Journal of Computer Science

Original Research Paper

Performance Assessment of CPU Scheduling Algorithms: A Scenario-Based Approach with FCFS, RR, and SJF

Olaa Hajjar, Escelle Mekhallalati, Nada Annwty, Faisal Alghayadh, Ismail Keshta and Mohammed Algabri

Department of Computer Science and Information Systems, College of Applied Sciences, AlMaarefa University, Saudi Arabia

Article history
Received: 22-02-2024
Revised: 23-03-2024
Accepted: 03-05-2024

Corresponding Author:
Ola Faisal Hajjar
Department of Computer Science and Information Systems, College of Applied Sciences, AlMaarefa University, Saudi Arabia
Email: 212220723@student.um.edu.sa

Abstract: This study presents an extensive examination of CPU scheduling algorithms, focusing on the First-Come, First-Served (FCFS), Round-Robin (RR), and Shortest-Job-First (SJF) strategies through a carefully designed scenario-based approach. By deploying a Java-based simulation to dynamically generate random process arrival and burst times, this study simulates a variety of operational conditions to test these scheduling algorithms' adaptability and performance in environments that closely resemble real-world computing scenarios. The research aims to explore the effects of dynamic quantum size allocation on RR scheduling and assess its impact on system performance metrics such as response time and context switching overhead. Through a detailed analysis, this study seeks to provide new insights into the operational efficiency of the FCFS, RR, and SJF scheduling strategies, highlighting their strengths, limitations, and applicability across different computing environments.

Keywords: CPU Scheduling Algorithms, Process Management in Operating Systems, First-Come, First-Served (FCFS) Scheduling, Round Robin (RR) Scheduling, Shortest Job First (SJF)

Introduction

A computer system's program execution and management process comprises various crucial components and scheduling algorithms. Upon a user's application request, the system initially verifies whether it resides in the RAM. If not, it is recovered from persistent storage, such as a hard disk drive. The role of the long-term scheduler is to determine the selection of programs that are loaded into the RAM, while the short-term scheduler is responsible for determining the allocation of Central Processing Unit (CPU) time to these applications. The medium-term scheduler performs process swapping in and out of RAM to manage the allocation of resources for high-priority tasks when the RAM becomes full. The short-term scheduler also referred to as the CPU scheduler, plays a crucial role in deciding which software will utilize the CPU next (Süzen and Taşdelen, 2018).

The optimal distribution of RAM is vital for the proper operation of the computer system as a whole. System performance can be improved by optimizing main memory and virtual memory management. This results in increased CPU utilization, decreased response time, and increased throughput (Gaikwad, 2021). Furthermore, evidence suggests that analysts' evaluations of companies' current activity management and their forecasts of the companies' future operational success are correlated with the utilization of particular RAM (Omar *et al*., 2021).

Many CPU scheduling algorithms play a crucial role in determining which processes are authorized access to the CPU. The referenced algorithms can optimize the computer system's performance by distributing the central processing unit to various duties. These algorithms' essence may be preemptive or non-preemptive (Adeleke, 2022). In addition, practical applications of neural networks within the operating system framework have been suggested, specifically in predicting delay periods, which could enhance the efficacy of CPU utilization (Lee and Chung, 2019).

A process control block is a data structure utilized by an operating system to maintain comprehensive information about individual processes. The aforementioned components comprise the process ID, program counter, process state, priority, inventory of open files and devices, stack section, and static and global variable information (Yamada and Kusakabe, 2008). The collection of data is frequently denoted as the "environment" of the procedure. If an operating system interrupts an ongoing process and transitions to another, it

is imperative to retain the interrupted process's context to facilitate a possible resumption. For this to transpire, the CPU must perform "context switching" between tasks (Yamada and Kusakabe, 2008).

Context switching involves saving the context of the preempted process and loading the context of the new process. It is a fundamental operation in multitasking operating systems and is crucial for efficient process management. Context switch misses, which occur when the required context is not readily available, can impact system performance (Liu *et al*., 2008). Additionally, context switches can affect cache performance, as the assumption of locality may be violated, leading to potential cache misses (Mogul and Borg, 1991). The impact of context switches on cache performance is a critical consideration for system optimization.

Multiprogramming has undergone substantial development since the inception of batch operating systems; contemporary multiprogramming and multiprocessing systems are the result. Earlier versions of batch operating systems had a single level of multiprogramming, which meant that the CPU could execute a single task in a non-preemptive fashion (Barroso *et al*., 2019). The requirement that subsequent processes await the completion of the present process before accessing the CPU led to suboptimal CPU utilization. In addition, the CPU undergoes phases of high activity during process execution, referred to as CPU bursts, which are succeeded by phases of inactivity termed I/O bursts. The CPU is rendered dormant as a result of these I/O bursts, which wastes CPU time (Towsley *et al*., 1978).

Today's multiprogramming and multiprocessing systems have addressed these inefficiencies by allowing multiple processes to be executed in parallel, thereby increasing CPU utilization and efficiency (Barroso *et al*., 2019). In modern systems, the idle time during I/O bursts is utilized by other processes, leading to improved CPU utilization (Towsley *et al*., 1978). Furthermore, to optimize CPU utilization, the principal aim of an optimal system task scheduler is to select processes from the available queue (Barroso *et al*., 2019).

This research paper is notable for its thorough analysis and comparison of three popular CPU scheduling algorithms-FCFS, RR, and SJF-with a particular emphasis on the impacts of dynamic time quantum allocation.

This study employs a scenario-based evaluation methodology to assess the performance of the FCFS, RR, and SJF scheduling algorithms. This tactic has undergone extensive planning and development. This methodology provides a strong framework for study and emulates dynamic computer environments, which are similar to real-world conditions. It uses a broad range of load attributes to analyze algorithms' performance in various scenarios, which sets it apart from standard evaluation methodologies.

The paper's main goal is to investigate the effects of dynamic quantum size allocation in RR scheduling on reaction time and context switching overhead on system performance. This study methodically examines how varying quantum sizes impact the trade-off between decreasing context switching time and satisfying the expectations of a variety of system workloads. The paper offers a thorough analysis of quantum size optimization along with theoretical and empirical insights into reaching the best scheduling efficiency.

This study combines analytical viewpoints with empirical data from well-planned studies to provide new insights into how the FCFS, RR, and SJF algorithms operate. This dual strategy reinforces the study's conclusions and establishes a solid basis for the suggested CPU scheduling techniques.

Through this structured exploration, our paper seeks to contribute valuable knowledge to the field of CPU scheduling, offering a scenario-based assessment that enhances our understanding of algorithmic performance in dynamic computing environments.

CPU Scheduling Criteria

The utilization of CPU scheduling criteria is critical to maximize the efficiency of computer systems. CPU utilization, a crucial metric for assessing system performance, quantifies the duration during which the CPU is engaged in task processing (Xiong and Chung, 2012). Throughput, which refers to the number of processes completed in the time allotted, is an additional critical factor that influences the system's overall efficacy (Avrahami and Azar, 2007). The duration required to complete a specific procedure, known as turnaround time, is a critical metric for evaluating a system's performance and responsiveness (Xiong and Chung, 2012). The duration that a process remains in the ready queue directly impacts the system's overall efficacy and its users' experience (Liang, 2019). Moreover, completion time, denoting the moment a process concludes its execution, is an essential parameter in CPU scheduling that impacts the system's efficacy and performance (Xiong and Chung, 2012; Avrahami and Azar, 2007). In addition, CPU allocation fairness is emphasized, as it guarantees that every process is allocated the CPU equitably and prevents depletion (Khatri, 2016).

