https://jsras.rcc.edu.ly/
Vol.3 No.2 (2026), 98-116
Article history:
Received: 20/6/2026
Accepted: 15/7/2026
Published: 20/8/2026

Journal of Sustainable Research in Applied Sciences

A Comprehensive Multi-Criteria Evaluation Framework for CPU Scheduling Algorithms

Khalifa Benghuzi[1,*], k.benghuzi.pg@it.misuratau.edu.ly, https://orcid.org/0009-0006-6315-9010
Anwar Alhenshiri[2], alhenshiri@it.misuratau.edu.ly, https://orcid.org/0009-0007-8825-7685
[1,2] Department of Computer Science, Faculty of Information Technology, Misurata University, Misurata, Libya
DOI https://doi.org/10.36602/jsras.2026.3.2.21
* Corresponding author: k.benghuzi.pg@it.misuratau.edu.ly

Index Terms

CPU Scheduling Algorithms, Multi-Criteria Decision Making (MCDM), Performance Evaluation Framework, Simple Additive Weighting (SAW).

Abstract

Evaluating CPU scheduling algorithms is inherently challenging due to the need to balance conflicting criteria such as efficiency, fairness, responsiveness, and starvation avoidance. However, most existing studies rely on a limited set of metrics, often leading to incomplete and potentially misleading conclusions. This paper proposes a comprehensive multi-criteria evaluation framework that enables a more balanced and reliable assessment of scheduling performance. The framework integrates key performance metrics, including average waiting time, turnaround time, response time, fairness, priority adherence, and starvation. A Simple Additive Weighting (SAW) method is employed to aggregate these metrics into a unified evaluation model. To enhance reliability, the non-parametric Friedman test is applied as a statistical validation procedure. Experimental results reveal significant trade-offs among algorithms, underscoring the limitations of single-metric evaluation and demonstrating the effectiveness of the proposed evaluation framework for systematic and informed comparison.

إطار شامل للتقييم متعدد المعايير لخوارزميات جدولة وحدة المعالجة المركزية

خليفة بن غزي [1،*]، k.benghuzi.pg@it.misuratau.edu.ly ، أنور الهنشيري [2]، alhenshiri@it.misuratau.edu.ly
[1،2] قسم علوم الحاسوب، كلية تقنية المعلومات، جامعة مصراتة، ليبيا
DOI https://doi.org/10.36602/jsras.2026.3.2.21
* المؤلف المراسل: k.benghuzi.pg@it.misuratau.edu.ly

الكلمات المفتاحية

خوارزميات جدولة وحدة المعالجة المركزية، اتخاذ القرار متعدد المعايير، إطار تقييم الأداء، أسلوب الترجيح الإضافي البسيط(SAW) .

الملخص

يُعدّ تقييم خوارزميات جدولة وحدة المعالجة المركزية **(CPU)** تحديًا بطبيعته، نظرًا إلى الحاجة إلى تحقيق توازن بين معايير متعارضة، مثل الكفاءة والعدالة وسرعة الاستجابة وتجنب الاستعصاء الناتج عن الانتظار **(Starvation).** ومع ذلك، تعتمد معظم الدراسات الحالية على مجموعة محدودة من المقاييس، مما يؤدي في كثير من الأحيان إلى استنتاجات غير مكتملة وقد تكون مضللة. تقترح هذه الورقة إطارًا شاملًا للتقييم متعدد المعايير، يتيح إجراء تقييم أكثر توازنًا وموثوقية لأداء خوارزميات الجدولة. ويجمع الإطار مجموعة من مقاييس الأداء الأساسية، تشمل متوسط زمن الانتظار، وزمن الدوران، وزمن الاستجابة، والعدالة، ومدى الالتزام بالأولوية، ومقاومة الاستعصاء الناتج عن الانتظار. كما تُستخدم طريقة الجمع الإضافي البسيط **(SAW)** لدمج هذه المقاييس في نموذج تقييم موحّد. ولتعزيز موثوقية النتائج، يُطبَّق اختبار فريدمان اللامعلمي **(Friedman Test)** بوصفه إجراءً للتحقق الإحصائي. وتكشف النتائج التجريبية عن وجود مفاضلات جوهرية بين خوارزميات الجدولة، مما يؤكد محدودية الاعتماد على مقياس أداء واحد، ويبرهن في الوقت ذاته على فعالية إطار التقييم المقترح في إجراء مقارنات منهجية وموثوقة وأكثر استنارة بين خوارزميات جدولة وحدة المعالجة المركزية.

1. Introduction

CPU scheduling is a fundamental function of operating systems, responsible for allocating processor time among competing processes in order to optimize system performance. Effective scheduling directly influences key performance metrics such as waiting time, turnaround time, response time, throughput, and fairness. With the increasing complexity of modern computing environments, including cloud systems and heterogeneous workloads, scheduling algorithms are required to balance multiple, often conflicting objectives rather than a single performance criterion (Sharma et al., 2022; Iqbal et al., 2023).

A wide range of CPU scheduling algorithms has been proposed, including classical approaches such as First-Come-First-Served (FCFS), Shortest Job First (SJF),

and Round Robin (RR), as well as more advanced hybrid and adaptive methods. Recent studies have explored dynamic time quantum techniques, hybrid scheduling strategies, and intelligent approaches to improve performance under diverse workloads (Shafi et al., 2020; Zohora et al., 2024). While these approaches demonstrate improvements in specific metrics, their performance often varies depending on workload characteristics, leading to trade-offs between efficiency, fairness, and responsiveness.

Despite these advancements, a major limitation exists in the evaluation of CPU scheduling algorithms. Most existing studies rely on a limited number of performance metrics typically average waiting time and turnaround time while neglecting other critical aspects such as fairness, priority adherence, and starvation (Kolios et al., 2016). This narrow evaluation perspective can result in incomplete or misleading conclusions regarding the overall effectiveness of scheduling algorithms. In particular, fairness has been identified as an important yet often overlooked metric, especially in modern systems where equitable resource allocation is essential (Kolios et al., 2016).

Another challenge is the lack of systematic and statistically grounded evaluation methodologies. Many studies compare algorithms using experimental results without applying rigorous statistical validation, making it difficult to assess the significance and generalizability of their findings. As a result, conclusions drawn from such comparisons may not be reliable across different scenarios or workloads.

To address these limitations, this paper proposes a comprehensive multi-criteria evaluation framework for CPU scheduling algorithms. The framework integrates multiple performance metrics into a unified evaluation model, enabling a holistic assessment of scheduling performance across efficiency, fairness, responsiveness, and robustness dimensions. A multi-criteria decision-making approach based on the Simple Additive Weighting (SAW) method is used to aggregate evaluation criteria, while statistical validation techniques are incorporated to ensure the reliability of the results.

The main contribution of this work is the development of a structured and extensible evaluation framework that shifts the focus from single-metric comparisons to multi-dimensional performance analysis. By providing a systematic methodology for evaluating scheduling algorithms, this paper supports more accurate comparison, analysis, and selection of scheduling strategies in modern computing environments.

The remainder of this paper is organized as follows. Section 2 reviews related work on CPU scheduling evaluation and performance metrics. Section 3 presents the proposed evaluation framework. Section 4 describes the experimental setup. Section 5 presents and analyzes the results, , Section 6 presents the limitations, and Section 7 concludes the paper.

2. RELATED WORK

CPU scheduling has been extensively studied as a fundamental operating-system function because scheduling decisions directly affect waiting time, turnaround time, responsiveness, fairness, and CPU utilization. Classical scheduling algorithms provide the foundation for most subsequent developments. First-Come-First-Served (FCFS) schedules processes according to their arrival order and provides a simple non-preemptive scheduling policy, whereas Shortest Job First (SJF) selects the available process with the shortest burst time and can achieve low average waiting time under suitable workload conditions. Priority Scheduling selects processes according to their assigned priorities, providing preferential service to important processes but potentially increasing the waiting time of lower-priority processes. Round Robin (RR) addresses fairness by allocating CPU time in fixed time slices and preempting processes when their time quantum expires. Multilevel Feedback Queue (MLFQ) extends this concept by using multiple priority queues and dynamically moving processes between queues according to their execution behavior. These classical approaches provide different trade-offs among efficiency, responsiveness, fairness, and scheduling overhead and therefore serve as important baselines for comparative CPU scheduling studies (Stallings, 2011; Silberschatz et al., 2013; Tanenbaum & Bos, 2015).

A major line of research has focused on improving RR through adaptive or dynamic time quantum selection. The Self-Adjusting RR algorithm proposed by Matarneh (2009) dynamically determines the time quantum according to the burst-time characteristics of the processes. Mohanty et al. (2010) proposed Shortest Remaining Burst Round Robin (SRBRR), combining burst-based process selection with RR execution. Yadav et al. (2010) introduced an Improved RR algorithm, while Singh et al. (2010) proposed an Optimized RR approach to improve the efficiency of CPU scheduling. Hiranwal and Saroj (2011) further developed an Adaptive RR based on the shortest burst approach, using process burst characteristics to determine a more suitable time slice. Mishra (2012) proposed another Improved RR model with the objective of reducing waiting time through improved quantum selection. These approaches established the importance of adapting the time quantum to workload characteristics rather than relying exclusively on a fixed value.

Subsequent studies investigated statistical and workload-based mechanisms for calculating the dynamic time quantum. Agha and Jassbi (2013) proposed RR-HARM, in which the harmonic and arithmetic means of burst-time characteristics are used to determine the quantum. Panda and Bhoi (2014) introduced RR-MMDM, using a minimum–maximum dispersion measure to calculate an adaptive time quantum. Mishra and Rashid (2014) proposed IRR-VTQ, in which the time quantum varies according to the characteristics of the current workload. Dash and Samantra (2016) developed an Optimized RR with Dynamic Time Quantum, further emphasizing continuous adjustment of the time slice according to process characteristics. These approaches demonstrate that the choice of time-quantum calculation method can substantially affect waiting time, turnaround time, response behavior, and context-switching overhead.

