Deferrable Server Algorithm

Kashif Raza
Hochschule Hamm-Lippstadt
Bachelor of Engineering - Electronic Engineering
Lippstadt, Germany
kashif.raza@stud.hshl.de

***Abstract*—Real-time systems require precise and predictable responses within strict time limits. Real-time systems use scheduling algorithms to ensure that tasks are run on time. This paper presents the Deferrable Server (DS) Algorithm, an organized method that permits tasks to be deferred and executed in a more convenient and flexible manner in real-time embedded systems. Unlike most existing scheduling algorithms, which serve either aperiodic or periodic tasks, the DS algorithm serves both effectively. It prioritizes aperiodic tasks until periodic tasks are about to miss their deadlines. This leads to an enhanced response time for the aperiodic tasks without delaying the hard deadlines for periodic tasks. Moreover, this paper presents an in-depth simulation study of its superior performance compared to traditional background and polling methods. An accompanying UPPAAL model is provided to validate the theoretical concepts and practical implications of using deferrable servers in real-time environments.**

***Index Terms*—Deferrable server, aperiodic tasks, periodic tasks, scheduling, real-time systems, resource allocation, UPPAAL modeling, response times.**

I. INTRODUCTION

The architecture of deferrable server is developed in a way that allows the execution of tasks at an optimal time while meeting its expected deadlines. In general, an architecture that creates a flexible way to handle the task. A task queue is also used to store tasks that may not yet have arrived at their time of execution. A deferrable server is one that can handle a large number of connections at the same time without affecting the tasks' performance. The server accomplishes this by deferring the execution of each connection until necessary, as opposed to executing all the incoming tasks immediately [1]. The DS algorithm keeps periodic and aperiodic tasks apart for their execution. This is in contrast to the traditional ways, where it was believed that the execution of non-critical aperiodic tasks may be handled as a background process, or the polling/Time Division Multiplexing approach may be used for very stringent timing requirements. The DS algorithm assimilates non-urgent aperiodic tasks into a unified approach. This way, the periodic task may be delayed so as to provide a better response time for the aperiodic task without losing any periodic task. This DS algorithm allows very robust solutions for modern real-time systems, providing a high performance with predictable behavior and ease of implementation. It is very effective in situations where there are many connected clients and high-order coexistence. Since requests are queued and executed serially as resources are available, the deferrable server can handle the large amount of traffic smoothly, without blocking or delaying any of the requests. This novel approach allows deferrable servers to handle heavy traffic loads efficiently, which ensures timely and reliable task execution in a large variety of applications. Although one will typically find such servers on a web server, where there may be many concurrent requests for web pages, images, video, and so on, deferrable servers can also be applied to email servers and file servers. Notable examples of deferrable servers include the Nginx web server and the Event Machine library for Ruby, both of which are capable of managing thousands of concurrent connections efficiently [2].

II. DEFERRABLE SERVERS: CONCEPTS AND CHALLENGES

A deferrable server is a specialized scheduling mechanism designed to handle aperiodic tasks in real-time systems. It is defined by two parameters: the server capacity (Cs) and the server period (Ts). The first of these parameters is the maximum amount of time that the server can spend in a period in order to serve the aperiodic tasks that have been released. The second parameter defines the period at which the server capacity is restored. At the beginning of each period Ts, the server capacity is reset to Cs, and the server is now available to serve new aperiodic tasks. The defining characteristic of the deferrable server is the way it can restore capacity so that aperiodic tasks that arrive later in a period can be quickly served. Other servers, such as the polling server, would waste service capacity by consuming some of its capacity in periods where no aperiodic task arrives. The deferrable server defers its usage to the last possible moment, thus saving the server's capacity for use when it is needed without losing the ability to accommodate a periodic task when the aperiodic tasks can be served. Deferrable or elastic servers were developed as a new server architecture that can handle resources dynamically. This approach allows for maximizing the system productivity and efficiency of used resources. As opposed to traditional fixed-capacity servers, which lead to underutilization during off-peak hours and overutilization during peak hours, deferrable servers dynamically change their performance levels based on