In CPU scheduling, response time is the duration required for an application to respond to a specified input or request. The metric is of utmost importance when assessing the effectiveness and efficacy of scheduling algorithms. The importance of response time in time-sensitive and real-time applications is underscored by the strong correlation between response time and scenario urgency (Engström *et al*., 2024). Furthermore, in software-defined wide-area networks, response time optimization strategies, such as those designed

to reduce the delay in processing control messages during switch migration, illustrate response time's significance for system performance (Sahoo *et al*., 2020).

Literature

The First Come, First Served (FCFS) scheduling algorithm, although widely recognized for its simplicity, has several shortcomings. These include prolonged waiting times, decreased throughput, and inefficient resource distribution. On the other hand, recent research has shown that the application of algorithms such as Nudge can improve FCFS stochastically, particularly for light-tailed work size distributions (Van Houdt, 2022). An improvement on the original FCFS method of Zhao and Stankovic (2003) called FCFSI, increases the likelihood that a new job will be added to the available queue (Choi *et al*., 2010). Additionally, Choi *et al*. (2010) showed that the scheduling results from a Mixed-Integer Linear Programming (MILP) scheduling method were better than those from a traditional FCFS scheduler.

Furthermore, in an attempt to increase processing speed and efficiency, the vast majority of research literature has been devoted to CPU scheduling algorithm optimization. Dash *et al*. (2019), for instance, emphasized the importance of refining disk scheduling techniques to increase computational efficiency and performance. Furthermore, to improve scheduling performance in High-Performance Computing (HPC) systems, (Fan *et al*., 2019) stressed the importance of actively optimizing numerous resources in addition to CPUs.

Multilevel queue algorithms are inflexible because processes cannot switch between different accessible queues. Asef *et al*. (2009) suggested Thombare *et al*. (2016) three-level Multilevel Feedback Queue (MLFQ) technique as a remedy for this issue. Any operations in the first queue that take longer than 10 ms are routed to the second queue, where they are sorted using the SJF algorithm before being handled by the RR algorithm, which has a variable time quantum. Should the time quantum be exceeded again, the processes are moved to the third queue, where they are subjected to a similar SJF-RR procedure. Using this approach results in a considerable reduction of the mean waiting time and turnover time when compared to the same configuration using a static time quantum.

The proposed MLFQ algorithm aligns with the findings of Lenka and Ranjan (2012), highlighting the suboptimal performance of traditional algorithms like RR in terms of turnaround time. Additionally, Abirami and Vasudevan (2023) introduced an improved version of MLFQ that uses a modified version of the RR algorithm called Shortest Remaining Burst Round-Robin (SRBRR) to mitigate depletion in reconfigurable computing systems.

Raheja *et al*. (2014) also proposed a multilevel hybrid scheduling technique to maximize processor usage in grid situations. We compare our approach to the idea of multilevel feedback queues. Additionally, Park *et al*. (2022) emphasized how difficult it is to predict queue waiting times due to job features and the scheduling algorithm that was used. This emphasizes how important it is to have dynamic and flexible scheduling procedures.

Shafi *et al*. (2020) used a neural network to determine the best time quantum for the RR algorithm. The authors gathered a body of information by running multiple algorithms with different static time quanta and saving the time quanta that resulted in the quickest turnaround times. When a new program was encountered, the knowledge base was used to predict the time quantum, provided that a similar program was found; if not, the program was given a different time quantum, and the knowledge base was updated accordingly. The authors showed that for time quantum algorithms, dynamic time slice allocation leads to better performance than static time slice implementation of the RR method.

Furthermore, compared to the traditional RR algorithm, Sohrawordi's (2019) experimental results confirm the effectiveness of a dynamic time quantum iteration of the RR CPU scheduling algorithm in solving the fixed time quantum problem and reducing the mean delay time and turnaround time. Furthermore, experimental analysis carried out by Datta (2015) demonstrated that, compared to previous algorithms, their effective RR scheduling algorithm-which included a dynamic time produced better average turnaround and waiting times as well as fewer context transitions.

The RR method is a widely used CPU scheduling mechanism in multitasking operating systems (Sohrawordi, 2019). A measure of central tendency from the previous set of processes, such as the median or arithmetic mean, is used by several implementations of the RR algorithm to calculate the current process's time quantum (Zafar Iqbal *et al*., 2022; Kumar Mishra and Rashid, 2014). However, this assumes that the selected processes have an even distribution of time quanta, which isn't always the case (Faizan *et al*., 2020). Omotehinwa *et al*. (2019) presented a method for figuring out the temporal quantum of RR when the burst timings have an asymmetric distribution. According to Faizan *et al*. (2020), this version of the RR algorithm showed reduced average turnaround and waiting periods compared to alternative iterations, such as NIRR, IRRVQ, and DABRR. This emphasizes how crucial it is to include the burst time distribution in the RR algorithm to increase its effectiveness.

In addition, many improvements and changes have been made to the RR algorithm, such as priority-based versions (Mohanty *et al*., 2011), dynamic time quantum adjustments (Kumar Mishra and Rashid, 2014), and hybrid techniques (Elmougy *et al*., 2017; Adamu *et al*., 2019). By reducing the average turnaround, waiting, and response times, these changes aim to improve the RR algorithm's effectiveness and efficiency. Experimental

investigations show that the suggested algorithms perform better than existing algorithms in terms of average turnaround time, average waiting time, and number of context switches (Datta, 2015; Abdulrahim *et al*., 2014).

When it comes to accurately calculating the CPU burst time of processes that are in the available queue, the SJF algorithm has a challenge (Altman, 1992). To address this problem, Helmy *et al*. (2015) presented a machine-learning approximation solution for the CPU surge time in SJF. A variety of procedures were selected, each with a unique set of characteristics. Additionally, a target variable and a feature vector were given. The processes were divided into distinct training and test sets and the feature vector was a filtered list of process properties. Models were created by applying a variety of machine learning methods, such as K-nearest neighbors and decision trees, using the training dataset. The models' effectiveness was evaluated by comparing the generated models to the test set using measures like the correlation coefficient. The best models were then used to anticipate CPU surge times for new processes and the SJF method was implemented based on these predictions. This approach is in line with the growing body of research that emphasizes nonparametric methods, particularly in the fields of computer science and machine learning (Altman, 1992). The application of machine learning techniques, such as decision trees and K-nearest neighbors, indicates the increasing trend of using these methods to address computational challenges (Altman, 1992; Lundberg *et al*., 2020).

One widely used CPU scheduling approach is Priority Scheduling (PS). It distributes CPU access to processes in the available queue based on their priority level (Xu *et al*., 2024). However, one major worry of PS is that low-priority operations could be stuck in the available queue forever; this is known as "starvation" (Xu *et al*., 2024). Chandiramani *et al*. (2019) proposed a redesigned version of PS that included RR scheduling to address this problem. Their research showed that, in contrast to the original PS, this hybrid technique reduces the impacts of famine and improves mean turnaround and waiting periods (Xu *et al*., 2024). CPU scheduling is essential to multiprogramming operating systems because it allows for the effective distribution of CPU resources among competing programs (Hasan, 2014). Algorithms for CPU scheduling are widely used in operating systems and communication networks, where they greatly increase system efficiency (Hasan, 2014). In addition, CPU scheduling includes deciding in which order to assign processes from the queue to the CPU (Omar *et al*., 2021).

The literature discusses numerous scheduling policies, including PS; last-come, first-served scheduling; FCFS scheduling; and shortest-job-first scheduling (Senan, 2017). According to Sohrawordi (2019), RR is the most common CPU scheduling technique used in multitasking operating systems. Furthermore, studies have compared pre-runtime scheduling to PS (Karapici *et al*., 2015).

