaining the Quality of Service (QoS) for different types of traffic.

Benefits of WFQ:

- Fairness: WFQ ensures that all traffic flows are treated fairly in terms of bandwidth allocation, preventing any single flow from dominating the network resources.
- Quality of Service (QoS): It supports Quality of Service by allowing priority traffic, such as voice or video, to be allocated more bandwidth, ensuring smoother transmission with minimal latency or jitter.
- Flexibility: WFQ is highly flexible and can be adjusted to meet the specific needs of different types of network traffic.
- Efficient Utilization of Bandwidth: It optimizes the usage of available bandwidth by dynamically adjusting the allocation based on the flow weights.
- Congestion Management: WFQ can help manage network congestion by allocating bandwidth in a controlled manner to different traffic types.

Challenges and Considerations of WFQ:

- Complexity: WFQ is more complex to implement and manage compared to simpler queuing mechanisms like First-In-First-Out (FIFO).
- Resource Intensive: It requires more processing power and memory to monitor and manage the traffic flows and their respective weights.
- Weight Assignment: Determining the appropriate weights for different traffic types can be challenging and requires a deep understanding of the network's traffic patterns.
- Dynamic Traffic Patterns: In networks with highly dynamic traffic patterns, maintaining optimal performance with WFQ can be difficult as the relative importance of flows may change rapidly.
- Scalability Issues: In very large and complex networks, the overhead of implementing and maintaining WFQ can be significant.
- Latency for Low Priority Traffic: While WFQ ensures fairness, low-priority traffic may experience higher latency during times of congestion.

In summary, while WFQ offers several advantages in managing network traffic efficiently and fairly, it also brings challenges in terms of complexity, resource requirements, and the need for careful configuration and management. The decision to use WFQ should be based on a thorough analysis of network requirements and traffic patterns.

WFQ in Action (real world use cases):

Weighted Fair Queuing (WFQ) is employed in various real-world scenarios, particularly in network traffic management and Quality of Service (QoS) optimization. Here are some of its key use cases:

1. **Internet Service Providers (ISPs):** ISPs use WFQ to manage bandwidth allocation among different

customers or types of services. For instance, higher priority might be given to business customers or real-time services like VoIP and video conferencing.

2. **Corporate Networks:** In corporate settings, WFQ is used to prioritize critical business applications over less critical traffic, ensuring that essential services like ERP systems and video conferencing get the necessary bandwidth.
3. **Data Centers:** WFQ aids in managing traffic flow within data centers, especially for balancing loads between servers and ensuring efficient data transfer across the network.
4. **Wireless Networks:** Mobile and wireless network operators use WFQ to manage bandwidth among users and applications, prioritizing services like emergency calls or real-time video streaming.
5. **Streaming Services:** WFQ can be used by streaming platforms to prioritize traffic and ensure smooth streaming experiences, especially when network resources are constrained.
6. **Voice over Internet Protocol (VoIP):** WFQ is crucial in VoIP applications to ensure voice packets are prioritized, minimizing latency and packet loss for clear voice transmission.
7. **Cloud Computing:** Cloud service providers employ WFQ for managing network traffic to and from cloud resources, ensuring fair usage and optimal performance for all users.
8. **E-Commerce Platforms:** For e-commerce platforms, WFQ helps in prioritizing critical transactions and user interactions, especially during high traffic periods.
9. **Online Gaming:** Online gaming platforms use WFQ to prioritize game traffic to ensure low-latency and high-quality gaming experiences.
10. **Video Surveillance Systems:** In video surveillance, WFQ helps in prioritizing video feed traffic over other network uses, ensuring real-time and uninterrupted video streaming.

These use cases demonstrate WFQ's flexibility and effectiveness in managing diverse traffic types and ensuring that high-priority tasks are serviced appropriately in various network environments.

3.Rate Monotonic Scheduling

Rate Monotonic Scheduling (RMS) is a fixed-priority algorithm predominantly used in real-time operating systems for scheduling periodic tasks. In RMS, tasks are assigned priorities based on their request rates (or frequency of execution); the task with the shortest period (or the highest frequency) receives the highest priority. RMS operates on the principle that shorter tasks are more critical, and hence, should be executed first. It’s a preemptive scheduling algorithm, meaning a higher priority task can interrupt a lower priority task.