workload and resource availability in real-time. Servers can either flex their performance capabilities or go into a low-power state to reduce energy consumption and costs during off-peak hours when there is less demand [3]. In reverse, when the workload increases, deferrable servers can call up other needed resources or scale up the available ones to accommodate the rise in demand. The innovation of deferrable servers answers a very significant requirement in computer environments today: the workloads' variability and resource use shared utilization. In this way, by dynamically allocating them according to the requirements of the workloads, the organization can obtain maximal use of resources, eliminating wastage and inefficiency. This, therefore, leads to not only cost savings but also environmental sustainability by reducing energy use and carbon emissions. Resource provisioning in a cloud computing environment is shared across different users and applications. This offers lucrative advantages in the case of deferrable servers: efficient resource allocation and workload scheduling provide users with more equitable access to computing resources without compromising performance or responsiveness. Moreover, the scalability and flexibility of deferrable servers make them well-suited for handling the diverse and unpredictable workloads characteristic of cloud-based applications. The use of deferrable servers is associated with a large number of challenges, especially in the area of task management and scheduling within real-time systems. In nature, these challenges cut across important areas that need maximum attention and solutions that are sophisticated to achieve overall optimal performance and efficiency. Task prioritization, therefore, becomes an important step for defining the priority levels of tasks within the system so that higher-priority tasks are executed in a timely fashion to meet crucial deadlines [1]. Establishing rules for preemption and deferral becomes crucial in dynamically managing task execution so that higher-priority tasks are not delayed. On the other hand, developing scheduling policies that balance task deadlines and priorities is important for efficient resource utilization and timely task execution. Effective resource allocation is therefore important in avoiding conflicts and maximizing the use of shared resources, such as CPU time and memory. Timing constraints should be incorporated into the scheduling function, serving the purpose of task completion within their stipulated time frames, thus achieving their real-time mandate. Finally, there is a requirement for mechanisms that can facilitate informed decisions regarding the scheduling and service of enhancing maximum responsiveness of tasks to aid in the propagation of priorities. The realization of these challenges will thus require sophisticated scheduling algorithms and techniques to ensure the success of the execution of tasks in dynamic and challenging environments.[2] By overcoming the above challenges, it will now be possible for organizations to fully utilize their deferrable servers to achieve optimal performance and efficiency for all the computing infrastructures within the organization.

III. DEFERRABLE SERVERS: ROLE IN REAL-TIME SYSTEMS

The Deferrable Server (DS) scheduling algorithm is one of the important real-time scheduling approaches that support adaptiveness and dynamic allocation of computational resources within a system. Drawn from real-time systems, DS dynamically allocates the CPU time in response to the variable demands of tasks. The algorithm classifies tasks as a critical segment and a non-critical segment. Critical tasks have a hard deadline; they have to be done within a certain time limit. On the other hand, non-critical tasks have looser timing constraints and can be postponed [2]. The DS ensures that there is always time allocated to the heavy tasks so that they are serviced in time. The remaining CPU time is used to service non-critical tasks. The algorithm is a preemptive one; it interrupts the non-critical tasks as the critical tasks arrive. This feature of the algorithm ensures that important tasks are carried out in time and enhances the performance and reliability of the system [2]. One of the key features of DS is how periodic server tasks are managed. If the server is idle and there are no aperiodic tasks pending, it defers its execution and becomes active when an aperiodic task arrives. This is more a mechanism similar to RMS, in that it refreshes the remaining computation time of the server at the beginning of each period, so that aperiodic tasks can always be serviced immediately. Much more interestingly, the DS technique proposed extends the thinking involved in the simple RMS algorithm. DS is based on the work of Liu and Strosnider, in which tasks are scheduled inversely to their periods. DS goes further than this by supplying abstractions specifically designed to schedule aperiodic tasks with short response times [4],[5]. DS supplies a flexible real-time resource allocation policy, where critical tasks are accommodated in time, leading to better utilization of resources. This incorporation of dynamic scheduling with quick response outlines the vital characteristic of DS in modern real-time computing.