A modified version of simple RR scheduling is created by combining PS with RR design. This effectively addresses the issue of low-priority processes being denied resources and improves system performance in general (Putra and Purnomo, 2022). Furthermore, Moharana *et al*. (2018) highlighted the ineffectiveness of scheduling techniques due to the random allocation of virtual CPUs to real CPU cores when context switching occurs.

In the field of CPU scheduling, Generalized Processor Sharing (GPS) is an idealized scheduling approach. It achieves perfect fairness and is used as a standard for evaluating the fairness of other scheduling algorithms (Mostafa and Kusakabe, 2015). In addition, the literature has reviewed the Priority-Based Round-Robin (PBRR) CPU scheduling algorithm, concentrating on the presumptions made during CPU scheduling (Zouaoui *et al*., 2019).

CPU Scheduling Algorithms

First Come, First Served (FCFS)

FCFS is the scheduling mechanism utilized by operating systems' central processing units. The execution of the procedures within this non-preemptive scheduling algorithm is based on the order of arrival. As a result, the CPU is allocated to the first process that arrives, with subsequent processes not receiving the CPU until the earlier processes have finished executing. FCFS is regarded as one of the most basic and uncomplicated scheduling algorithms; furthermore, it is remarkably easy to understand and implement (Sambath *et al*., 2020). However, this may also lead to the convoy effect, wherein shorter processes are forced to wait for longer processes, which can result in delays, inefficiencies in CPU utilization, and resource scarcity (Sambath *et al*., 2020; Stallings, 2012). However, FCFS could be a viable and straightforward alternative when processes have similar durations or the system necessitates a transparent, equitable method that does not involve prioritization (Stallings, 2012). Crucially, FCFS, as with banking clients, operates under the fundamental principle of serving processes in the order in which they enter the available queue. Due to the algorithm's inherently non-preemptive nature, a process that initiates execution remains in the queue until it completes (Silberschatz *et al*., 2018). FCFS is frequently implemented using a First In, First Out (FIFO) queue in which processes are released according to their arrival time (Stallings, 2012). When discussing CPU scheduling, FCFS is occasionally contrasted with PS, RR, and SJF, among others. Despite its user-friendly interface and straightforwardness, FCFS may not consistently generate the most optimal or efficient schedule, specifically regarding average waiting times and turnover times (Abdul Kareem and Hussein, 2022). FCFS and other scheduling algorithms are

extensively utilized in practical applications, specifically in the contexts of grid and cloud computing (Somasundaram and Radhakrishnan, 2009).

Round Robin (RR)

Operating systems commonly employ the RR CPU scheduling method for job management. It is well known for allocating CPU time to programs in an equitable and uncomplicated manner. The RR method assigns a set time unit, known as a time quantum, to each process in a circular queue. When a process is scheduled, the CPU is allocated to it for a single time quantum. If the process has not completed its execution, it is placed at the end of the queue to await its next turn (Tajwar *et al*., 2017). This approach is suitable for time-sharing systems since it ensures that every activity is allocated an equitable amount of CPU time and prevents any single process from monopolizing the entire CPU (Abdulrahim *et al*., 2014).

Crucially, in RRCPU scheduling when a process arrives, it is appended to the end of a circular queue. The CPU scheduler selects the initial process from the queue, establishes a timer to interrupt once the time slice elapses, and allows the process to execute. When the process is completed before its allocated time slice elapses, it is taken out of the queue and the CPU proceeds to the subsequent process. If the allocated time slice elapses before completion, the process is preempted and returned to the end of the queue, while the CPU transitions to the subsequent process in the queue (Tanenbaum and Bos, 2014).

RR offers a notable benefit in terms of fairness, as it ensures that each process receives an equitable allocation of the CPU. Nevertheless, the effectiveness of RR scheduling heavily relies on the duration of the time quantum. A brief time quantum results in frequent context shifts and reduces CPU efficiency, but an extended time quantum transforms RR scheduling into FCFS scheduling, resulting in longer reaction times for shorter operations (Stallings, 2012).

Notwithstanding these limitations, RR continues to be a favored option for time-sharing systems because of its straightforwardness and impartiality in allocating CPU resources among activities. It guarantees that all processes receive regular CPU access without experiencing endless delays, which is vital in interactive systems where a responsive user experience is essential (Silberschatz *et al*., 2018).

The RR scheduling approach is particularly effective when the duration of the CPU burst for each task is not predetermined. Implementing a time quantum restriction on each activity prevents shorter processes from being deprived of resources by longer-running processes (Matarneh, 2009). However, importantly, the RR algorithm may not be suitable for real-time operating systems because it tends to result in extended wait, response, and turnaround times, as well as reduced throughput (Zouaoui *et al*., 2019). In addition, one of the traditional RR scheduling algorithm's significant limitations is the overhead caused by context switching. Context switching refers to the process of saving one process's state and loading another's, which consumes system resources and time (Tajwar *et al*., 2017).

To address the limitations of the traditional RR algorithm and enhance its effectiveness, numerous researchers have proposed enhancements and modifications. Soft real-time systems incorporate priority-based scheduling, intelligent time slice allocation, and dynamic time quantum allocation to handle their requirements (Behera *et al*., 2010; Dash *et al*., 2015; Mohanty *et al*., 2011). In addition, attempts have been made to enhance the system's overall performance by optimizing the RR algorithm by adjusting time slices according to the remaining CPU bursts of active processes (Chhugani and Silvester, 2017).

Shortest Job First (SJF)

The SJF scheduling technique prioritizes the pending process with the shortest execution time (Jeyaprakash and Sambath, 2021). This method has received considerable recognition as being the most efficient for decreasing the average duration needed to accomplish a task (Hu and Li, 2022). SJF aims to enhance throughput by reducing process waiting times by selecting the shortest available assignment for execution (Pon Pushpa and Devasigamani, 2014). According to Jeyaprakash and Sambath (2021), the algorithm operates by prioritizing the execution of the process with the shortest duration, regardless of the order in which the processes were received.

One significant advantage of SJF is its ability to decrease waiting time by prioritizing shorter activities, leading to faster completion of processes (Hashim Yosuf *et al*., 2022). SJF's effectiveness is particularly evident in scenarios where the primary goal is to reduce assignment completion time, as highlighted by Jeyaprakash and Sambath (2021). Furthermore, its simplicity and ease of implementation make SJF an attractive option for various computer systems (Abdul Kareem and Hussein, 2022).

However, SJF does have some disadvantages. A significant constraint is the potential for starvation, as lengthier processes may be compelled to wait indefinitely for the arrival of shorter processes (Younis, 2021). This issue has the potential to create a power asymmetry throughout the execution of procedures, as shorter activities are regularly given higher priority than longer ones (Younis, 2021). In addition, it is imperative to guarantee the accuracy of project length estimations, as inaccurate estimates might result in inefficient scheduling decisions (Mi *et al*., 2012). Therefore, to effectively utilize the SJF algorithm in various computer contexts, it is crucial to possess a thorough understanding of its characteristics and implications.

Experimental Setup

We developed a Java-based simulation designed to dynamically produce random arrival and burst times, tailored to five distinct scenarios, for a collection of eight processes. This innovative approach of incorporating randomness into the simulation of process arrival and execution times serves as a powerful and holistic method for evaluating the FCFS, RR, and SJF scheduling algorithms' performance. By embracing this strategy, we can simulate a wide spectrum of operational conditions that closely mirror the complexities and unpredictability inherent in real-world computing environments.

Furthermore, our use of randomness to model process timings enables us to challenge the scheduling algorithms with scenarios that range from typical to extreme, including the simulation of high-concurrency environments, variable process loads, and unexpected spikes in system demand. This approach not only tests the FCFS, RR, and SJF algorithms' robustness and adaptability but also contributes to the development of more resilient and flexible scheduling solutions capable of accommodating the dynamic nature of computing workloads. The provided description outlines two main components of a program designed to simulate a process scheduling scenario commonly studied in operating systems courses. The program consists of a Main Program flowchart and a generateRandomTimes function flowchart. The following breaks down each part.