Let's consider three periodic tasks with different periods and execution times.

- Task A: Execution Time = 1 unit, Period = 4 units
- Task B: Execution Time = 2 units, Period = 6 units
- Task C: Execution Time = 3 units, Period = 8 units

In RMS, the task with the shortest period gets the highest priority. Thus, Task A has the highest priority, followed by Task B, and then Task C.

Time Units	Task Scheduled
0-1	Task A
1-3	Task B
3-6	Task C
6-7	Task A
7-9	Task B
9-12	Task C
12-13	Task A
….	….

- In the first time unit, Task A is executed as it has the highest priority.
- In the next two time units, Task B is executed. Although Task C is also ready, Task B has a higher priority.
- Then Task C is executed for 3 units of time as it’s the only task remaining.
- At time unit 6, Task A is ready again (as its period is 4 units), so it preempts Task C and is executed for one unit.
- The cycle repeats based on the periods of the tasks.

This table demonstrates how RMS schedules tasks based on their periods. It ensures that tasks with shorter periods (and hence higher priorities) are executed first. RMS is optimal for systems where all tasks are periodic, and their execution time is always less than their periods. This example assumes no other overheads like context switching time.

Benefits of RMS:

1. **Simplicity and Predictability:** RMS is straightforward and easy to implement. The fixed priority assignment simplifies the design of real-time systems.
2. **Optimality for Preemptive Systems:** For a set of periodic tasks with static priorities, RMS is optimal. This means no other static priority scheduling can meet deadlines for a given task set if RMS cannot.
3. **Determinism:** RMS provides deterministic behavior, essential in real-time systems where understanding how the system will behave in any situation is crucial.
4. **Efficient for Hard Real-Time Systems:** It’s well-suited for hard real-time systems where missing a deadline could lead to system failure or catastrophic results.

Challenges and Considerations of RMS:

1. **Limited to Periodic Tasks:** RMS is most effective for systems with entirely periodic tasks. It's less suitable for a periodic or sporadic tasks.
2. **Deadlines Equal to or Less Than Periods:** RMS assumes that the deadline of a task is equal to or less than its period, which may not always be the case in complex systems.
3. **Priority Inversion:** Lower priority tasks holding resources needed by higher priority tasks can lead to

priority inversion, though this can be mitigated by protocols like Priority Inheritance.

4. **Utilization Bound:** The utilization of CPU for 'n' tasks in RMS is bounded, which may lead to underutilization of the processor.
5. **Difficulty in Priority Assignment for Mixed Systems:** In systems where periodic, aperiodic, and sporadic tasks coexist, assigning priorities can be challenging.
6. **Task Dependency Handling:** RMS doesn't inherently handle task dependencies, which can be an issue in systems where tasks are interdependent.
7. **Scalability Issues:** As the number of tasks increases, managing and maintaining RMS can become more complex.
8. **Not Ideal for Soft Real-Time Systems:** For systems where deadlines are important but not critical, RMS might be overly rigid, leading to inefficient processing.

In summary, while RMS is highly effective for certain types of real-time systems, it has limitations that must be considered during system design, particularly regarding task types, frequency, and interdependencies. Understanding these challenges is crucial for effectively implementing RMS in appropriate real-time applications. Let’s take a look at some of the real world implementations of RMS.