IV. DEFERRABLE SERVERS: ALGORITHM DESCRIPTION

The Deferrable Server (DS) algorithm is used for aperiodic task management in a real-time system with tight periodic deadlines. The main parameters are the server's capacity (Cs) and its period (Ts). The DS has a fixed priority, which is higher than the low-priority periodic tasks but lower than the high-priority periodic tasks. Aperiodic tasks are queued upon arrival. If there is enough capacity on the server, the aperiodic task is executed by the server. In this case, the server capacity is reduced by the amount of the executed task, consuming a part of Cs. If an aperiodic task's service demand exceeds the remaining capacity, service is resumed in the next period, beginning with the full available capacity. Before the beginning of each period, Ts, the capacity of the server is restored to the value of Cs, and the aperiodic task can be handled in the following periods without violating any deadline.[1] The Deferrable Server (DS) algorithm extends the basic concepts of the Rate Monotonic Scheduling (RMS)

algorithm to support the scheduling of aperiodic tasks with stringent latency requirements. The RMS algorithm, as established by Liu and Layland, assumes several key properties of a periodic task set.[5]

1) All periodic tasks τ have fixed periods T and constant, known execution times C. Tasks are ready for execution at the beginning of each period.
2) Due dates (D) are at the end of each task's period.
3) Tasks are independent; they do not synchronize with or block each other, and do not suspend themselves.
4) All overhead due to scheduling, context switching, etc. is assumed to be negligible.[3]

In RMS, tasks are assigned priorities inversely proportional to their periods; shorter periods correspond to higher priorities, with ties broken arbitrarily. Although this algorithm is effective for scheduling periodic tasks, it does not provide an effective mechanism for handling aperiodic tasks. To better facilitate this, the Polling Server () was introduced, followed by the more powerful DS algorithm.[3]

The Polling Server (ps) algorithm manages aperiodic tasks in real-time systems by operating as a periodic task with a fixed period Tps and execution time Cps. At initialization, these parameters are set to determine the frequency and duration of the ps's activity. The Polling Server (PS) works at a fixed priority level, higher than lower-priority periodic tasks but lower than higher-priority periodic tasks. This priority assignment allows the ps to preempt lower-priority tasks as needed to service aperiodic tasks in a timely manner, while not allowing the critical tasks to be too severely delayed. At the beginning of each period, the PS checks if any aperiodic tasks are waiting to be serviced and begins to service them, for up to the maximum amount of processing time, Cps . If necessary, it will continue to service aperiodic tasks after the execution of higher priority tasks has completed. The ps does not carry over unused capacity at the end of the period; therefore, it cannot bank any unused execution time for future use. Aperiodic tasks that become eligible to execute while the ps is inactive or while the server is already running at its maximum processing capacity are queued and processed as a background priority task, but are ensured service in the next period in which they become eligible to execute. This analytic method provides for the aperiodic task handling to be easily incorporated within the real-time scheduling model.[3]