Main Program Flowchart

1. **Set numberOfProcesses to 8**: Initialize the **numberOfProcesses** variable.
2. **Generate arrivalTimes**:
   - Call **generateRandomTimes** function with parameters **numberOfProcesses**, 0 and 20.
   - Output of this step is the **arrivalTimes** array.
3. **Generate burstTimes**:
   - Call **generateRandomTimes** function with parameters **numberOfProcesses**, 1 and 10.
   - Output of this step is the **burstTimes** array.

End: The end of the Main Program flowchart

generateRandomTimes Function Flowchart

1. **Start Function**: Indicates the beginning of the **generateRandomTimes** function.
2. **Initialize Array times**: Create an array of integers with a size equal to the **size** parameter.
3. **Create Random Number Generator**: Instantiate a Random object.
4. **Loop from i = 0 to size - 1**:
   - Inside the loop, generate a random number between **min** and **max** (inclusive).
   - Assign this number to **times[i]**.
5. **Return times**: After completing the loop, return the **times** array.
6. **End Function**: Marks the end of the **generateRandomTimes** function

Table shows the process ID, Arrival Time (AT), and Burst Time (BT) for each process in each scenario.

Scenario 1

- Arrival times: Processes generally arrive at a steady rate, starting from time 0. There's a noticeable gap between the arrival of process 1 and 2
- Burst times: Vary significantly, ranging from 1-10 time units. Processes 1, 2, 6, and 8 have high burst times (10 units), suggesting longer processing requirements

Scenario 2

- Arrival times: Processes start arriving later compared to scenario 1, with the first process arriving at time 8. The arrival times are more spread out
- Burst times: Shorter on average compared to scenario 1, with most processes requiring less than 5 time units. This could imply quicker processing for each process

Scenario 3

- Arrival times: Most processes arrive early, within the first 2 time units, indicating a congested start
- Burst times: More varied, ranging from 1-10 time units. Some processes require significant processing time (like processes 1, 2, and 6)

Scenario 4

- Arrival times: Processes start arriving at time 0, similar to scenario 1. The arrival pattern is more evenly spread over time
- Burst times: Generally moderate, with no process exceeding 8-time units. This might lead to a more balanced processing load

Scenario 5

- Arrival times: Processes have later start times compared to other scenarios, beginning at time 6. The arrival times are relatively spread out
- Burst times: Mostly short, with many processes requiring 2-7 time units. This scenario indicates quick processing for most processes

Table 1: Process IDs with arrival (AT) and burst times (BT) across scenarios

Process	Scenario 1 (AT, BT)	Scenario 2 (AT, BT)	Scenario 3 (AT, BT)	Scenario 4 (AT, BT)	Scenario 5 (AT, BT)
1	AT: 0, BT: 10	AT: 8, BT: 4	AT: 1, BT: 10	AT: 0, BT: 4	AT: 6, BT: 1
2	AT: 3, BT: 10	AT: 10, BT: 4	AT: 1, BT: 10	AT: 2, BT: 8	AT: 8, BT: 6
3	AT: 4, BT: 1	AT: 11, BT: 5	AT: 2, BT: 9	AT: 3, BT: 5	AT: 10, BT: 10
4	AT: 9, BT: 4	AT: 13, BT: 8	AT: 2, BT: 7	AT: 5, BT: 4	AT: 11, BT: 4
5	AT: 12, BT: 4	AT: 14, BT: 1	AT: 10, BT: 2	AT: 6, BT: 5	AT: 14, BT: 2
6	AT: 16, BT: 10	AT: 16, BT: 4	AT: 14, BT: 10	AT: 6, BT: 7	AT: 15, BT: 7
7	AT: 18, BT: 7	AT: 18, BT: 5	AT: 15, BT: 8	AT: 10, BT: 6	AT: 15, BT: 10
8	AT: 18, BT: 10	AT: 20, BT: 2	AT: 16, BT: 1	AT: 15, BT: 4	AT: 17, BT: 2

Table 2: Key differences and similarities in arrival and burst times across scenarios

Scenario	Arrival time characteristics	Burst time characteristics	General observation
1	Steady, starting from time 0 with a noticeable initial gap	Ranges from 1-10, with and several longer bursts	Potentially longer waits for of processing
2	Later starts, spread out arrivals	Shorter on average, mostly under 5 units	Quicker processing, less initial congestion
3	Early and congested, most arriving within the first 2 units	Varied, from 1-10, unpredictable delays	Early congestion, varied processing times
4	Early like scenario 1, more evenly spread over time	Moderate, none exceeding 8 units	Balanced processing load
5	Later starts like scenario 2, arrivals spread out	Mostly short, between 2 and 7 units	Quick processing, less congestion

Table shows the key differences and similarities among the five scenarios, focusing on arrival and burst times.

It is important to note here that, in terms of arrival patterns, scenarios 1 and 4 have earlier arrivals, with scenario 1 having a more concentrated arrival pattern. scenarios 2 and 5 have later arrivals. Scenario 3 is unique for its very early and congested arrival pattern. Meanwhile, scenario 1 has longer burst times, indicating potentially longer waits for processing. Scenarios 2 and 5 have shorter burst times, suggesting quicker processing. Scenario 3 shows the most variance in burst times, which could lead to unpredictable processing delays. Scenario 4 strikes a balance with moderate burst times.

Materials and Methods

This section outlines the methodology used to evaluate three CPU scheduling algorithms FCFS, RR, and SJF using a Java-based simulation designed to imitate various operational scenarios. It also describes the technical requirements, such as software and hardware configurations, process generating methodologies, algorithm implementation, and performance measurements, to ensure that the research is clear and reproducible.

Experimental Design

The primary objective of this research was to assess and compare the performance of three CPU scheduling algorithms First-Come, First-Served (FCFS), Round-Robin (RR) and Shortest-Job-First (SJF) under a variety of simulated conditions. The study employed a Java-based simulation to generate dynamic, random process arrivals and burst times across five distinct scenarios, representing varying levels of system load and operational demands.

Simulation Environment

- Programming language: Java
- Software tools: Eclipse IDE for Java developers
- System specifications: The simulation was run on a computer with an Intel Core i7 processor, 16GB Ram, and windows 10 operating system

Description of the Simulation

- Process generation: A total of eight processes were dynamically created for each simulation run. Each process was characterized by two main attributes: arrival time and burst time
- Random time generation: A custom function, generateRandomTimes (intnumberOfProcesses, int min, int max), was implemented to produce random values for arrival and burst times within specified ranges, ensuring variability across simulation runs
- Scenarios setup: Five scenarios were designed to reflect different operational environments:
    - Scenario 1: High burst times with uniform arrivals
    - Scenario 2: Short burst times with staggered arrivals
    - Scenario 3: Mixed burst times with clustered early arrivals
    - Scenario 4: Moderate burst times with evenly spaced arrivals
    - Scenario 5: Short burst times with delayed, spread-out arrivals

CPU Scheduling Algorithms Implementation

- FCFS: Processes were managed in a queue based on their arrival order without preemption
- RR: This algorithm utilized a time quantum; the quantum size was varied in different simulation runs (1, 5, and 10 units) to examine its impact on performance. Processes exceeding their quantum were re-queued
- SJF: Processes were selected based on the shortest burst time, prioritizing shorter tasks to reduce average waiting time

Performance Metrics

The simulation recorded Key Performance Indicators (KPIs) to evaluate the efficacy of each scheduling algorithm:

- Response time: Time from the moment of arrival to the first response
- Turnaround time: Total time from arrival to completion

Data Collection and Analysis

Data collection: The output from each simulation run was automatically logged into a structured format, capturing detailed timing information for each process under each scheduling algorithm.