Another group of studies combined adaptive quantum mechanisms with hybrid scheduling strategies. Hamayun and Khurshid (2015) proposed an Optimized SJF algorithm to improve the performance of shortest-job-

based scheduling. Gupta et al. (2016) introduced Smart Job First Dynamic Round Robin (SJFDRR), combining SJF-based process selection with dynamic RR execution. Elmougy et al. (2017) proposed a Hybrid SJF + RR with Dynamic Variable Quantum, integrating shortest-job selection with a dynamically varying time slice. These approaches attempt to overcome the limitations of individual scheduling policies by combining burst-time-based selection with time-sliced execution. However, improving one or two conventional metrics does not necessarily guarantee balanced performance across efficiency, fairness, responsiveness, and priority-related measures.

Priority-aware extensions of RR have also been investigated. Rajput and Gupta (2012) proposed Priority-RR (PBRR), which combines priority-based process selection with RR time slicing. Arya et al. (2018) developed a Priority-based Improved RR algorithm that incorporates process priority into RR scheduling. Zouaoui et al. (2019) proposed another Priority-Based Round Robin (PBRR) approach to improve scheduling performance while preserving priority considerations. These methods demonstrate the importance of incorporating process priority into time-sliced scheduling, particularly when processes have different levels of importance. Nevertheless, priority awareness may introduce trade-offs between priority compliance and fairness, especially when lower-priority processes experience prolonged waiting.

More recent work has continued to refine adaptive RR through workload-aware and intelligent quantum selection. Mody and Mirkar (2019) proposed Smart Round Robin (SRR), which dynamically adjusts scheduling behavior according to workload characteristics. Paul et al. (2019) introduced Progressive Time Quantum RR, in which the time quantum is progressively adjusted across scheduling cycles. Shafi et al. (2020) proposed an Amended Dynamic RR algorithm for time-shared systems, incorporating dynamic quantum selection to improve scheduling performance. Sharma et al. (2021) developed an Intelligent RR algorithm that applies heuristic decision-making to improve the allocation of CPU time. Alhaidari and Balharith (2021) extended adaptive RR to cloud computing environments through an Enhanced RR algorithm designed to improve task scheduling and resource utilization. Sharma et al. (2022) proposed Median-Average RR, using statistical characteristics of burst times to determine the scheduling quantum. Iqbal et al. (2023) further incorporated process priorities with dynamic time quantum selection, demonstrating the potential benefits of jointly considering temporal and priority characteristics. Zohora et al. (2024) proposed an Enhanced Dynamic RR approach for real-time cloud environments with asymmetric burst-length processes, further highlighting the importance of adaptive quantum mechanisms under heterogeneous workloads.

A closely related hybrid approach is HACS-DTQSAW (Hybrid Adaptive CPU Scheduling Algorithm Using a Dynamic Time Quantum and Simple Additive Weighting) introduced by Benghuzi and Alhenshiri (2026). The algorithm combines Arrival Time, Burst Time, and Priority within a SAW-based scheduling decision and uses a dynamic time quantum to adapt CPU allocation to workload characteristics. This approach extends adaptive RR scheduling by considering multiple process attributes simultaneously rather than relying primarily on burst-time information.

The reviewed algorithms demonstrate a clear progression from fixed-quantum RR toward adaptive, hybrid, priority-aware, and workload-sensitive scheduling strategies. However, the algorithms differ substantially in the criteria used to make scheduling decisions. Classical algorithms such as FCFS, SJF, Priority, and RR primarily emphasize a single scheduling principle, whereas adaptive algorithms generally focus on burst-time characteristics and dynamic quantum adjustment. Hybrid approaches combine multiple scheduling mechanisms, while priority-based variants introduce an additional process attribute into scheduling decisions. HACS-DTQSAW further combines multiple process attributes with adaptive time-quantum allocation. Consequently, an algorithm that performs well in average waiting time, for example, may not necessarily provide the best response time, fairness, priority compliance, or resistance to excessive waiting.

This observation is particularly important when comparing a large and heterogeneous set of scheduling algorithms. The algorithms considered in this study include four classical approaches (FCFS, SJF, Priority, and RR) and twenty-five adaptive, dynamic, priority-aware, and hybrid scheduling approaches, representing different generations of CPU scheduling research. The latter group includes MLFQ and the dynamic and hybrid RR-related algorithms reviewed above. In addition, multiple configurations of HACS-DTQSAW are included in the experimental benchmark. This diversity provides a suitable basis for evaluating whether an algorithm's apparent advantage under an individual metric remains valid when multiple dimensions of scheduling performance are considered simultaneously.

Despite the extensive development of these scheduling algorithms, most reported evaluations emphasize a relatively small number of conventional measures, particularly Average Waiting Time (AWT), Average Turnaround Time (ATT), and, in some cases, response time or context switches. Such evaluations may favor algorithms optimized for a particular objective while overlooking other important properties of scheduling behavior, including fairness, priority adherence, and resistance to excessive waiting. Direct comparison is therefore difficult when different studies use different workloads, metrics, and evaluation procedures.

To address this evaluation gap, the present study does not focus on evaluating a scheduling algorithm using a single performance objective. Instead, it establishes a unified multi-criteria evaluation framework for comparing the heterogeneous scheduling algorithms included in the experimental benchmark under identical workload conditions and measurement procedures. Nine complementary criteria are considered: Average Waiting Time, Average Turnaround Time, Average Response Time, Context Switches, Jain's Fairness Index for Waiting Time, Jain's Fairness Index for Turnaround Time, Jain's Fairness Index for Response Time, Priority Respect Ratio, and Starvation Index. The criteria are converted into normalized rank-based preference values and aggregated using the Simple Additive Weighting (SAW) method, followed by statistical validation, followed by statistical validation. This approach enables the study to evaluate

whether the performance advantages reported for individual scheduling algorithms remain consistent when efficiency, responsiveness, fairness, priority handling, and resistance to excessive waiting are assessed jointly.

Overall, the existing literature provides a strong foundation of classical, dynamic, adaptive, priority-aware, and hybrid scheduling algorithms, but their comparative evaluation remains largely metric-specific. The present work complements these algorithmic developments by providing a common and reproducible evaluation methodology through which all selected algorithms, including the different HACS-DTQSAW configurations, can be assessed using the same workloads, performance criteria, aggregation procedure, and statistical validation framework.

3. PROPOSED MULTI-CRITERIA EVALUATION FRAMEWORK

This section presents the proposed multi-criteria evaluation framework for systematically comparing heterogeneous CPU scheduling algorithms under multiple and potentially conflicting performance objectives. Rather than relying on a single measure such as Average Waiting Time or Average Turnaround Time, the framework integrates efficiency, responsiveness, scheduling overhead, fairness, priority compliance, and starvation resistance into a unified evaluation procedure.

The framework uses a two-level rank-based Simple Additive Weighting (SAW) strategy. At the first level, the nine performance criteria are used to rank all evaluated algorithms within each workload. These criterion-wise ranks are normalized and aggregated with equal weights to obtain a workload-level SAW score and workload rank. At the second level, the eight workload-level ranks are normalized and aggregated with equal workload weights to obtain one overall SAW score for each algorithm and its final rank.

This design separates the evaluation of individual scheduling properties from the assessment of overall consistency across heterogeneous workloads. It also ensures that differences in measurement units do not directly influence the aggregation, because SAW operates on normalized ranks rather than on raw metric values.

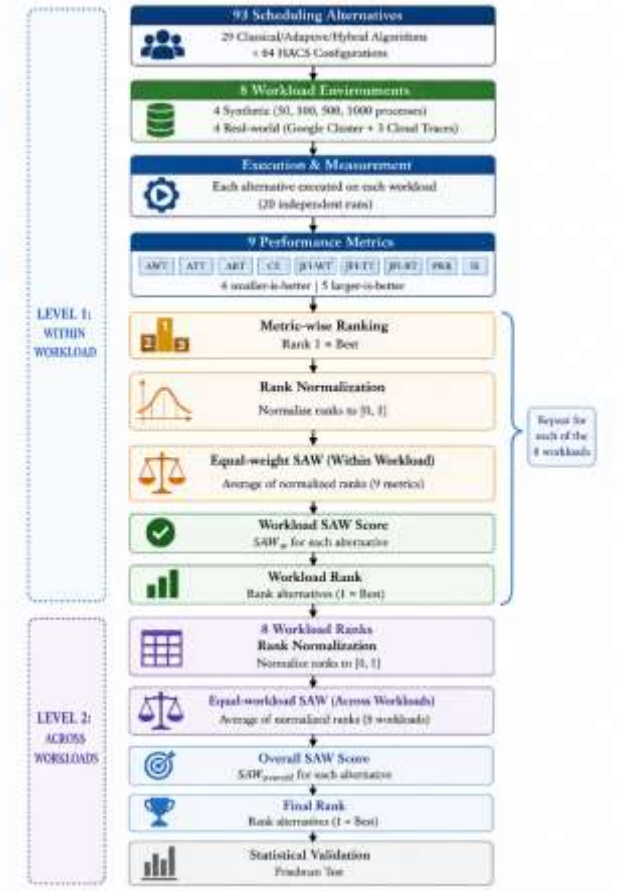


Figure 1. Overview of the proposed two-level rank-based multi-criteria evaluation framework.

Figure 1 illustrates the complete evaluation framework, consisting of two successive aggregation levels. At the first level, the nine performance criteria are independently ranked for all 93 evaluated alternatives within each workload, normalized using the rank transformation, and aggregated using equal-weight SAW to obtain a workload-level score and rank. At the second level, the eight workload-level ranks are normalized and aggregated with equal workload weights to obtain the overall SAW score and final ranking. Statistical validation is subsequently applied as a complementary stage to assess whether the observed performance differences are supported by statistical evidence.

3.2 Performance Evaluation Metrics

The framework evaluates nine complementary criteria covering scheduling efficiency, responsiveness, overhead, fairness, priority handling, and starvation resistance. The criteria are selected to capture both conventional scheduling objectives and dimensions that cannot be adequately represented by waiting time and turnaround time alone.