The Deferrable Server (DS) algorithm plays a key role within real-time systems, laying out a systematic way of aperiodic task handling without compromising the timely dispatch of periodic tasks. Unlike its cousin, the Polling Server (ps), the DS is designed with a significantly different set of features geared towards making dynamic task scheduling possible within such systems. Underlying its design principle, the DS reeks of periodicity, with a specifically designed period Tds and capacity Cds . The period and capacity determine not only the frequency with which the DS activates but also the number of aperiodic task arrivals that it can handle. Unlike the ps, whose execution window is strictly enforced, the DS has the unique capability of maintaining its execution capacity throughout its period, saving any unused time for later aperiodic tasks. During operation, the DS employs a prioritization scheme, often dictated by the rate monotonic algorithm, relative to other periodic tasks in the system. This prioritization is of great importance, as it determines the DS's ability to preempt low-priority tasks from urgent aperiodic task service while avoiding excessive delay for high-priority periodic tasks. With the DS assigned the highest priority, real-time systems can therefore ensure that any aperiodic alerts are serviced promptly and that the soft-deadline aperiodic tasks are highly responsive, maintaining other critical system requirements.[3] Yet, priority assignment to the DS comes with its own rubs. At priority levels other than the highest, the DS can have difficulty delivering timely service to aperiodic tasks, as its capacity is vulnerable to preemptions by higher-priority tasks—even when aperiodic tasks are ready for service. It is for this reason that real-time systems opt to have the DS set at the highest priority, thereby preventing such interruptions and delivering guaranteed service to aperiodic tasks. Not to mention, the DS differs from classical periodic tasks in its demand-driven behavior. In contrast to other tasks with rigidly predefined start times, a DS can execute dynamically, triggered by aperiodic task arrivals, at any instant within its period. This feature, along with the fact that it commonly has the highest priority, means it is the most responsive critical component and becomes the cornerstone in hard real-time environments. However, the unique ability of the DS to postpone the time of execution to the later part of its period invokes the violation of most basic scheduling assumptions, which leads to a break from the traditional schedulability analysis such as that by Liu and Layland [5]. It is in this sense that the DS prompts the development of novel analytical frameworks tailored for its operational peculiarities, in a consistent stride toward the pivotal role of modern real-time systems. Essentially, the Deferrable Server (DS) algorithm is a dynamic, critical tool in real-time system design that allows a fine line between periodic and aperiodic task management. With its careful priority handling mechanism, adaptive means of execution, and response to dynamic task arrivals, the DS forms the epitome of efficiency and reliability at the same time, ensuring the proper functioning of critical systems.

V. SCHEDULABILITY ANALYSIS

The schedulability analysis of the Deferrable Server (DS) under the Rate-Monotonic (RM) scheduling algorithm shows dramatic deviations from the basic principles of RM scheduling. RM requires that any periodic task must be executed immediately whenever this task is the highest-priority task ready to run, ensuring that higher-priority tasks preempt lower-priority tasks to meet their deadlines[6]. The Deferrable Server, however, violates this assumption since it defers its execution even when it is the highest-priority task ready to run. Figure.1 clearly shows this behavior of the DS, where the DS, though ready at time $t = 0$, defers its execution until $t = 5$ when the first aperiodic request arrives. The deferred

execution model of the DS causes considerable problems for the RM scheduling framework, particularly affecting the timely execution of lower-priority tasks.[1]

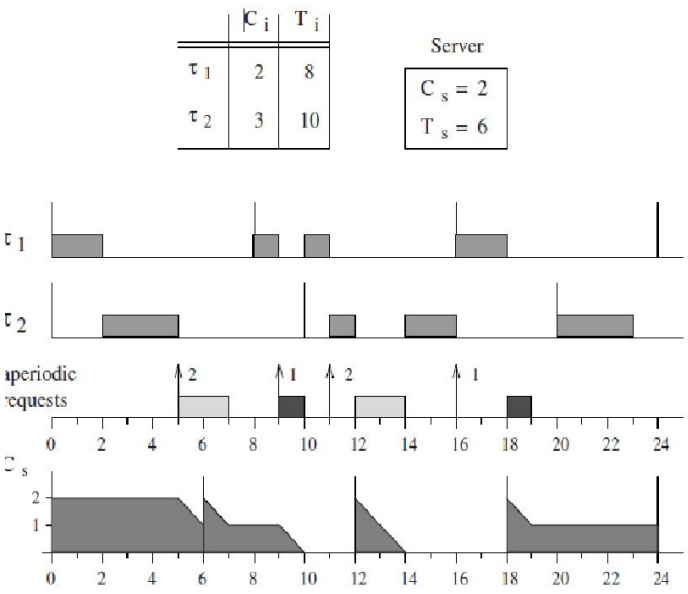

Fig. 1. Example of High-priority DS [1]