The study's analysis techniques included a detailed examination of the collected data using descriptive statistics, calculating the average of key performance metrics such as response time and turnaround time, and establishing a baseline understanding of each CPU scheduling algorithm's performance across different scenarios. Graphical representations, such as bar charts, were utilized to visually display the algorithms' comparative and changeable performance, with an emphasis on average response and turnaround times under various operational settings. Furthermore, a comparative study enabled a direct evaluation of the metrics, revealing disparities in performance and providing greater insights into the algorithms' efficiency and efficacy in diverse simulated scenarios.

By detailing these materials and methods, the study aims to provide a clear and comprehensive account of the experimental setup, ensuring that the findings are reproducible and verifiable by other researchers and practitioners interested in CPU scheduling performance analysis.

Results and Discussion

Figure 1 presents a column bar chart comparing the different CPU scheduling algorithms' response times across the five scenarios.

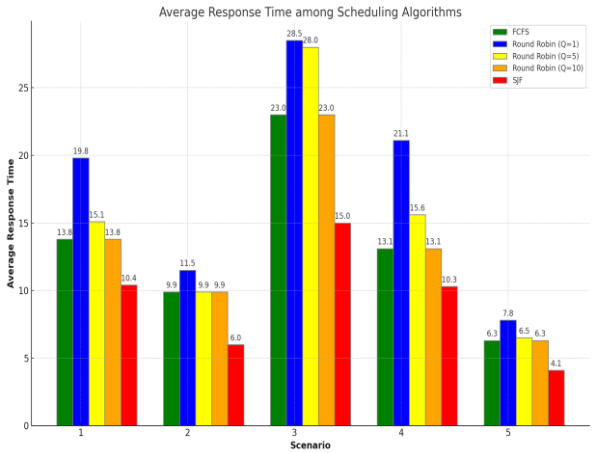

Fig. 1: Average response times of different CPU scheduling algorithms across five scenarios

In the first scenario, we observe a diverse array of process burst times. The SJF algorithm excels in this environment by swiftly completing jobs with shorter durations, thus achieving a reduced overall average response time. The FCFS method shows reasonable effectiveness, as it seldom encounters situations where short processes are delayed by preceding longer ones, with the initial sequence being the only exception. On the other hand, RR scheduling with a quantum of one unit performs poorly. This inefficiency stems from the high cost associated with frequent context switching.

In the second scenario, while the processes' burst times are again quite varied, their arrival times are more clustered. This scenario similarly sees the SJF algorithm outperforming others due to its preference for shorter tasks. The RR strategy with a quantum of 1 remains suboptimal. However, when the quantum is increased to 5 or 10 units, the RR approach begins to mirror the performance of FCFS. This improvement is attributable to the burst times being relatively short and the arrival times showing less variation. In the third scenario, processes arrive in close succession, which inevitably pushes up the average response time for all scheduling algorithms. Yet, SJF continues to distinguish itself by efficiently processing the shorter jobs first.

In the fourth scenario, the trend noticed in previous scenarios persists, with SJF securing the best average response time. Conversely, the RR method with a quantum of 1 exhibits the least desirable performance, a trend consistent with the reasons previously outlined. The fifth scenario is characterized by generally shorter burst times for processes, a condition that is advantageous for all scheduling algorithms. Nonetheless, SJF maintains a performance lead by leveraging the shorter process durations to minimize waiting times.

To encapsulate the findings, the SJF algorithm consistently delivers superior performance across various scenarios. Its strategy of prioritizing shorter processes significantly cuts down on the average response time. The effectiveness of the FCFS method is contingent upon the sequence in which processes arrive; its performance falters when longer processes precede shorter ones. The RR algorithm's efficiency is closely linked to the quantum size selected. A quantum that is too small leads to excessive overhead due to context switching, while a quantum that is too large essentially reduces RR to an FCFS-like operation. In the scenarios considered, smaller quantum sizes have typically yielded poorer results when compared to larger ones due to the high costs of context switching.

In the scenarios presented, the FCFS scheduling method demonstrates varied performance. In scenarios with a mix of short and long process burst times (scenarios 1 and 4) or predominantly short processes (scenario 5), FCFS achieves moderate to good average response times, due to either a large initial process setting a consistent pace or minimal waiting times for most processes. However, its performance drops significantly in scenarios where processes with longer burst times arrive early (scenario 3), as this causes subsequent processes to endure extended waits, thus inflating the average response time.

The RR scheduling method with a quantum of 1 suffers from high context switching overhead, resulting in poor performance across all scenarios. However, as the quantum increases to 5 or 10, RR's performance begins to align with that of FCFS, especially when the processes have shorter burst times (scenarios 2, 4, and 5). This improvement is due to the reduction in context switching frequency. Across all scenarios, SJF stands out as the most efficient algorithm, consistently delivering the best performance by always executing the shortest available process, thereby minimizing the wait times for subsequent processes and, consequently, the average response time. This efficiency is based on the precondition of known burst times for all processes.

Figure 2 offers a graphical representation comparing the average turnaround times of five CPU scheduling algorithms over five distinct scenarios. The following paragraphs detail the outcomes for each scenario and conclude with a comprehensive summary.

In scenario 1, FCFS faces a large process arriving first, but due to the subsequent arrival of smaller processes, the average turnaround time remains moderate. Scenario 2 sees FCFS performing well as the processes are predominantly short, leading to less variability in waiting times. Conversely, scenario 3 is where FCFS struggles the most; early-arriving processes with longer burst times significantly delay later ones, resulting in the highest average turnaround time among the scenarios. Scenario 4 presents a balanced mix of process burst times, allowing FCFS to perform adequately without significant delays. In scenario 5, the shorter burst times overall benefit FCFS, resulting in a considerably low average turnaround time.

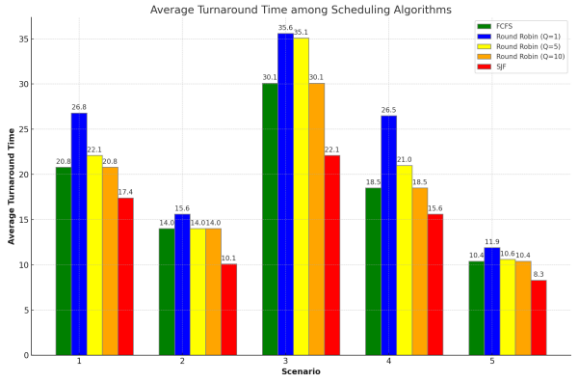

Fig. 2: Average turnaround time of different CPU scheduling algorithms across the five scenarios

RR scheduling in scenario 1 with a quantum of 1 leads to high context-switching overhead and thus the worst performance, but with a quantum of 10, the performance matches that of FCFS by minimizing context switches. In scenario 2, a short quantum again introduces inefficiency; however, increasing the quantum size makes RR's performance equivalent to FCFS thanks to the shorter and more uniformly sized processes. Scenario 3 demonstrates that RR with a larger quantum size performs better because fewer context switches are required for the long burst processes that arrive early. Scenario 4's performance is hindered by a quantum of 1 but improved with higher quanta due to the shorter nature of processes. In scenario 5, with mostly short processes, the differences in quantum size have a less significant impact on the average response times, with lower overhead and quick process turnover contributing to lower average turnaround times across all quanta.

Across all five scenarios, SJF consistently offers the best performance by prioritizing the execution of the shortest processes available. This approach significantly reduces the wait times for other processes, thereby minimizing the average turnaround time. SJF's ability to quickly process shorter jobs and effectively manage the queue results in the most efficient turnaround times compared to the other algorithms. This consistent efficiency underscores SJF's utility in situations where process burst times are known and minimizing turnaround time is the primary objective.