RMS in Action (real world use cases):

Some of the key use cases include:

1. **Automotive Systems:** In car control systems, RMS is used for scheduling tasks like engine monitoring, fuel injection control, and braking systems, where timely execution is critical for safety and performance.
2. **Avionics and Aerospace:** RMS is employed in avionics for scheduling tasks in flight control systems, such as navigation, communication, and system monitoring, to ensure smooth and safe operation of aircraft.
3. **Industrial Automation:** In automated manufacturing and processing plants, RMS schedules tasks in robotic arms, conveyor belts, and other machinery to optimize production efficiency and safety.
4. **Consumer Electronics:** In devices like smart TVs or gaming consoles, RMS can be used to manage various periodic tasks like streaming, rendering, and user input processing to ensure a seamless user experience.
5. **Telecommunications:** RMS is applied in network routers and switches for scheduling tasks related to data packet processing and transmission, ensuring efficient and timely data flow.
6. **Medical Devices:** Critical healthcare devices like pacemakers and ventilators use RMS to ensure that vital functions are executed at regular intervals for patient safety.
7. **Embedded Systems:** In embedded systems, such as home automation or security systems, RMS is used to manage periodic tasks like sensor data processing and actuator control.
8. **Real-time Operating Systems (RTOS):** Operating systems designed for real-time applications often implement RMS for managing system-level tasks efficiently.

In these applications, RMS is chosen for its predictability and optimality in handling periodic tasks with strict timing constraints, ensuring that all tasks are executed within their defined time periods.

Comparing scheduling & resource allocation

If we need to make a choice on which method to simply based on assignments:

- **EDF (Earliest Deadline First)**: Best suited for real-time systems where tasks have varying deadlines. It dynamically prioritizes tasks based on their deadlines, aiming to minimize deadline misses. However, it can be complex and has higher overhead.
- **WFQ (Weighted Fair Queuing)**: Designed for network environments, it allocates bandwidth fairly among flows. It's effective in handling diverse and dynamic network traffic but can be less efficient with bursty traffic patterns.
- **RMS (Rate-Monotonic Scheduling)**: Ideal for simple real-time systems with fixed, periodic tasks. It's easy to implement and predictable, but not suitable for tasks with irregular periods or varying execution times.

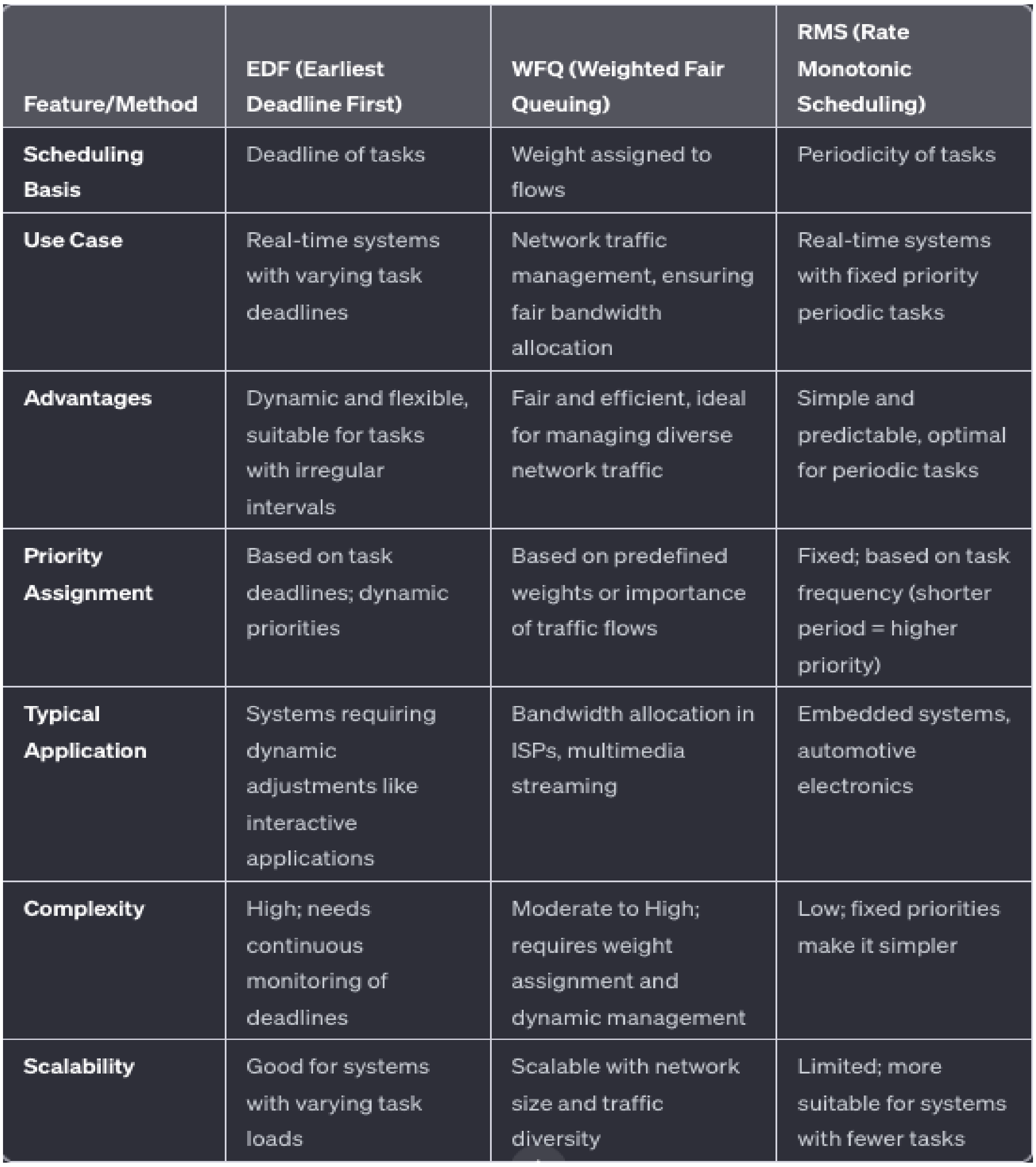
However, for a complete method study, you need to consider, all of these:

1. **Real-Time Performance Metrics**:
   - **Deadline Miss Rate**: Frequency of missing deadlines under each algorithm.
   - **Response Time:** Time taken from task initiation to completion.
   - **Jitter:** Variability in response time, important for time-sensitive applications.
2. **Resource Utilization**:
   - **CPU Utilization:** How effectively each algorithm utilizes CPU resources.
   - **Memory Overhead:** Amount of memory required for scheduling tasks.
3. **Scalability and Adaptability**:
   - **Performance Under Load:** How each algorithm performs under high load conditions.
   - **Adaptability to Changing Workloads:** Ability to handle sudden changes in task volume or priorities.
4. **Fault Tolerance and Robustness**:
   - **Behavior Under Failure:** How each algorithm copes with component failures or unexpected system behavior.
   - **Recovery Mechanisms:** Ability to recover from missed deadlines or errors.
5. **Implementation Complexity**:
   - **Ease of Implementation:** Complexity involved in implementing each algorithm in a real-world scenario.
   - **Maintenance Requirements:** Ongoing maintenance efforts required.
6. **Quality of Service (QoS) Metrics** (especially relevant for WFQ in network scenarios):
   - **Bandwidth Allocation Efficiency:** How effectively the algorithm allocates bandwidth among different flows.

- **Packet Loss Rate:** Frequency of packet loss in network traffic.

7. **Energy Efficiency**:
    - **Power Consumption:** How much power each scheduling algorithm consumes, particularly important in battery-operated or energy-sensitive environments.
8. **Predictability**:
    - **Determinism:** How predictable the behavior of each algorithm is, crucial for hard real-time systems.
9. **Applicability to Different Domains**:
    - **Versatility:** How well each algorithm can be adapted to different application domains (e.g., embedded systems, telecommunications, cloud computing).
10. **Cost Analysis**:
    - **Implementation Cost:** Resources and time required for implementation.
    - **Operational Cost:** Ongoing costs associated with the operation of each algorithm.
11. **User and Industry Acceptance**:
    - **Popularity:** How widely each algorithm is used in the industry.
    - **User Satisfaction:** Feedback from users or system administrators regarding each algorithm's performance**.**
12. **Compliance and Standards**:
    - **Conformance to Standards:** How each algorithm aligns with industry standards and regulations.