One of the primary consequences of the DS's deferred execution is the increased risk of missed deadlines for lower-priority tasks, though originally considered schedulable under RM. For instance, Figure.2 compares the behavior of a periodic task with that of a Deferrable Server, both having the same period and execution time. The periodic task set to be considered is composed of two tasks, τ_1 and τ_2, both with a computation time equal to 2 but with a period of 4 and 5, respectively. Under the RM scheduling algorithm, both tasks are schedulable. As Figure.2(a) shows, they can complete their execution within their respective periods without missing any deadlines.[1]

But when τ_1 is replaced by a Deferrable Server with the same period and execution time, the situation changes dramatically. Figure.2(b) demonstrates this by showing one sequence of aperiodic requests that causes τ_2 to miss its deadline. In particular, at time $t = 8$, instead of executing as it would like a normal periodic task, the DS postpones its execution, saving its capacity for later requests. This creates a series of events in which two successive aperiodic requests arrive between $t = 10$ and $t = 14$. This preserved capacity is used by the DS to serve these requests, preventing τ_2 from executing during this interval. As a consequence, τ_2 is not able to finish its execution by the deadline, i.e., it misses its deadline at time $t = 15$.[1]

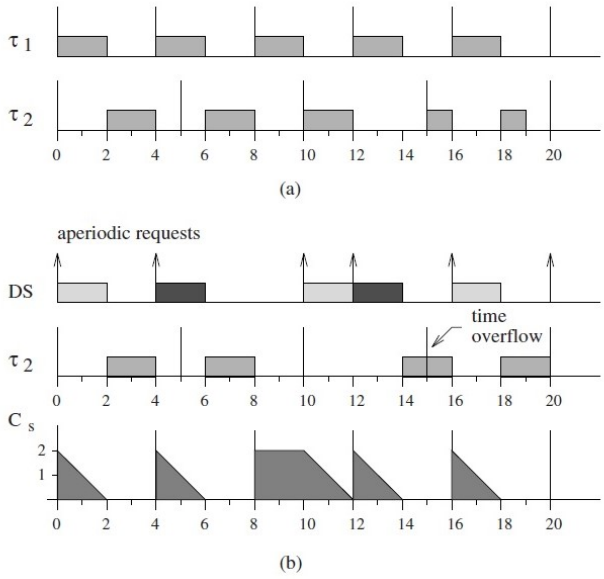

Fig. 2. Example of High-priority DS [1]

This example underscores the invasive nature of the Deferrable Server and its impact on the schedulability of the task set. The deferred execution of the DS can monopolize processor time intended for servicing aperiodic requests, to the detriment of periodic tasks. Hence, in the presence of a DS, a new analysis of the system's schedulability is required since the traditional assumptions described for RM are no longer valid. In particular, the least upper bound of the processor utilization factor needs to be recomputed to take into consideration the behavior of the DS, with the aim of meeting all tasks' deadlines, especially for lower-priority tasks. Although the Deferrable Server introduces flexibility in handling aperiodic requests, its introduction into an RM-scheduled system requires careful consideration. It has been demonstrated that lower-priority tasks can miss their deadlines due to the DS's unique pattern of execution. Thus, an adjusted schedulability analysis is necessary to ensure the system's reliability, so that all timing constraints are met, maintaining the balance and predictability essential for real-time systems.[1]