In conclusion, while SJF typically leads in efficiency due to its preferential treatment of shorter tasks, its real-world application is limited by the need for accurate burst time information. FCFS is most effective when processes have similar burst times, or at least when long processes do not precede shorter ones. RR's performance is highly contingent on the quantum size; a well-chosen quantum can balance the need for process responsiveness with the overhead of context switching. The ideal scheduling strategy thus depends on the specific requirements of the operating environment, including the nature of the tasks and the desired balance between fairness, efficiency, and response time.

Research Novelty and Contribution

To reinforce the conclusions of our study on CPU scheduling methods, we need to contextualize them within the larger research environment by comparing them to past studies. Such comparisons not only highlight the novelty and significance of our study, but also provide a clearer picture of the impacts and advances we have made.

In this study, we investigate three primary areas of innovation: the implementation of dynamic quantum size adjustments in RR scheduling, the adoption of a scenario-based evaluation methodology, and a comprehensive comparative performance analysis of various CPU scheduling algorithms. Each of these areas introduces new methodologies and insights that make a substantial contribution to the field, thereby improving the theoretical and practical aspects of current scheduling practices.

Dynamic Quantum Size Allocation for Round-Robin Scheduling

Our research looks on the effects of dynamic quantum size on the Round-Robin (RR) scheduling algorithm. Previous research, such as those by Behera *et al*. (2010); Datta (2015), investigated fluctuations in quantum size but did not delve deeply into dynamic adjustment based on real-time system demands. Our work makes a unique contribution to this field by empirically demonstrating how altering quantum sizes can improve both response time and context switching, especially in high-variability workload conditions.

Scenario-Based Evaluation Methodology

Our evaluation methodology takes a scenario-based approach that closely resembles real-world operational situations, unlike traditional studies that generally model static or limited environments (Stallings, 2012; Abdulrahim *et al*., 2014). This method enables more rigorous testing of algorithms under a variety of load scenarios, providing insights that are directly applicable to practical settings. This is a considerable improvement over traditional evaluations, offering a more complete picture of algorithm performance dynamics.

Comparative Performance Analysis

Xiong and Chung (2012); Mogul and Borg (1991) found that CPU scheduling techniques are often compared based on a limited set of criteria and fail to consider complicated system demands. Our study expands on existing assessments by include a wide range of performance indicators such as response time, turnaround time, and context switching. For example, our findings on the SJF algorithm's improved performance in minimizing reaction times are consistent with those of Jeyaprakash and Sambath (2021), but we go into greater detail about how burst time variability influences this efficiency.

Highlighting Key Findings in Relationship to Existing Literature

Our findings suggest that FCFS can perform effectively under conditions of uniform process arrival times, which is consistent with Van Houdt (2022) findings. However, our scenario-based findings add to the knowledge foundation provided by Zhao and Stankovic (2003) by providing additional insights into how initial lengthy burst processes have a negative impact on FCFS performance. In addition, the analysis confirms the efficiency of Shortest Job First (SJF) in predictable process environments, aligning with previous research (Yosuf *et al*., 2021). However, we uniquely quantify the impact of prediction accuracy on SJF performance, providing a critical perspective that supports Helmy *et al*. (2015) theoretical model. Moreover, our research supports Sohrawordi (2019) findings that the Round Robin (RR) approach allocates CPU time fairly. However, we contribute to this understanding by illustrating how ideal quantum sizes can considerably reduce the overhead associated with frequent context switching, as proposed by Mohanty *et al*. (2011).

Conclusion

The results obtained from our exhaustive scenario-based assessment illustrate the distinct merits and drawbacks of the CPU scheduling algorithms FCFS, RR, and SJF. In diverse scenarios, the SJF algorithm consistently achieved lower average response and turnover times than FCFS and RR. This demonstrates the algorithm’s effectiveness in processing environments where task durations are predetermined and it is possible to prioritize shorter tasks. Although the FCFS algorithm is uncomplicated, its performance can vary depending on the order in which processes arrive. In situations where lengthier tasks come before shorter ones, it struggles significantly. The selection of quantum size had a significant impact on the RR scheduling method’s performance; under specific conditions, optimal quantum settings reduced context switching overhead and closely resembled the efficiency of FCFS. By illustrating the performance of widely used algorithms under simulated real-world conditions, this study contributes to the ongoing discourse on CPU scheduling with empirical data.

Our results indicate that, although no algorithm performs better than others in every circumstance, the specific demands of the computing environment the characteristics of the tasks, and the intended equilibrium between efficiency, fairness, and response time should guide the selection of a scheduling strategy. Subsequent investigations may delve into the amalgamation of machine learning methodologies to forecast the dynamic quantum size, as well as the formulation of hybrid scheduling approaches that leverage the respective merits of FCFS, RR and SJF to augment the efficiency of CPU scheduling.

Acknowledgment

The authors would like to express sincere gratitude to AlMaarefa University, Riyadh, Saudi Arabia, for supporting this research.

Funding Information

The authors have not received any financial support or funding to report.

Author's Contributions

Olaa Hajjar: Conducted the primary research, designed and executed the experiments, and contributed extensively to the manuscript written. Implementing the scenario-based approach, analyzed the results and revising the manuscript for clarity. Conducted extensive research to develop the experimental framework.

Escelle Mekhallalati: Designed and executed the experiments, ensuring the accuracy and reliability of results. contributed extensively to the manuscript written. Implemented the scenario-based approach, analyzed the results, and revised the manuscript for clarity.

Nada Annwty: Analyzed data meticulously, interpreted findings to draw meaningful conclusions. Contributed extensively to the manuscript written.

Faisal Alghayadh: Played a central role in drafted the manuscript, contributed to the introduction, methodology, results and discussion sections and contributed extensively to the manuscript written.

Ismail Keshta: Assisted in data collection and processed, provided technical support throughout the research contributed to the discussion section contributed extensively to the manuscript written.

Mohammed Algabri: Oversaw the project, provided guidance on methodology selection and critically reviewed and revised the manuscript for intellectual content and coherence, contributed extensively to the manuscript written. Played a central role in drafting the manuscript, and contributed to the introduction, methodology, results and discussion sections.

Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

References

Abdul Kareem, E. I., & Hussein, S. A. (2022). Optimal CPU Jobs Scheduling Method Based on Simulated Annealing Algorithm. *Iraqi Journal of Science*, *63*(8), 3640–3651. https://doi.org/10.24996/ijs.2022.63.8.38

Abdulrahim, A., E. Abdullahi, S., & B. Sahalu, J. (2014). A New Improved Round Robin (NIRR) CPU Scheduling Algorithm. *International Journal of Computer Applications*, *90*(4). https://doi.org/10.5120/15563-4277

Abirami, B., & Vasudevan, V. (2023). Modified multilevel feedback queue scheduling algorithm with starvation mitigation for reconfigurable computing systems. *Research Square*. https://doi.org/10.21203/rs.3.rs-2820144/v1

Adamu, I. M., Gital, A. Y., Boukari, S., & Zahraddeen Yakubu, I. (2019). Performance Evaluation of Hybrid Round Robin Algorithm and Modified Round Robin Algorithm in Cloud Computing. *International Journal of Recent Technology and Engineering (IJRTE)*, *8*(2), 5047–5051. https://doi.org/10.35940/ijrte.a9139.078219

Adeleke, I. A. (2022). Comparative Analysis and Performance Evaluation of Contiguous Memory Techniques. *UNIOSUN Journal of Engineering and Environmental Sciences*, *4*(2). https://doi.org/10.36108/ujees/2202.40.0270

Altman, N. S. (1992). An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression. *The American Statistician*, *46*(3), 175–185. https://doi.org/10.2307/2685209