3.2.1 Average Waiting Time (AWT)

Average Waiting Time measures the average time that processes spend waiting in the ready queue before receiving CPU service. It is a conventional indicator of scheduling efficiency, and lower values indicate shorter queueing delays.

$$AWT = 1/N \sum_{i=1}^{N} WT_i$$

where WT_i is the waiting time of process i and N is the number of processes. AWT is a cost criterion because lower values are preferred.

3.2.2 Average Turnaround Time (ATT)

Average Turnaround Time represents the average elapsed time from process arrival until completion. It captures both waiting and execution periods and therefore provides an overall measure of process completion efficiency.

$$ATT = (1/N) \sum_{i=1}^{N} TT_i$$

where TT_i is the turnaround time of process i. ATT is a cost criterion, with lower values indicating better performance.

3.2.3 Average Response Time (ART)

Average Response Time measures the average interval between process arrival and its first allocation of CPU time. It captures initial scheduler responsiveness, which is especially relevant to interactive and time-sharing workloads.

$$ART = (1/N) \sum_{i=1}^{N} RT_i$$

where RT_i is the response time of process i. Lower ART is preferred, so ART is classified as a cost criterion.

3.2.4 Context Switches (CS)

Context Switches represent the number of process-to-process CPU switching events generated during a scheduling run. Although switching is necessary in preemptive and time-sharing systems, excessive switching introduces scheduling overhead and can interrupt useful execution. Therefore, lower CS values are preferred.

3.2.5 Jain's Fairness Indices

Efficiency measures do not reveal how evenly scheduling outcomes are distributed among processes. Jain's Fairness Index is therefore calculated separately for waiting time, turnaround time, and response time.

$$JFI = (\sum_{i=1}^{N} x_i)^2 / [N \sum_{i=1}^{N} x_i^2]$$

where x_i represents the relevant process-level performance quantity. The resulting indices are JFI-WT for waiting time, JFI-TT for turnaround time, and JFI-RT for response time. The index ranges from 0 to 1, with values closer to 1 indicating a more balanced distribution. All three JFI measures are therefore benefit criteria.

3.2.6 Priority Respect Ratio (PRR)

PRR = Priority-respecting decisions / Total scheduling decisions

Higher PRR indicates stronger adherence to process priorities; therefore, PRR is a benefit criterion.

3.2.7 Starvation Index (SI)

Starvation Index evaluates resistance to excessive waiting. In this framework, a process is classified as experiencing excessive waiting when its waiting time exceeds the average waiting time of the corresponding workload. The index is defined as:

$$SI = 1 - (N_s / N)$$

where N_s is the number of processes whose waiting time exceeds the workload average and N is the total number of processes. Higher SI indicates that a larger proportion of processes remain at or below the workload-level waiting-time reference and therefore indicates stronger resistance to excessive waiting.

Table 3.1. Performance evaluation criteria and optimization directions.

Criterion	Description	Type	Preferred Direction
AWT	Average Waiting Time	Cost	Lower is better
ATT	Average Turnaround Time	Cost	Lower is better
ART	Average Response Time	Cost	Lower is better
CS	Context Switches	Cost	Lower is better
JFI-WT	Jain's Fairness Index—Waiting Time	Benefit	Higher is better
JFI-TT	Jain's Fairness Index—Turnaround Time	Benefit	Higher is better
JFI-RT	Jain's Fairness Index—Response Time	Benefit	Higher is better
PRR	Priority Respect Ratio	Benefit	Higher is better
SI	Starvation Index	Benefit	Higher is better

3.3 Metric-wise Ranking

The raw values of the nine criteria have different units, scales, and optimization directions. Rather than applying SAW directly to these raw values, the framework first ranks all evaluated algorithms separately for every criterion within each workload. This converts heterogeneous measurements into a common ordinal representation.

Let the set of evaluated algorithms and configurations be:

$$A = \{a_1, a_2, \dots, a_n\}, \qquad n = 93$$

For each workload k and criterion j, the algorithms are ordered according to the criterion's optimization direction. Rank 1 always represents the best observed performance. For AWT, ATT, ART, and CS, the smallest value receives Rank 1. For JFI-WT, JFI-TT, JFI-RT, PRR, and SI, the largest value receives Rank 1.

$$r_{ijk} = rank\ of\ algorithm\ i\ for\ criterion\ j\ under\ workload\ k$$

The ranking is performed independently for each criterion and workload. Thus, the framework does not assume that an algorithm that is strong in one metric must also be strong in another. The resulting nine ranks provide the basis for the first SAW aggregation.

3.4 Rank Normalization

After metric-wise ranking, the ranks are normalized to a common preference scale. Because 93 algorithms/configurations are evaluated, the normalized rank is calculated using a linear transformation in which Rank 1 maps to 1 and Rank 93 maps to 0.

$$q_{ijk} = (n - r_{ijk}) / (n - 1)$$

$$q_{ijk} = (93 - r_{ijk}) / 92$$

where r_{ijk} is the rank of algorithm i for criterion j under workload k and q_{ijk} is the corresponding normalized preference value. Therefore:

$$Rank\ 1 \rightarrow 1.000000 \ ; \ Rank\ 93 \rightarrow 0.000000$$

Intermediate ranks are mapped proportionally to [0,1]. Since the metric-wise ranking has already incorporated the cost/benefit direction, the normalized values all have the same interpretation: higher values represent better relative performance.

This rank normalization is central to the framework. It prevents metrics with large numerical magnitudes from dominating the aggregation and makes the nine criteria directly comparable. It also makes the evaluation less dependent on the original units of measurement.

3.5 Rank-Based SAW Aggregation Within Each Workload

The normalized metric ranks are aggregated using Simple Additive Weighting (SAW). The workload-level SAW score represents the overall multi-criteria performance of each algorithm within a particular workload.

Equal weighting is adopted across the nine performance criteria so that efficiency, responsiveness, overhead, fairness, priority compliance, and starvation resistance contribute equally to the aggregate evaluation.

$$w_j = 1/9 = 0.111111 \ldots$$

$$\sum_{j=1}^{9} w_j = 1$$

$$SAW^{W}_{ik} = \sum_{j=1}^{9} w_j \, q_{ijk}$$

$$SAW^{W}_{ik} = (1/9) \sum_{j=1}^{9} [(93 - r_{ijk}) / 92]$$

The resulting score lies between 0 and 1. A higher workload-level SAW score indicates stronger overall relative performance across the nine criteria. Importantly, this score is derived from criterion ranks, not from direct aggregation of the raw AWT, ATT, ART, CS, JFI, PRR, and SI values.

3.5.1 Distinction Between Evaluation SAW and HACS-DTQSAW

The SAW procedure used in this framework is an evaluation mechanism and is conceptually separate from the internal decision mechanism of HACS-DTQSAW. The HACS-DTQSAW scheduler uses its internal Arrival Time, Burst Time, and Priority parameters to influence scheduling decisions. In contrast, the evaluation SAW operates after execution and assesses the resulting performance using the nine independent criteria. Thus, the two uses of SAW serve different purposes and must not be interpreted as the same weighting process.

3.6 Workload-Level Ranking

After the workload-level SAW scores are calculated for all 93 algorithms/configurations, the algorithms are ranked in descending order of their SAW scores. The highest score receives Rank 1.

$$R_{ik} = rank\ of\ algorithm\ i\ according\ to\ SAW^{W}_{ik}$$

This process is repeated independently for all eight workloads. The resulting workload-level rank therefore summarizes how each algorithm performs when the nine performance dimensions are considered jointly under a specific workload.

3.7 Overall Rank-Based SAW Across Workloads

The overall evaluation aggregates the eight workload-level ranks rather than directly re-aggregating the nine performance metrics. This second level measures the consistency of each algorithm across heterogeneous workload environments.

For each algorithm i, the eight workload-level ranks are denoted by R_{ik}, where k = 1,…,8. These ranks are normalized using the same transformation applied at the metric level:

$$Q_{ik} = (n - R_{ik}) / (n - 1)$$

$$Q_{ik} = (93 - R_{ik}) / 92$$

Each workload is assigned equal importance:

$$w_k = 1/8 = 0.125$$

$$\sum_{k=1}^{8} w_k = 1$$

The overall SAW score is then calculated as the mean of the eight normalized workload-rank values:

$$SAW^{overall}_{i} = \sum_{k=1}^{8} w_k \, Q_{ik}$$

$$SAW^{overall}_{i} = (1/8) \sum_{k=1}^{8} [(93 - R_{ik}) / 92]$$

Consequently, an algorithm that consistently obtains high workload ranks receives a high overall SAW score, whereas an algorithm with strong performance in only a small subset of workloads is penalized by weaker ranks in the remaining workloads.

Table 3.2. Two-level rank-based SAW aggregation.

Aggregation Level	Inputs	Normalization	Weighting	Output
Workload level	9 metric ranks	(93 − r)/92	1/9 per metric	Workload SAW score + workload rank
Overall level	8 workload ranks	(93 − R)/92	1/8 per workload	Overall SAW score + final rank



Figure 2. Detailed two-level rank-based SAW aggregation procedure.

3.8 Final Algorithm Ranking

The final ranking is obtained by ordering all algorithms and configurations in descending order of their overall SAW scores. The highest score receives Final Rank 1.

$$SAW^{\text{overall}}(1) \geq SAW^{\text{overall}}(2) \geq \ldots \geq SAW^{\text{overall}}(n)$$

The final rank is therefore a relative measure within the evaluated set of 93 algorithms/configurations and the eight selected workloads. It does not imply universal optimality across all CPU scheduling environments.

For example, HACS-DTQSAW (A0.05-B0.475-P0.475) has workload-level ranks of 4, 1, 2, 3, 4, 1, 7, and 4. The overall score is obtained by normalizing each of these ranks and taking their equal-weight mean:

$$SAW^{\text{overall}} = (1/8)\,[(93-4)/92 + (93-1)/92 + (93-2)/92 + (93-3)/92 + (93-4)/92 + (93-1)/92 + (93-7)/92 + (93-4)/92]$$

$$SAW^{\text{overall}} = 0.975543478$$

This example demonstrates that the reported overall score is derived directly from the eight workload-level ranks. Rank 1 contributes the maximum normalized value of 1, while lower ranks contribute proportionally smaller values.