VI. UPPAAL MODEL FOR DEFFERABLE SERVER

This UPPAAL model shown in Fig.3 represents a Deferrable server system designed to handle both periodic and aperiodic tasks efficiently. The model includes declarations and constants defining channels (arrival and serve), time periods, execution times, and server parameters such as time slice (Ts) and capacity (Cs). The Deferrable Server template initializes with an idle state, transitions to execution when tasks arrive, and replenishes its capacity periodically. The server's execution transitions are guarded by conditions ensuring the server does not operate beyond its capacity or time slice, and synchronization ensures tasks are served correctly. The model

also includes templates for periodic and aperiodic tasks, each with its own execution states and synchronization mechanisms. The periodic task transitions based on its period and execution time, while the aperiodic task synchronizes with the server to execute based on arrival. These components are integrated into the system to manage tasks dynamically.

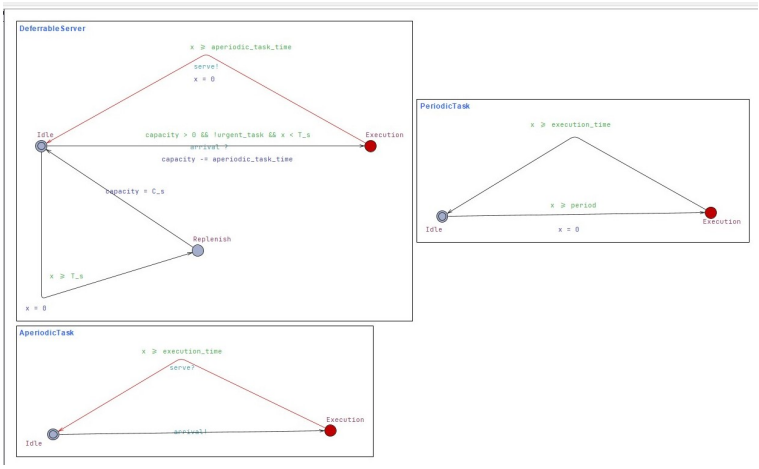

Fig. 3. Defferable server modelling

This sequence diagram in Fig.4 describes how the interaction between the major components of our Deferrable Server system is intended to flow. The sequence of events begins with triggering periodic tasks in accordance with their assigned period and execution time. Each periodic task changes state from idle to execution and negotiates with the deferrable server for service completion. The DeferrableServer makes capacity commitments to periodic tasks in such a way that they are assured of the resources they need to be able to accomplish their timing constraints without failing. Periodically, the DeferrableServer replenishes its capacity, passing through states of idle, execution, and replenishment when necessary. Capacity is periodically replenished so that the server will be assured to have available, at any future, further moments in time, servers to supply eventual further tasks.

As soon as an aperiodic task arrives, it interacts with DeferrableServer—the one that is ready to give enough capacity for this aperiodic task to execute. This inter-synchronization of the aperiodic task arrival and server handling ensures that the aperiodic task is taken care of immediately and effectively without infringing on the timing guarantees of other periodic tasks. The system will balance the periodic and aperiodic tasks, treating each in accordance with its respective time frame. Critical transactions are handled by the DeferrableServer in the most appropriate way, ensuring that the allocative capacity meets the replenishment required to run the system efficiently.