Asef, A.-K., Abdullah, R., & Abdul Rash, N. (2009). Job Type Approach for Deciding Job Scheduling in Grid Computing Systems. *Journal of Computer Science*, *5*(10), 745–750. https://doi.org/10.3844/jcssp.2009.745.750

Avrahami, N., & Azar, Y. (2007). Minimizing Total Flow Time and Total Completion Time with Immediate Dispatching. *Algorithmica*, *47*(3), 253–268. https://doi.org/10.1007/s00453-006-0193-6

Barroso, L. A., Hölzle, U., & Ranganathan, P. (2019). The data center as a Computer. *Synthesis Lectures on Computer Architecture*, XVIII, 189. https://doi.org/10.2200/s00516ed2v01y201306cac024

Behera, H. S., Mohanty, R., & Nayak, D. (2010). A New Proposed Dynamic Quantum with Re-Adjusted Round Robin Scheduling Algorithm and Its Performance Analysis. *International Journal of Computer Applications*, *5*(5). https://doi.org/10.5120/913-1291

Chandiramani, K., Verma, R., & Sivagami, M. (2019). A Modified Priority Preemptive Algorithm for CPU Scheduling. *Procedia Computer Science*, *165*, 363-369. https://doi.org/10.1016/j.procs.2020.01.037

Chhugani, B., & Silvester, M. (2017). Improving Round Robin Process Scheduling Algorithm. *International Journal of Computer Applications*, *166*(6). https://doi.org/10.5120/ijca2017914034

Choi, S., Robinson, J. E., Mulfinger, D. G., & Capozzi, B. J. (2010). Design of an optimal route structure using heuristics-based stochastic schedulers. *29th Digital Avionics Systems Conference*, 2.A.5-1-2.A.5-17. https://doi.org/10.1109/dasc.2010.5655500

Dash, A. R., Kumar Sahu, S., & Kewal, B. (2019). An Optimized Disk Scheduling Algorithm with Bad-Sector Management. *International Journal of Computer Science, Engineering and Applications*, *9*(3). https://doi.org/10.5121/ijcsea.2019.9301

Dash, A. R., Sahu, S. kumar, & Samantra, S. K. (2015). An Optimized Round Robin CPU Scheduling Algorithm with Dynamic Time Quantum. *International Journal of Computer Science, Engineering and Information Technology*, *5*(1). https://doi.org/10.5121/ijcseit.2015.5102

Datta, L. (2015). Efficient Round Robin Scheduling Algorithm with Dynamic Time Slice. *International Journal of Education and Management Engineering*, *5*(2), 10–19. https://doi.org/10.5815/ijeme.2015.02.02

Elmougy, S., Sarhan, S., & Joundy, M. (2017). A novel hybrid of Shortest job first and round Robin with dynamic variable quantum time task scheduling technique. *Journal of Cloud Computing*, *6*, 12. https://doi.org/10.1186/s13677-017-0085-0

Faizan, K., Marikal, A., & Anil, K. (2020). A Hybrid Round Robin Scheduling Mechanism for Process Management. *International Journal of Computer Applications*, *177*(36). https://doi.org/10.5120/ijca2020919851

Fan, Y., Lan, Z., Rich, P., Allcock, W. E., Papka, M. E., Austin, B., & Paul, D. (2019). Scheduling Beyond CPUs for HPC. *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, 97–108. https://doi.org/10.1145/3307681.3325401

Gaikwad, G. D. (2021). Refresh Rate Identification Strategy for Optimal Page Replacement Algorithms for Virtual Memory Management. *International Journal for Research in Applied Science and Engineering Technology*, *9*(11). https://doi.org/10.22214/ijraset.2021.38770

Hasan, T. F. (2014). CPU Scheduling Visualization. *Diyala Journal of Engineering Sciences*, *7*(1), 16–29. https://doi.org/10.24237/djes.2014.07102

Hashim Yosuf, R., A. Mokhtar, R., A. Saeed, R., Alhumyani, H., & Abdel-Khalek, S. (2022). Scheduling Algorithm for Grid Computing Using Shortest Job First with Time Quantum. *Intelligent Automation & Soft Computing*, *31*(1), 581–590. https://doi.org/10.32604/iasc.2022.019928

Helmy, T., Al-Azani, S., & Bin-Obaidellah, O. (2015). A Machine Learning-Based Approach to Estimate the CPU-Burst Time for Processes in the Computational Grids. *2015 3rd International Conference on Artificial Intelligence, Modelling and Simulation (AIMS)*, 3–8. https://doi.org/10.1109/aims.2015.11

Hu, Z., & Li, D. (2022). Improved heuristic job scheduling method to enhance throughput for big data analytics. *Tsinghua Science and Technology*, *27*(2), 344–357. https://doi.org/10.26599/tst.2020.9010047

Jeyaprakash, T., & Sambath, M. (2021). Performance analysis of CPU scheduling algorithms-A problem solving approach. *International Journal of Science and Management Studies (IJSMS)*, *4*(4). https://doi.org/10.51386/25815946/ijsms-v4i4p138

Karapici, A., Feka, E., Tafa, I., & Allkoci, A. (2015). The Simulation of Round Robin and Priority Scheduling Algorithm. *2015 12th International Conference on Information Technology - New Generations*, 758-758. https://doi.org/10.1109/itng.2015.131

Khatri, J. (2016). An Enhanced Round Robin CPU Scheduling Algorithm. *IOSR Journal of Computer Engineering*, *18*(4), 20–24. https://doi.org/10.9790/0661-1804022024

Kumar Mishra, M., & Rashid, F. (2014). An Improved Round Robin CPU Scheduling Algorithm with Varying Time Quantum. *International Journal of Computer Science, Engineering and Applications*, *4*(4). https://doi.org/10.5121/ijcsea.2014.4401

Lee, J., & Chung, S. G. (2019). Analysts' reactions to firms' real activities management. *Review of Accounting and Finance*, *18*(4), 589–612. https://doi.org/10.1108/raf-05-2017-0105

Liang, C.-C. (2019). Enjoyable queuing and waiting time. *Time & Society*, *28*(2), 543–566. https://doi.org/10.1177/0961463x17702164

Liu, F., Guo, F., Solihin, Y., Kim, S., & Eker, A. (2008). Characterizing and modeling the behavior of context switch misses. *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, 91–101. https://doi.org/10.1145/1454115.1454130

Lundberg, S. M., Erion, G., Chen, H., DeGrave, A., Prutkin, J. M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., & Lee, S.-I. (2020). From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*, *2*, 56–67. https://doi.org/10.1038/s42256-019-0138-9

Lenka, K. R., & Ranjan, P. (2012). A 2LFQ Scheduling with Dynamic Time Quantum using Mean Average. *International Journal of Computer Applications*, *47*(23). https://doi.org/10.5120/7495-0560

Mostafa, M. S., & Kusakabe, S. (2015). Effect of Thread Weight Readjustment Scheduler on Scheduling Criteria. *Information Engineering Express*, *1*(2), 1-10. https://doi.org/10.52731/iee.v1.i2.9

Matarneh, R. J. (2009). Self-Adjustment Time Quantum in Round Robin Algorithm Depending on Burst Time of the Now Running Processes. *American Journal of Applied Sciences*, *6*, 1831–1837. https://doi.org/10.3844/ajassp.2009.1831.1837

Mi, N., Casale, G., & Smirni, E. (2012). ASIdE: Using Autocorrelation-Based Size Estimation for Scheduling Bursty Workloads. *IEEE Transactions on Network and Service Management*, *9*(2), 198–212. https://doi.org/10.1109/tnsm.2012.041712.100073