3.9 Reproducibility and Computational Procedure

Reproducibility is treated as an integral property of the framework because comparative scheduling results can be affected by workload generation, metric definitions, ranking rules, and aggregation procedures. The proposed procedure therefore applies the same computational rules to every algorithm and workload.

- Execute every algorithm using the same workload realization and simulation conditions.
- Calculate the nine performance criteria using identical definitions for all algorithms.
- Rank the 93 algorithms/configurations independently for each criterion and workload.
- Assign Rank 1 to the best value according to the criterion's optimization direction.
- Normalize each rank using q = (93 − rank)/92.
- Aggregate the nine normalized metric ranks using equal weights of 1/9 to obtain the workload-level SAW score.
- Rank all algorithms according to the workload-level SAW score.
- Repeat the procedure for all eight workloads.
- Normalize each of the eight workload-level ranks using the same transformation.
- Aggregate the eight normalized workload ranks using equal weights of 1/8 to obtain the overall SAW score.
- Rank the algorithms by overall SAW score to obtain the final ranking.
- Apply statistical validation to the underlying performance results and/or rank structures as specified in the experimental analysis.

3.10 Statistical Validation

The rank-based SAW score provides an aggregate ordering, but aggregation alone does not establish whether the observed performance differences are statistically significant. Statistical validation is therefore incorporated as a complementary stage of the framework.

Because CPU scheduling performance data may not satisfy the assumptions required by parametric procedures, the non-parametric Friedman test is employed. The Friedman test is used to determine whether statistically significant differences exist among the evaluated algorithms across repeated or matched experimental conditions. The test provides an overall non-parametric assessment of performance differences without assuming normality. A significance level of $\alpha = 0.05$ is adopted throughout the analysis.

The Friedman test complements the SAW ranking by distinguishing descriptive differences in aggregate performance from differences supported by statistical evidence. However, the test provides an overall assessment of whether differences exist among the evaluated algorithms and does not identify the specific algorithm pairs responsible for those differences.

3.11 Framework Interpretation and Scope

The framework is intended as an evaluation methodology rather than a scheduling policy. It does not change the execution logic of FCFS, SJF, Priority, RR, MLFQ, adaptive algorithms, dynamic-quantum algorithms, priority-aware algorithms, hybrid algorithms, or HACS-DTQSAW configurations. Instead, it applies a common measurement and ranking procedure after the algorithms have completed their executions.

The two-level rank-based SAW design provides three important advantages. First, it prevents raw measurement scales from dominating the aggregation. Second, it gives equal importance to the nine performance dimensions and to the eight workload environments. Third, it evaluates consistency across workloads rather than allowing a single favorable workload to determine the overall result.

At the same time, the framework does not eliminate the need to inspect individual metrics. Because SAW is an additive aggregation method, strong performance in some dimensions can compensate for weaker performance in others. The individual metric results therefore remain essential for interpreting why an algorithm achieves a particular workload-level or overall rank. The aggregate score should consequently be considered a summary measure of relative multi-criteria performance rather than a replacement for detailed metric-level analysis.

4. EXPERIMENTAL METHODOLOGY AND SETUP

This section describes the experimental methodology, simulation environment, workload construction, implementation procedures, and reproducibility protocol used to evaluate the CPU scheduling algorithms. A unified Python-based simulation framework was developed to execute all evaluated algorithms under identical experimental conditions. The simulator uses a common process representation, maintains process execution states, records scheduling decisions and execution intervals, computes the performance criteria defined in Section 3, and stores the resulting measurements for subsequent comparative analysis.

The experimental design combines controlled synthetic workloads with heterogeneous real-world datasets. Synthetic workloads provide repeatable and systematically varied conditions across different workload sizes and process characteristics, whereas real-world workloads provide additional evaluation under practically observed execution patterns. All evaluated algorithms and HACS-DTQSAW configurations are subjected to the same workload instances, simulation environment, performance measurement procedures, and subsequent rank-based multi-criteria evaluation.

4.1 Experimental Environment and Simulator Architecture

The experiments were conducted using a custom discrete event-driven CPU scheduling simulator implemented in Python. The simulator was designed as a unified execution and measurement environment so that differences among algorithms arise from their scheduling policies rather than from differences in input representation, execution infrastructure, or performance calculation.

The simulator accepts standardized process records containing the attributes required by the evaluated scheduling policies. For each experiment, the original workload is preserved and an independent copy is supplied to each algorithm or configuration. This isolation is necessary because scheduling execution modifies dynamic process attributes such as remaining burst time, response time, waiting time, and completion time. Using independent copies ensures that the execution of one algorithm cannot affect the input state of another.

The simulation workflow consists of four principal stages: workload preparation, scheduling execution, performance measurement, and result storage. First, the workload is loaded and converted into the internal process representation. Second, the scheduling policy determines process selection and CPU allocation. Third, the completed execution records are used to calculate the nine performance criteria. Finally, the results are stored in a structured format for comparison and subsequent multi-criteria analysis.

This architecture provides a common computational foundation for classical, adaptive, dynamic-quantum, priority-aware, and hybrid scheduling approaches.

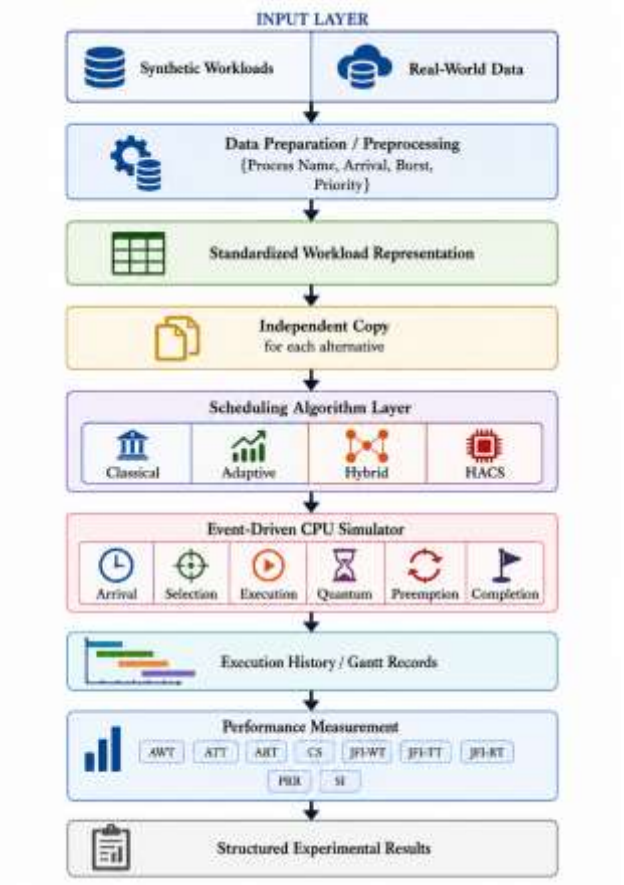


Figure 4. Architecture of the unified event-driven CPU scheduling simulation framework.

4.2 Event-Driven Simulation Model

The simulator uses a discrete event-driven model in which scheduling decisions are triggered by changes in process and CPU execution states rather than by unnecessary advancement of the simulation clock one unit at a time.

The principal events include process arrival, CPU allocation, time-quantum expiration, process completion, and reinsertion of unfinished processes into the ready queue. At each scheduling point, the simulator identifies processes that have arrived and are eligible for execution. The scheduling policy then determines which process receives CPU service.

For preemptive Round-Robin-based algorithms, the selected process executes for its assigned time quantum or until completion, whichever occurs first. If the process remains unfinished when the quantum expires, its remaining execution time is updated and the process is returned to the ready queue. Dynamic Round-Robin variants may recalculate the quantum according to their respective scheduling mechanisms.

When a process completes, its completion time and associated performance information are recorded. If no process is currently available, the simulation clock advances to the next process arrival rather than performing unnecessary idle iterations. This event-oriented mechanism is particularly useful for large workloads

because execution intervals can be processed directly, reducing unnecessary computational operations.

The event-driven structure also provides a consistent execution history from which context switches, priority-respecting transitions, and other scheduling characteristics can subsequently be derived.

4.3 Process Representation and Execution States

Each process is represented internally using a logical Process Control Block (PCB). The process representation contains both static workload attributes and dynamic execution attributes.

The primary static attributes are: Process Name (unique identifier), Arrival Time (time at which the process becomes available for scheduling), Burst Time (total CPU execution time required), and Priority (scheduling priority, where a lower numerical value represents a higher priority).

During simulation, the following dynamic attributes are maintained: Remaining Time, Completion Time, Response Time, Waiting Time, and Turnaround Time.

The simulator models the principal process states as New, Ready, Running, and Terminated. A process initially exists in the New state, enters the Ready state when its arrival time is reached, and moves to the Running state when selected by the scheduler. After completing its required CPU execution, it enters the Terminated state. For preemptive algorithms, an additional transition occurs from Running to Ready when the allocated time quantum expires before completion.

The conceptual state flow is therefore: New → Ready → Running → Terminated, with the additional preemption transition Running → Ready. This common process representation allows all evaluated scheduling algorithms to operate on the same fundamental workload attributes while maintaining their individual scheduling logic.

4.4 Data Structures and Execution Records

The simulator uses lightweight Python data structures to support repeated scheduling experiments. Process records are represented internally using dictionaries, while collections of processes are maintained using lists. Pandas DataFrames are used for workload processing, metric aggregation, and comparative result management.

The ready queue is dynamically constructed from processes that have arrived and remain unfinished. Its ordering depends on the scheduling policy. For example, SJF-based algorithms select processes according to burst characteristics, priority-based algorithms use priority information, and Round-Robin-based algorithms maintain the sequence required for time-sliced execution.