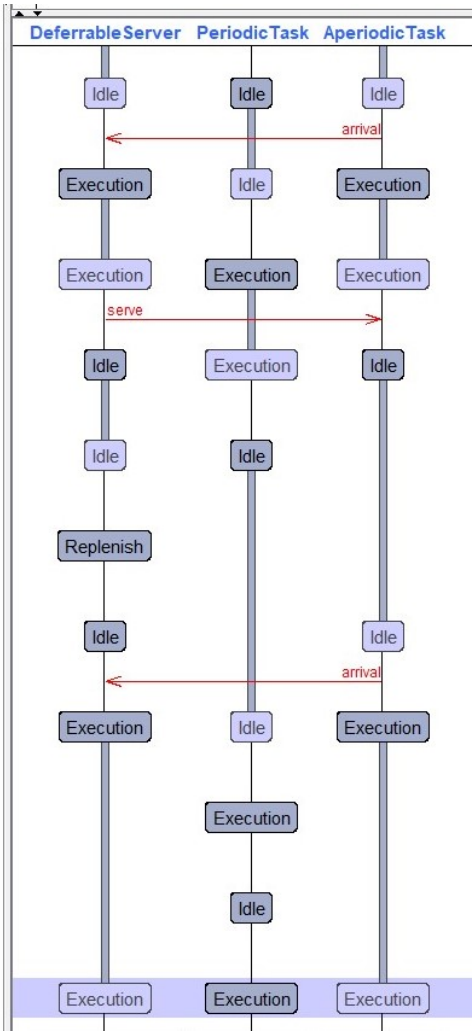

Fig. 4. Defferable server sequence diagram

The verification results in Fig.5 proved that the system satisfied several important properties, which further explains the accuracy and reliability of the deferrable server model. A key query (‘A[] not deadlock‘) ensures the system avoids deadlock, confirming the model’s reliability and robustness. This comprehensive design allows for effective simulation and analysis of the Deferrable Server’s behavior in real-time task management scenarios.

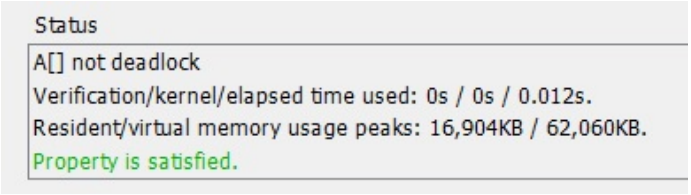

Fig. 5. Model Verification

VII. CONCLUSION

In conclusion, deferrable servers provide a flexible and efficient approach to managing tasks in software systems. The delaying of service execution enables these servers to give full utility to resources, thus improving their overall performance. However, the use of deferrable servers introduces some key concerns. Most importantly, proper task prioritization is needed to meet the runtime requirements of high-priority or time-critical jobs. Effective mechanisms to ensure the proper management of different work priorities must be introduced. Monitoring tools and techniques are also important, apart from that, to monitor and take note of how tasks lined up are progressing and their state. Also, efficient management of the immediate queue is necessary. This would include maximum utilization of resources with limits placed, handling backpressure needs, and optimization of resource allocation to maintain system stability and performance. The addition of a deferrable server can complicate the system and will, without any doubt, be designed, configured, and tested to operate correctly and perform to the top. However, deferrable servers represent a strong base for task organization and execution in a software system. They might have several problems in terms of performance, but well-implemented ones offer huge improvements in system performance, scalability, and user experience, and therefore, they are an indispensable part of current application development.

VIII. REFERENCES

REFERENCES

[1] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer Science & Business Media, September 2011.

[2] Giorgio C. Buttazzo. *Predictable Scheduling Algorithm and Applications*. In Hard Real-Time Computing Systems, pages 119–159. Springer and Business Media, 2005.

[3] The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. Publisher: IEEE.

[4] J. K. Stronsnider, “Highly responsive real-time token rings,” Ph.D. thesis, Carnegie Mellon Univ., Aug. 1988.

[5] C. L. Liu and J. W. Layland, “Scheduling algorithms for multiprogramming in a hard real-time environment,” JACM, vol. 20, no. 1, pp. 46-61, 1973.

[6] J. P. Lehoczky, L. Sha, and Y. Ding, “The rate monotonic scheduling algorithm: exact characterization and average case behavior,” in Proc. 10th IEEE Real-Time Syst. Symp., 1989, pp. 166-171.

IX. DECLARATION OF ORIGINALITY

I hereby declare that I myself have written this paper, and that I have not used any external sources other than those mentioned. Anything borrowed from a different source, whether phraseology or idea, is duly acknowledged. I further declare that this paper has not previously been submitted for any course or examination, in this form or in a similar version.

Kashif Raza
Lippstadt, 01.07.2024