Mogul, J. C., & Borg, A. (1991). The effect of context switches on cache performance. *ACM SIGPLAN Notices*, *26*(4), 75–84. https://doi.org/10.1145/106973.106982

Mohanty, R., Behera, H. S., Patwari, K., Dash, M., & Prasanna, L. (2011). Priority Based Dynamic Round Robin (PBDRR) Algorithm with Intelligent Time Slice for Soft Real Time Systems. *International Journal of Advanced Computer Science and Applications*, *2*(2). https://doi.org/10.14569/ijacsa.2011.020209

Moharana, S. C., Samal, S., Swain, A. R., & Mund, G. B. (2018). Dynamic CPU scheduling for load balancing in virtualized environments. *Turkish Journal of Electrical Engineering & Computer Sciences*, *26*(5), 2512–2524. https://doi.org/10.3906/elk-1709-112

Omar, H. K., Jihad, K. H., & Hussein, S. F. (2021). Comparative analysis of the essential CPU scheduling algorithms. *Bulletin of Electrical Engineering and Informatics*, *10*(5), 2742–2750. https://doi.org/10.11591/eei.v10i5.2812

Omotehinwa, T., Azeeze, I., & Oyekanmi, E. (2019). An improved round robin cpuscheduling algorithm for asymmetrically distributed burst times. *Africa Journal Management Information System*, *1*(4), 50–68.

Park, J.-W., Kwon, M.-W., & Hong, T. (2022). Queue congestion prediction for large-scale high performance computing systems using a hidden Markov model. *The Journal of Supercomputing*, *78*, 12202–12223. https://doi.org/10.1007/s11227-022-04356-z

Pon Pushpa, S. E., & Devasigamani, M. (2014). Utilization Bound Scheduling Analysis for Nonpreemptive Uniprocessor Architecture Using UML-RT. *Modelling and Simulation in Engineering*, *2014*, 705929. https://doi.org/10.1155/2014/705929

Putra, T. D., & Purnomo, R. (2022). Simulation of Priority Round Robin Scheduling Algorithm. *Sinkron*, *6*(4), 2170–2181. https://doi.org/10.33395/sinkron.v7i4.11665

Raheja, S., Dadhich, R., & Rajpal, S. (2014). 2-Layered Architecture of Vague Logic Based Multilevel Queue Scheduler. *Applied Computational Intelligence and Soft Computing*, *2014*, 341957. https://doi.org/10.1155/2014/341957

Sahoo, K. S., Tiwary, M., Sahoo, B., Mishra, B. K., RamaSubbaReddy, S., & Luhach, A. Kr. (2020). RTSM: Response time optimisation during switch migration in software-defined wide area network. *IET Wireless Sensor Systems*, *10*(3), 105–111. https://doi.org/10.1049/iet-wss.2019.0125

Sambath, M., K. Padmaveni, Joseph, L., Ravi S., J. Thangakumar, & Aravindhar, J. (2020). Convoy effect elimination in fcfs scheduling. *International Journal of Engineering and Advanced Technology*, *9*(3). https://doi.org/10.35940/ijeat.c6092.029320

Senan, S. (2017). A Neural Net-Based Approach for CPU Utilization. *Bilişim Teknolojileri Dergisi*, *10*(3), 263–272. https://doi.org/10.17671/gazibtd.331037

Shafi, U., Shah, M., Wahid, A., Abbasi, K., Javaid, Q., Asghar, M., & Haider, M. (2020). A Novel Amended Dynamic Round Robin Scheduling Algorithm for Timeshared Systems. *The International Arab Journal of Information Technology*, *17*(1). https://doi.org/10.34028/iajit/17/1/11

Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts.*

Sohrawordi, Ali, E., Uddin, P., & Hossain, M. (2019). A Modified Round Robin CPU Scheduling Algorithm with Dynamic Time Quantum. *International Journal of Advanced Research*, *7*(2), 422–429. https://doi.org/10.21474/ijar01/8506

Somasundaram, K., & Radhakrishnan, S. (2009). Task Resource Allocation in Grid using Swift Scheduler. *International Journal of Computers Communications & Control*, *4*(2), 158. https://doi.org/10.15837/ijccc.2009.2.2423

Stallings, W. (2012). *Operating Systems: Internals and Design Principles*.

Süzen, A. A., & Taşdelen, K. (2018). Recovering Multimedia Files from a Memory Image. *Journal of Polytechnic*, *21*(3), 731–737. https://doi.org/10.2339/politeknik.417767

Tajwar, M. M., Pathan, Md. N., Hussaini, L., & Abubakar, A. (2017). CPU Scheduling with a Round Robin Algorithm Based on an Effective Time Slice. *Journal of Information Processing Systems*, *13*(4), 941–950. https://doi.org/10.3745/jips.01.0018

Thombare, M., Sukhwani, R., Shah, P., Chaudhari, S., & Raundale, P. (2016). Efficient implementation of Multilevel Feedback Queue Scheduling. *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, 1950–1954. https://doi.org/10.1109/wispnet.2016.7566483

Towsley, D., Chandy, K. M., & Browne, J. C. (1978). Models for parallel processing within programs: application to CPU: I/O and I/O: I/O overlap. *Communications of the ACM*, *21*(10), 821–831. https://doi.org/10.1145/359619.359622

Engström, J., Liu, S.-Y., Dinparastdjadid, A., & Simoiu, C. (2024). Modeling road user response timing in naturalistic traffic conflicts: A surprise-based framework. *Accident Analysis & Prevention*, *198*, 107460. https://doi.org/10.1016/j.aap.2024.107460

Van Houdt, B. (2022). On the Stochastic and Asymptotic Improvement of First-Come First-Served and Nudge Scheduling. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, *6*(3), 1–22. https://doi.org/10.1145/3570610

Xiong, B., & Chung, C. (2012). Completion Time Scheduling and the WSRPT Algorithm. In A. R. Mahjoub, V. Th. Paschos, V. Markakis, & I. Milis (Eds.), *Springer, Berlin, Heidelberg* (Lecture Notes in Computer Science, Vol. *7422*, pp. 426–426). https://doi.org/10.1007/978-3-642-32147-4_37

Xu, Y., Ge, H., Wu, S., & Yang, J. (2024). TELKOMNIKA (Telecommunication Computing Electronics and Control). *UAD Universitas Ahmad Dahlan*, *22*(3). https://doi.org/10.12928/telkomnika.v14i3a.4433

Yamada, S., & Kusakabe, S. (2008). Effect of context aware scheduler on TLB. *2008 IEEE International Symposium on Parallel and Distributed Processing*, 1–8. https://doi.org/10.1109/ipdps.2008.4536361

Younis, M. F. (2021). ESJF Algorithm to Improve Cloud Environment. *Iraqi Journal of Science*, *62*(11), 4171–4180. https://doi.org/10.24996/ijs.2021.62.11.35

Zafar Iqbal, S., Gull, H., Saeed, S., Saqib, M., Alqahtani, M., A. Bamarouf, Y., Krishna, G., & Issa Aldossary, M. (2022). Relative Time Quantum-based Enhancements in Round Robin Scheduling. *Computer Systems Science and Engineering*, *41*(2), 461–477. https://doi.org/10.32604/csse.2022.017003

Zhao, W., & Stankovic, J. A. (2003). Performance analysis of FCFS and improved FCFS scheduling algorithms for dynamic real-time computer systems. *[1989] Proceedings. Real-Time Systems Symposium*, 156–165. https://doi.org/10.1109/real.1989.63566

Zouaoui, S., Boussaid, L., & Mtibaa, A. (2019). Priority based round robin (PBRR) CPU scheduling algorithm. *International Journal of Electrical and Computer Engineering (IJECE)*, *9*(1), 190–202. https://doi.org/10.11591/ijece.v9i1.pp190-202