Let's try to get some of these based on the arguments presented in the paper listed:



Feature/Method	EDF (Earliest Deadline First)	WFQ (Weighted Fair Queuing)	RMS (Rate Monotonic Scheduling)
Scheduling Basis	Deadline of tasks	Weight assigned to flows	Periodicity of tasks
Use Case	Real-time systems with varying task deadlines	Network traffic management, ensuring fair bandwidth allocation	Real-time systems with fixed priority periodic tasks
Advantages	Dynamic and flexible, suitable for tasks with irregular intervals	Fair and efficient, ideal for managing diverse network traffic	Simple and predictable, optimal for periodic tasks
Priority Assignment	Based on task deadlines; dynamic priorities	Based on predefined weights or importance of traffic flows	Fixed; based on task frequency (shorter period = higher priority)
Typical Application	Systems requiring dynamic adjustments like interactive applications	Bandwidth allocation in ISPs, multimedia streaming	Embedded systems, automotive electronics
Complexity	High; needs continuous monitoring of deadlines	Moderate to High; requires weight assignment and dynamic management	Low; fixed priorities make it simpler
Scalability	Good for systems with varying task loads	Scalable with network size and traffic diversity	Limited; more suitable for systems with fewer tasks

4.Conclusion

In conclusion, the comparison between Earliest Deadline First (EDF), Weighted Fair Queuing (WFQ), and Rate-Monotonic Scheduling (RMS) reveals that each algorithm has distinct characteristics and is suited to specific scenarios. EDF excels in environments where task deadlines are critical and varied, offering dynamic prioritization to minimize deadline misses, albeit at the cost of higher complexity and resource overhead. WFQ stands out in network traffic management, ensuring fair bandwidth distribution and efficiently handling diverse traffic, though it may struggle with bursty traffic patterns. RMS, with its simplicity and predictability, is ideal for systems with fixed, periodic tasks, particularly in hard real-time environments, but it falls short in handling tasks with irregular periods or execution times.

The choice of scheduling algorithm should, therefore, be guided by the specific requirements of the system in question, including factors like real-time performance, resource utilization, scalability, and the nature of the tasks or network traffic involved. Understanding the strengths and limitations of each algorithm is crucial for system designers and network administrators to optimize performance, reliability, and efficiency in their respective domains.

References

[1] Brandt and M. Brandt. On the M(n)/M(n)/s Queue with Impatient Calls. *Performance Evaluation*, 35:1–18, 1999.
[2] L. Georgiadis, R. Guérin, V. Peris and K. Sivarajan, "Efficient network QoS provisioning based on per node traffic shaping", *Proceedings of IEEE INFOCOM 96*, pp. 102-110, 1996.
[3] G. Quadros, A. Alves, E. Monteiro and F. Boavida, "How unfair can weighted fair queuing be?," Proceedings ISCC 2000. Fifth IEEE Symposium on Computers and Communications, Antibes-Juan Les Pins, France, 2000, pp. 779-784, doi: 10.1109/ISCC.2000.860738.
A. Atlas and A. Bestavros, "Statistical rate monotonic scheduling," Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No.98CB36279), Madrid, Spain, 1998, pp. 123-132, doi: 10.1109/REAL.1998.739737.
[4] Lui Sha, R. Rajkumar and S. S. Sathaye, "Generalized rate-monotonic scheduling theory: a framework for developing real-time systems," in Proceedings of the IEEE, vol. 82, no. 1, pp. 68-82, Jan. 1994, doi: 10.1109/5.259427.
[5] Alan A. Bertossi, Andrea Fusiello, Rate-monotonic scheduling for hard-real-time systems, European Journal of Operational Research, Volume 96, Issue 3, 1997, Pages 429-443, ISSN 0377-2217, https://doi.org/10.1016/S0377-2217(97)83306-1.
[6] Casavant T. L., Kuhl J. G., A Taxonomy of Scheduling in General Purpose Distributed Computing Systems, "IEEE Transactions on Software Engineering", 1988, 14(2), pp. 141-154.
[7] Cheng S., Stankovic J.A., Ramamritham K., Scheduling Algorithms for Hard RealTime Systems: A Brief Survey, (In:) Hard Real-Time Systems: Tutorial,Stankovic J.A., Ramamritham K. (eds.), IEEE,1988, pp. 150-173