During execution, the simulator maintains an execution history analogous to a Gantt chart. Each execution segment records the process identifier together with its start and finish times. These records provide the basis for determining execution transitions and calculating measures such as Context Switches and Priority Respect Ratio.

After an algorithm completes, its process-level results and execution history are converted into structured records. The calculated performance measures are then stored together with the corresponding algorithm identifier.

4.5 Evaluated Algorithms and Scheduling Policies

The experimental benchmark contains 93 evaluated algorithms/configurations. It consists of 29 previously published scheduling algorithms representing classical, adaptive, dynamic, priority-aware, and hybrid approaches, together with 64 configurations of HACS-DTQSAW.

Each HACS-DTQSAW configuration represents a different assignment of the internal scheduling weights associated with Arrival Time (A), Burst Time (B), and Priority (P). These configurations are treated as separate evaluated alternatives because their scheduling behavior differs according to the adopted internal weight combination. The multi-criteria evaluation procedure itself remains unchanged across all configurations.

Accordingly: 29 published algorithms + 64 HACS-DTQSAW configurations = 93 evaluated alternatives.

All 93 alternatives are executed using the same workload instances and evaluated using the same nine performance criteria.

The benchmark includes both non-preemptive and preemptive scheduling policies. FCFS, SJF, and Priority Scheduling are implemented as non-preemptive approaches. Once selected, a process executes until completion. In contrast, Round Robin and Round-Robin-based adaptive and hybrid algorithms use time-sliced execution and may preempt unfinished processes when their allocated quantum expires.

This common execution environment allows algorithms with substantially different scheduling mechanisms to be compared without changing the underlying experimental conditions.

4.6 Performance Measurement

Following each simulation, the resulting process records and execution history are used to calculate the nine performance criteria defined in Section 3: AWT, ATT, ART, CS, JFI-WT, JFI-TAT, JFI-RT, PRR, and SI.

4.6.1 Jain's Fairness Indices

Jain's Fairness Index is used to assess the distributional fairness of scheduling outcomes across processes. Because lower waiting, turnaround, and response times represent better outcomes, the implementation applies the Jain formulation to their reciprocal values rather than directly to the raw time measurements.

For a corresponding time-based measure T_i, the transformed value is:

$$x_i = 1 / T_i$$

The Jain Fairness Index is then calculated as:

$$JFI = (\sum x_i)^2 / [N \sum x_i^2]$$

where N is the number of processes and x_i is the reciprocal of the relevant process-level measure. This formulation is applied independently to waiting time, turnaround time, and response time, producing JFI-WT, JFI-TAT, and JFI-RT, respectively. The resulting values range from 0 to 1, with values closer to 1 indicating a more balanced distribution of the corresponding scheduling outcome.

The reciprocal transformation is important because directly applying the Jain formula to raw waiting or response times could interpret larger time values as greater fairness without accounting for their undesirable direction. The adopted formulation instead evaluates the distribution of scheduling outcomes while preserving the principle that lower temporal delays are preferable.

4.6.2 Priority Respect Ratio

The Priority Respect Ratio (PRR) is an operational metric defined in this study to quantify the extent to which the generated execution sequence follows the priority ordering specified in the workload.

For every transition between two different processes in the recorded execution sequence, the simulator compares the priority values of the previously executed and subsequently executed processes. Under the adopted priority convention, a smaller numerical value represents a higher priority. A transition is classified as priority-respecting when the priority of the previously executed process is less than or equal to that of the subsequently executed process.

PRR is calculated as the proportion of priority-respecting transitions among the evaluated process transitions. Therefore, a value closer to 1 indicates stronger adherence to the adopted priority ordering. When no process transition occurs, PRR is assigned a value of 1.

PRR is therefore treated as an operational definition adopted by this study, rather than as a universally standardized priority metric. The same definition is applied consistently to every evaluated algorithm and workload.

4.6.3 Starvation Index

The Starvation Index (SI) is an operational measure adopted in this study to assess resistance to excessive waiting.

For each workload, the average waiting time is first determined. A process is classified as experiencing excessive waiting when its waiting time exceeds the corresponding workload-level average. SI is then calculated as:

$$SI = 1 - (N_s / N)$$

where N_s is the number of processes whose waiting time exceeds the workload average and N is the total number of processes.

Higher SI values indicate that a larger proportion of processes have waiting times at or below the workload average and therefore indicate stronger resistance to excessive waiting under the adopted operational definition. As with PRR, SI is not presented as a universally standardized starvation measure. Its formulation constitutes the operational definition used consistently throughout this study.

4.6.4 Context Switches

Context Switches (CS) represent the number of transitions between different executing processes and are used as an indicator of scheduling overhead.

The simulator derives CS from the recorded execution segments. Because the first execution segment represents the initial CPU allocation rather than a transition from one process to another, the number of context switches is calculated as:

$$CS = max(0, N_s - 1)$$

where N_s represents the number of execution segments in the recorded execution history. Lower CS values indicate fewer process-switching events and therefore lower scheduling-switching overhead.

Collectively, the nine criteria provide complementary information about scheduling behavior. AWT and ATT represent temporal efficiency, ART captures initial responsiveness, CS reflects switching overhead, the three JFI measures characterize fairness, PRR measures priority compliance, and SI evaluates resistance to excessive waiting.

4.7 Synthetic Workloads

Controlled synthetic workloads were generated using a Python-based workload-generation procedure to evaluate scheduling behavior under progressively increasing workload sizes and different process characteristics.

Four workload sizes were considered: 50 processes (Low Load), 100 processes (Medium Load), 500 processes (High Load), and 1000 processes (Very High Load).

For each process size, four workload scenarios were generated: Light Load, Heavy Load, Priority-Sensitive, and Starvation-Prone.

The scenarios were designed to vary workload characteristics rather than merely the number of processes.

Five independent workload realizations were generated for every combination of workload size and scenario. Therefore:

$$4\ sizes \times 4\ scenarios \times 5\ runs = 80\ synthetic\ workload\ files$$

The five realizations provide independent workload samples while allowing all algorithms to be tested on identical input conditions for each realization.**4.7.1 Arrival-Time Generation**

Process arrival behavior was generated using a Poisson-based inter-arrival model. For each workload, inter-arrival increments were sampled according to a scenario-specific parameter, and process arrival times were obtained from their cumulative values.

Workload Scenario	Poisson Parameter
Light Load	3
Heavy Load	1
Priority-Sensitive	2
Starvation-Prone	0.5

Under the adopted generation procedure, smaller parameter values produce more closely spaced arrivals. Consequently, Heavy Load and Starvation-Prone workloads generate denser process arrivals than the Light Load condition.

4.7.2 Burst-Time Generation

Burst times were generated using a mixed distribution containing short, medium, and long processes.

Process Category	Burst-Time Range
Short	50–199
Medium	201–5999
Long	6001–9999

The proportions of these categories were adjusted according to the workload scenario:

Scenario	Short	Medium	Long
Light Load	25%	50%	25%
Heavy Load	25%	50%	25%
Priority-Sensitive	50%	30%	20%
Starvation-Prone	20%	30%	50%

The generated burst-time values were randomly shuffled before assignment to individual processes to prevent systematic association between process identifiers and burst-time categories.

4.7.3 Priority Generation

Priority values were generated as discrete levels from 1 to 6, where lower numerical values represent higher priority.

The levels were grouped into high priority (1–2), medium priority (3–4), and low priority (5–6).

Scenario	High	Medium	Low
Light Load	15%	60%	25%
Heavy Load	15%	60%	25%
Priority-Sensitive	50%	35%	15%
Starvation-Prone	15%	35%	50%

Priority values were randomly shuffled before assignment to individual processes. Each generated process therefore contains four principal workload attributes: Process Name, Arrival, Burst, and Priority. The generated workloads were stored as Excel files and subsequently used as standardized simulator inputs.

4.8 Aggregation of Synthetic Workloads

Although 80 synthetic workload files were generated, the multi-criteria comparison treats the four process sizes as four aggregate synthetic workloads: 50, 100, 500, and 1000 processes.

For each process size, the results from the four workload scenarios and five independent realizations are aggregated using the arithmetic mean. Thus, each process size is represented by the combined results of 4 scenarios × 5 runs = 20 synthetic workload realizations.

For each algorithm/configuration and each performance criterion, the corresponding values from the 20 realizations are averaged to produce a single representative result for that workload size. This procedure reduces dependence on any individual workload realization while preserving the differences associated with workload scale.

Accordingly, the 80 generated files are transformed into four aggregate synthetic workloads. The resulting four workloads are then treated consistently with the four real-world workloads in the subsequent ranking and overall SAW evaluation.

This aggregation procedure is important because the final multi-criteria evaluation uses eight workload-level rankings rather than 80 individual file-level rankings.

4.9 Real-World Workloads

To complement the controlled synthetic experiments, four real-world datasets were incorporated into the evaluation: Cloud Task Scheduling Dataset, Comprehensive Cloud Workload Performance Dataset, Google Cluster Traces v3, and PC Dataset.

These datasets provide heterogeneous workload characteristics and allow the evaluated scheduling alternatives to be examined beyond artificially generated process distributions.

4.9.1 Cloud Task Scheduling Dataset

The Cloud Task Scheduling Dataset contains 6,000 task records. For the scheduling experiments, execution time is used as burst time, task priority is extracted from the corresponding priority attribute, and arrival time is reconstructed from the available temporal attributes. Specifically, arrival time is derived from completion time, execution time, and waiting time using the relationship represented in the source dataset. The resulting records are transformed into the common process representation required by the simulator.

4.9.2 Comprehensive Cloud Workload Performance Dataset

The Comprehensive Cloud Workload Performance Dataset contains 5,000 workload records. Task start time is used to represent arrival time, task execution time is used as burst time, and the corresponding priority attribute is used as process priority. The preprocessing retains the temporal and execution characteristics necessary for CPU scheduling while converting the dataset into the common input structure used by the simulator.

4.9.3 Google Cluster Traces v3

The Google Cluster Traces v3 dataset contributes 4,887 task records to the experimental evaluation. The relevant scheduling attributes are extracted from the trace, with task start-time information used to represent arrival time and execution duration derived from the available execution timestamps. The corresponding priority information is also extracted and mapped to the common process representation required by the simulator.

4.9.4 PC Dataset

The PC Dataset contains 182 filtered process records collected from a Windows-based personal computing environment using a Python monitoring procedure based on the psutil library. The dataset contains process-level execution information, including process creation and termination information, CPU execution time, and operating-system priority information. For the scheduling experiments, the recorded arrival information is used as arrival time, CPU execution time is represented as burst time, and the available priority classification is mapped to the simulator's priority representation.

4.9.5 Common Preprocessing

Before simulation, all four real-world datasets are converted into the same process representation used for the synthetic workloads. The preprocessing therefore produces, for each record: {Name, Arrival, Burst, Priority}. This common representation allows the same simulation infrastructure to process both synthetic and real-world workloads without modifying the scheduling algorithms.

Dataset-specific source information and preprocessing details are provided in Appendix D.

4.10 Experimental Repetitions and Random Seeds

Five independent synthetic workload realizations were generated for every combination of workload size and workload scenario. The five runs were controlled using predetermined random seeds: 100, 200, 300, 400, and 500. The same seed sequence was used across workload sizes and scenarios. Once a workload realization was generated, the resulting file was retained and reused for all evaluated algorithms.

This design is important for ensuring paired comparisons among algorithms. The algorithms do not receive independently generated workloads; instead, every algorithm/configuration is evaluated using exactly the same workload realization for each experimental condition. Consequently, differences in performance can be attributed to differences in scheduling behavior rather than to different randomly generated input data.

4.11 Implementation Environment

All experiments were conducted on a desktop computer equipped with an Intel Core i5-1145G7 processor, 8 GB of RAM, and a solid-state drive, running Microsoft Windows 11 (64-bit).

The simulation framework was implemented in Python 3.9 using a modular architecture separating workload

processing, scheduling logic, performance measurement, and result aggregation.

Component	Purpose
Python 3.9	Simulation and algorithm implementation
Pandas	Dataset processing and result aggregation
NumPy	Numerical computation and workload generation
Statistics module	Statistical calculations
Matplotlib	Result visualization
Microsoft Excel	Workload and experimental-result storage

All experiments were executed through an automated computational workflow. The same hardware and software environment was maintained throughout the experimental process.

4.12 Reproducibility Protocol

Reproducibility was incorporated into the experimental design at the workload, simulation, and result-processing levels.

First, synthetic workload generation was controlled using the predetermined random seeds. Second, each generated workload realization was saved as an independent Excel file, allowing the exact experimental input to be reused without regeneration.

Third, every scheduling algorithm/configuration received an independent copy of the original workload. This prevents execution-state modifications from propagating between algorithms.

Fourth, all evaluated alternatives use the same process representation, simulation environment, performance definitions, and result-processing procedures. Fifth, the metric calculation and result-collection procedures are automated, reducing manual intervention and the possibility of inconsistent processing.

Finally, all simulation results are stored in structured output files. This provides a persistent record of the measurements used for the subsequent ranking and statistical analyses.

4.13 Experimental Procedure

Step 1 — Workload Preparation. Synthetic and real-world datasets are converted into the standardized process representation required by the simulator.

Step 2 — Independent Initialization. For each evaluated algorithm/configuration, an independent copy of the original workload is created and all dynamic process attributes are initialized.

Step 3 — Scheduling Execution. The selected scheduling policy controls CPU allocation according to its specific scheduling mechanism. Non-preemptive algorithms execute selected processes until completion, whereas preemptive algorithms may return unfinished processes to the ready queue after quantum expiration.

Step 4 — Execution Recording. Every CPU allocation is recorded with its process identifier, start time, and completion time. The execution history is retained as the simulated Gantt representation.

Step 5 — Metric Computation. The nine performance criteria are calculated from the completed process records and execution history.

Step 6 — Synthetic Workload Aggregation. For each synthetic process size, the metric results from the four workload scenarios and five realizations are combined using the arithmetic mean, producing one aggregate workload result for 50, 100, 500, and 1000 processes.

Step 7 — Multi-Criteria Evaluation. The workload-level metric results are passed to the rank-based multi-criteria procedure defined in Section 3. Each algorithm receives a rank for each individual performance criterion, and the resulting criterion ranks are aggregated using the adopted SAW procedure.

Step 8 — Overall Workload Ranking. The resulting workload-level scores are used to rank the 93 evaluated algorithms/configurations.

Step 9 — Overall Evaluation. The eight workload-level ranks are subsequently combined using the rank-based overall SAW procedure defined in Section 3 to obtain the overall SAW score and final ranking.

Step 10 — Statistical Validation. The resulting performance differences are subjected to the Friedman test described in Section 3.

4.14 Pseudocode

Algorithm 1. General CPU Scheduling Simulation Procedure

```
Input: Process workload W
       Set of 93 scheduling algorithms/configurations A
Load workload W
Convert W to the standardized process representation
For each algorithm a in A do
   Create an independent copy W_a of W
   Initialize process execution states
   Initialize simulation time
   Initialize execution history
   While unfinished processes remain do
      Identify processes that have arrived
      If no process is currently available then
         Advance simulation time to the next arrival
      Else
         Construct the ready queue according to
algorithm a
         Select the next process
         If a is non-preemptive then
            Execute the selected process until
completion
         Else
            Determine the current time quantum
            Execute the selected process
            until quantum expiration or completion
            If the process remains unfinished then
               Return it to the ready queue
            End If
         End If
         Record execution interval
         Update remaining execution time
         Update simulation time and process state
         If the process has completed then
            Record completion information
         End If
      End If
   End While
   Calculate AWT, ATT, ART, CS
   Calculate JFI-WT, JFI-TAT, JFI-RT
```

```
    Calculate PRR and SI
    Store the results for algorithm a

End For

Return comparative results for all 93 alternatives
```

Algorithm 2. Synthetic Workload Aggregation

```
Input: Results from 4 workload scenarios
       5 independent runs per scenario
       Workload size s
For each algorithm a do
   For each performance criterion c do
      Collect the 20 observations
      corresponding to 4 scenarios × 5 runs
      Calculate the arithmetic mean
      Store the aggregated value
      for algorithm a and criterion c
   End For
End For
Return one aggregate workload result for size s
```

5. RESULTS AND DISCUSSION

5.1 Multi-Metric Performance Results

The experimental evaluation demonstrates that CPU scheduling performance is inherently multi-dimensional and cannot be adequately characterized by a single performance metric. In this study, ninety-three scheduling configurations, including classical, hybrid, and HACS-DTQSAW configurations, were evaluated using nine complementary criteria covering efficiency, responsiveness, scheduling overhead, fairness, priority compliance, and starvation resistance. The evaluation considered both synthetic workloads of different sizes and real-world workload traces, allowing the performance of each configuration to be examined under diverse operating conditions.

The individual metric rankings confirm that no scheduling algorithm consistently dominates all evaluation criteria. Different algorithms achieve their strongest performance in different dimensions. For example, ERRDTQ obtains the best ranking in Average Waiting Time (AWT), SJF ranks first in Average Turnaround Time (ATT), SRBRR achieves the best Average Response Time (ART), and FCFS provides the lowest Context Switches (CS). PRIORITY achieves the highest Priority Respect Ratio (PRR), reflecting its explicit emphasis on priority ordering. These results illustrate the inherent trade-offs between efficiency, responsiveness, overhead, fairness, and priority compliance.

The results for fairness-related criteria further demonstrate why single-metric evaluation can be misleading. An algorithm may obtain excellent aggregate efficiency while producing an uneven distribution of waiting or response times among processes. Jain's fairness measures therefore provide complementary information that cannot be inferred directly from AWT, ATT, or ART. Similarly, PRR and SI capture aspects of scheduling quality that are not represented by conventional efficiency measures. The nine metrics jointly provide a broader representation of scheduling behavior by combining efficiency, responsiveness, overhead, fairness, priority handling, and resistance to excessive waiting.

In contrast to algorithms that perform strongly in only one or two dimensions, the HACS-DTQSAW configurations demonstrate a more balanced performance profile. In particular, HACS-DTQSAW (A0.05–B0.475–P0.475) remains competitive across the nine criteria rather than depending on a single exceptional metric. Its individual metric rankings remain within a relatively strong range, supporting the interpretation that its performance advantage results from a balanced scheduling behavior rather than isolated optimization of one objective.

This finding is important because the purpose of the proposed framework is not to identify the algorithm that is optimal for one metric, but to identify configurations that provide the most effective compromise among competing scheduling objectives. The individual metric results therefore provide the basis for the subsequent multi-criteria aggregation and overall ranking.

5.2 Overall Multi-Criteria Ranking Across Workloads

To obtain an overall assessment of scheduling performance, the rankings obtained from the nine evaluation criteria were aggregated using the Simple Additive Weighting (SAW) method. The SAW evaluation assigns equal importance to the nine criteria, ensuring that efficiency, fairness, responsiveness, scheduling overhead, priority compliance, and starvation resistance contribute equally to the overall assessment. The internal A, B, and P parameters of HACS-DTQSAW are independent of these evaluation weights and represent alternative configurations of the proposed scheduling algorithm rather than weights used by the evaluation framework.

The overall ranking provides a substantially different perspective from the individual metric results. While classical algorithms achieve excellent positions for particular criteria, the aggregated results favor configurations that maintain consistently strong performance across multiple dimensions. The best-performing configuration is HACS-DTQSAW (A0.05–B0.475–P0.475), which achieves an overall SAW score of 0.975543478 and the first position among all 93 evaluated configurations. Its workload-specific ranks are 4th, 1st, 2nd, 3rd, 4th, 1st, 7th, and 4th across the eight evaluated workload scenarios, demonstrating a consistently strong performance rather than dependence on a single workload.

The second position is obtained by HACS-DTQSAW (A0–B0.5–P0.5), with a SAW score of 0.971467391. It is followed by HACS-DTQSAW (A0.15–B0.425–P0.425), which achieves 0.937500000, and HACS-DTQSAW (A0.2–B0.4–P0.4), which achieves 0.906250000. The fifth-ranked configuration, HACS-DTQSAW (A0.175–B0.65–P0.175), obtains a score of 0.898097826. Thus, the five highest-ranked configurations are all HACS-DTQSAW variants.

Table 1. Top-Ranked Scheduling Configurations Based on Overall SAW Score

Rank	Scheduling configuration	SAW Score
1	HACS-DTQSAW (A0.05–B0.475–P0.475)	0.9755
2	HACS-DTQSAW (A0–B0.5–P0.5)	0.9715
3	HACS-DTQSAW (A0.15–B0.425–P0.425)	0.9375
4	HACS-DTQSAW (A0.2–B0.4–P0.4)	0.9063
5	HACS-DTQSAW (A0.175–B0.65–P0.175)	0.8981

The overall ranking results are further illustrated in Figure 4, which presents the top 20 algorithms and HACS-

DTQSAW configurations according to their aggregated multi-criteria SAW scores. The figure provides a direct visual comparison between the highest-ranked configurations and the strongest competing algorithms.

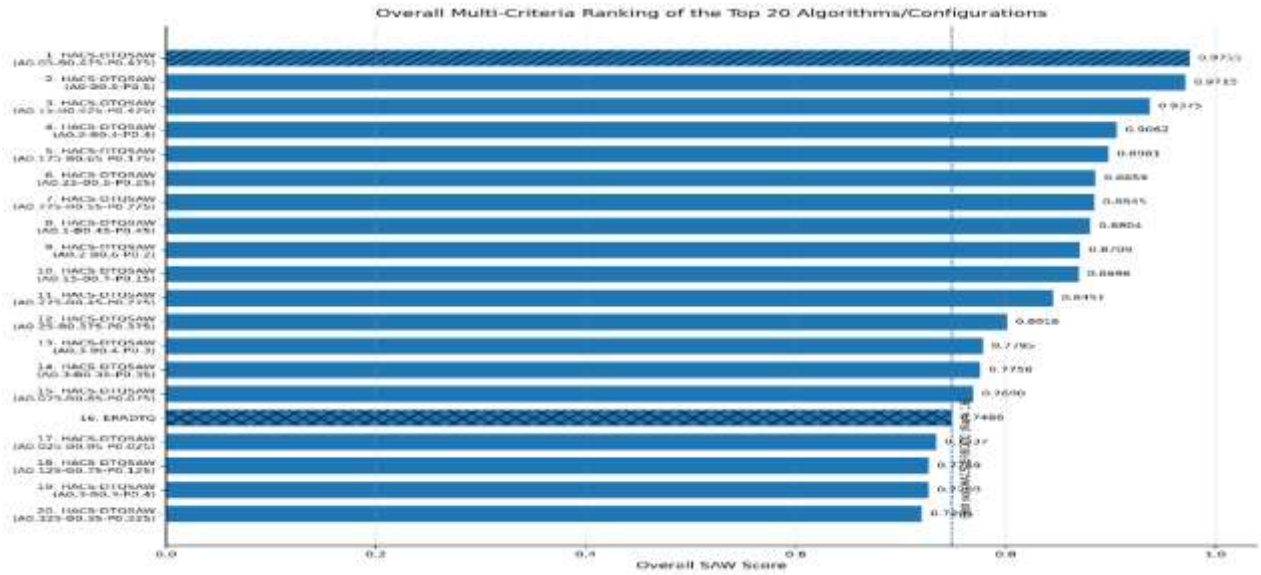


Figure 5. Top 20 overall multi-criteria ranking of CPU scheduling algorithms and HACS-DTQSAW configurations.

Figure 5 shows a clear concentration of HACS-DTQSAW configurations at the top of the overall ranking. The first 15 positions are occupied by HACS-DTQSAW variants, with the best configuration, HACS-DTQSAW (A0.05-B0.475-P0.475), achieving a SAW score of 0.9755. The second-ranked configuration, HACS-DTQSAW (A0-B0.5-P0.5), achieves a closely comparable score of 0.9715. In contrast, ERRDTQ is the highest-ranked non-HACS-DTQSAW algorithm, appearing at Rank 16 with a score of 0.7486. The difference of approximately 0.227 between the best proposed configuration and ERRDTQ indicates a substantial separation in overall multi-criteria performance. These results demonstrate that the proposed framework is not limited to a single high-performing configuration, but produces a broad set of configurations capable of achieving superior overall performance across the evaluated workloads.

The dominance of HACS-DTQSAW configurations becomes particularly evident when they are compared with the established scheduling algorithms. Among the previously published algorithms, ERRDTQ achieves the highest overall position at 16th, with a SAW score of approximately 0.7486. Optimized SJF follows at 36th, while SJF, PRIORITY, FCFS, and RR are ranked 54th, 60th, 75th, and 87th, respectively.

The complete distribution of the overall SAW scores is presented in Figure 6 to provide a broader view of the relative performance of all 93 evaluated scheduling configurations. Unlike Figure 4, which focuses on the highest-ranked configurations, Figure 5 shows the entire ranking distribution and illustrates the separation between the leading HACS-DTQSAW configurations and the remaining evaluated algorithms.

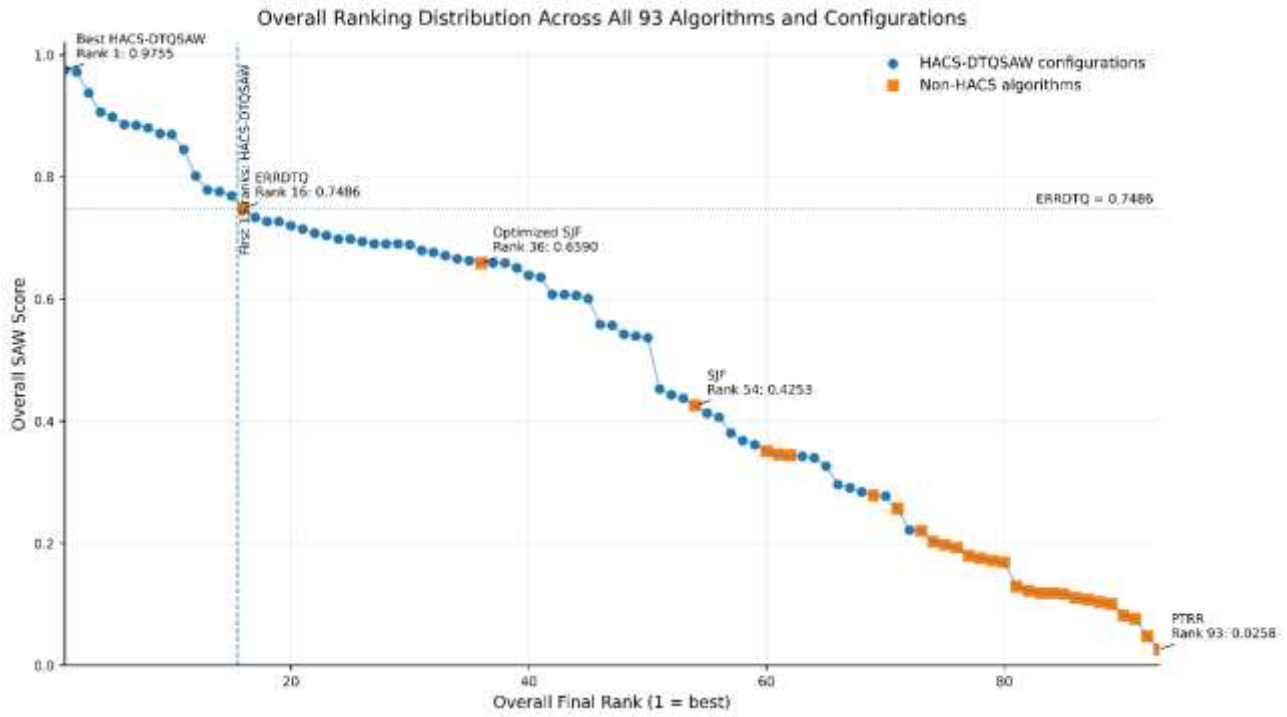


Figure 6. Overall SAW-score distribution across all 93 evaluated scheduling algorithms and HACS-DTQSAW configurations.

Figure 6 provides a complete view of the ranking distribution and shows that the high performance of HACS-DTQSAW is not restricted to a single configuration. The first 15 positions are occupied by HACS-DTQSAW configurations, forming a distinct high-performing group above the strongest non-HACS algorithm, ERRDTQ. The transition at Rank 16 is particularly evident, where the SAW score decreases from approximately 0.7690 for the fifteenth-ranked HACS-DTQSAW configuration to 0.7486 for ERRDTQ. Beyond this point, the overall scores generally decline across the remaining algorithms and configurations, with substantial separation from the highest-ranked group. This distribution indicates that the proposed scheduling framework contains a relatively broad region of high-performing configurations rather than relying exclusively on a single parameter setting.

These results demonstrate that strong performance in an individual metric does not necessarily translate into strong overall performance. SJF, for example, is highly competitive in turnaround-time efficiency but its overall position is substantially reduced when fairness, priority compliance, starvation resistance, and other criteria are considered simultaneously. Similarly, RR is commonly associated with fairness-oriented scheduling behavior, but its overall position is affected by the efficiency and overhead trade-offs introduced by frequent preemption. This further demonstrates that evaluating scheduling algorithms using only one conventional metric can produce incomplete or potentially misleading conclusions. While the overall SAW score identifies the best configurations, the workload-specific rankings provide additional insight into the consistency and robustness of their performance. Figure 7 summarizes the ranks achieved by the top HACS-DTQSAW configurations across the eight evaluated workloads and datasets, with ERRDTQ included as the strongest non-HACS-DTQSAW reference.



Figure 7. Rank consistency of the top HACS-DTQSAW configurations across the evaluated workloads and datasets. Lower rank values indicate better performance.

Figure 7 reveals that the overall superiority of HACS-DTQSAW is associated with its ability to maintain strong and relatively balanced rankings across heterogeneous workloads rather than relying on dominance in a single workload. The top-ranked configuration, HACS-DTQSAW (A0.05-B0.475-P0.475), achieves consistently strong positions across the evaluated workloads, including Ranks 1–4 in the 50-, 100-, 500-, and 1000-process workloads and Ranks 1–7 across the remaining workload categories. Although HACS-DTQSAW (A0-B0.5-P0.5) achieves better ranks in several individual workloads, its Rank 18 in the PC Dataset reduces its overall consistency. This explains why the A0.05-B0.475-P0.475 configuration obtains the highest aggregate SAW score despite not being the best configuration in every individual workload. The comparison with ERRDTQ further demonstrates this distinction: although ERRDTQ achieves a strong Rank 6 on the Google Cluster Traces v3 workload, its substantially weaker ranks on the synthetic workloads prevent it from achieving a comparable overall position.

5.3 HACS-DTQSAW Weight Configuration Analysis

The results also reveal a clear relationship between the internal HACS-DTQSAW weights and overall scheduling performance. The strongest configurations generally assign a relatively small weight to Arrival Time (A) while distributing the remaining weight approximately equally between Burst Time (B) and Priority (P). In particular, configurations with A ≤ 0.15 and approximately balanced B and P values dominate the highest positions in the global ranking.

This pattern indicates that the scheduling mechanism benefits from giving greater combined importance to execution characteristics and priority than to arrival order alone. Burst-time awareness contributes to efficient processor utilization and shorter waiting and turnaround times, while priority awareness provides better control over the treatment of processes according to their assigned importance. Maintaining a relatively low arrival-time weight prevents process arrival order from becoming overly influential in scheduling decisions.

The ranking also shows a progressive decline in performance as the arrival-time weight becomes dominant. Configurations assigning more than approximately 0.50 to A exhibit substantially weaker overall performance. Excessive emphasis on arrival time can cause the scheduler to favor process order without adequately considering execution requirements and priority, thereby reducing its ability to balance the competing objectives represented by the evaluation framework.

The same pattern is observed for configurations dominated by a single criterion. HACS-DTQSAW (A1–B0–P0), HACS-DTQSAW (A0–B0–P1), and HACS-DTQSAW (A0–B1–P0) perform substantially worse than the balanced configurations. This provides direct experimental evidence that optimizing one scheduling dimension alone is insufficient for achieving strong overall performance.

An important implication is that the proposed scheduler does not obtain its advantage simply by emphasizing one specific scheduling objective. Instead, its strongest performance emerges from a balanced interaction between

burst-time and priority information, supported by a relatively limited influence of arrival time. This balance allows the scheduler to capture efficiency and priority awareness simultaneously while avoiding excessive sensitivity to process arrival order.

The configuration analysis therefore provides an important explanation for the overall ranking results: the superiority of the leading HACS-DTQSAW configurations is associated with a balanced internal decision structure rather than an extreme optimization of a single scheduling characteristic.

5.4 Statistical Validation of the Results

Statistical validation was performed to determine whether the observed differences among scheduling configurations could reasonably be attributed to systematic performance differences rather than random variation. Because the evaluation compares 93 scheduling configurations, the Friedman test was conducted with 92 degrees of freedom. The Friedman test was independently applied to the evaluated workloads and performance criteria. The results show substantial and statistically significant differences among the scheduling configurations. Across the synthetic workloads, the Friedman statistics range approximately from 721.77 to 1801.86, with all reported p-values below 0.001. These results provide strong evidence that the evaluated configurations do not perform equivalently across the considered performance dimensions.

Table 2. Summary of Friedman Statistical Validation

Aspect	Result
Number of configurations	93
Degrees of freedom	92
Test	Friedman non-parametric test
Workloads	50, 100, 500, and 1000 processes
Friedman statistic range	≈721.77–1801.86
Significance	$p < 0.001$

The statistical evidence also indicates that performance differences become more pronounced as workload size increases. For example, the Friedman statistic for PRR increases from approximately 1151.10 for the 50-process workload to 1630.87 for the 1000-process workload. Similar behavior is observed in several fairness and robustness measures, indicating that the framework retains strong discriminative capability under increasingly demanding workloads.

The Friedman test provides an overall assessment of whether statistically significant differences exist among the evaluated scheduling configurations. Friedman results are interpreted as evidence of overall differences among the compared configurations rather than evidence of statistically significant differences between specific algorithm pairs.

The statistical findings therefore reinforce the SAW ranking results. The observed differences among the leading HACS-DTQSAW configurations and the other evaluated alternatives are supported by statistically significant overall differences in the Friedman analysis. Accordingly, the Friedman test is used as a complementary global validation of the multi-criteria ranking.

5.5 Integrated Discussion of the Findings

The combined metric-level, multi-criteria, configuration-level, and statistical results provide several important insights into CPU scheduling performance.

First, the results confirm that scheduling performance involves fundamental trade-offs among competing objectives. Algorithms that achieve strong efficiency in measures such as waiting or turnaround time may not provide comparable fairness or priority compliance. Conversely, algorithms emphasizing fairness or priority handling may incur efficiency or scheduling-overhead costs. The results for SJF, RR, FCFS, and PRIORITY illustrate these differences clearly and demonstrate why no single metric can provide a complete assessment of scheduling quality.

Second, the strongest HACS-DTQSAW configurations consistently combine low arrival-time influence with balanced burst-time and priority influence. Their high overall rankings indicate that this structure provides an effective compromise between efficiency and process-level considerations. Rather than strongly favoring one objective, these configurations maintain competitive performance across the nine evaluation criteria and across both synthetic and real-world workloads.

Third, highly biased configurations show weaker and less stable performance. Increasing the arrival-time weight excessively reduces the contribution of burst-time and priority information, while assigning the entire weight to a single scheduling factor limits the scheduler's ability to respond to the multi-dimensional nature of workload behavior. The experimental results therefore support the use of balanced internal weights rather than single-criterion dominance.

Fourth, the consistency of the leading configurations across workloads provides evidence of robustness. The best configuration achieves high positions across all eight workload scenarios rather than obtaining its first overall position through exceptional performance in only one dataset. This stability suggests that the balanced decision mechanism is less sensitive to changes in process distribution, workload scale, and workload characteristics.

Finally, the Friedman analysis strengthens these interpretations by demonstrating statistically significant overall differences among the 93 evaluated configurations. The combination of multi-criteria aggregation and non-parametric statistical validation therefore provides a more reliable basis for evaluating scheduling algorithms than isolated metric comparisons.

Overall, the findings demonstrate that effective CPU scheduling is better characterized by balanced multi-objective performance than by optimization of a single criterion. The proposed evaluation framework captures these trade-offs by jointly considering efficiency, responsiveness, scheduling overhead, fairness, priority compliance, and starvation resistance. Within this framework, HACS-DTQSAW configurations with low arrival-time weighting and approximately balanced burst-time and priority weighting achieve the strongest overall performance, with HACS-DTQSAW (A0.05–B0.475–

P0.475) emerging as the best-performing configuration across the complete evaluation.

6. CONCLUSION

This study presented a comprehensive multi-criteria evaluation framework for CPU scheduling algorithms that addresses the limitations of conventional single-metric comparisons. By jointly considering Average Waiting Time, Average Turnaround Time, Average Response Time, Context Switches, Jain's Fairness Indices, Priority Respect Ratio, and Starvation Index**,** the framework provides a unified assessment of efficiency, responsiveness, scheduling overhead, fairness, priority compliance, and resistance to excessive waiting.

The evaluation of 93 scheduling configurations across eight workload scenarios demonstrated that CPU scheduling performance is inherently multi-dimensional and involves significant trade-offs among competing objectives. Algorithms that perform well on individual metrics do not necessarily achieve strong overall performance when multiple criteria are considered simultaneously. In contrast, the highest overall rankings were consistently achieved by HACS-DTQSAW configurations that maintained a relatively low influence of Arrival Time while balancing Burst Time and Priority. In particular, HACS-DTQSAW (A0.05–B0.475–P0.475) achieved the highest overall performance, with an SAW score of 0.975543478 and Rank 1 among the 93 evaluated configurations. Its strong positions across synthetic and real-world workloads indicate that its performance was not dependent on a single workload or metric, but resulted from a more balanced scheduling behavior.

The study also demonstrated the value of evaluating scheduling algorithms through a rank-based multi-criteria aggregation approach rather than relying solely on raw metric values. The use of rank normalization, equal weighting, and SAW aggregation provided a consistent basis for comparing heterogeneous algorithms and configurations. The subsequent statistical analysis further strengthened the findings, with the Friedman test identifying statistically significant overall differences among the evaluated configurations.

Overall, the findings confirm that balanced multi-objective scheduling is more informative than single-objective optimization when assessing CPU scheduling algorithms**.** The proposed evaluation framework provides a reproducible and extensible methodology for benchmarking classical, adaptive, dynamic, priority-aware, and hybrid scheduling approaches under common experimental conditions. It can also serve as a general evaluation basis for future scheduling studies without modifying the scheduling logic of the algorithms being assessed.

Future research may extend the framework to multi-core, distributed, real-time, cloud, and heterogeneous computing environments**,** and may investigate alternative weighting strategies, including application-specific or adaptive weighting, where the relative importance of scheduling objectives changes dynamically with workload conditions.

REFERENCES

1. **Agha, A. E. A., & Jassbi, S. J. (2013).** A new method to improve round robin scheduling algorithm with quantum time based on harmonic-arithmetic mean (HARM). *International Journal of Information Technology and Computer Science, 5*(7), 56–62.
2. **Alhaidari, F., & Balharith, T. Z. (2021).** Enhanced round-robin algorithm in the cloud computing environment for optimal task scheduling. *Computers, 10*(5), 63.
3. **Arya, G. P., Nilay, K., & Prasad, D. (2018).** An improved round robin CPU scheduling algorithm based on priority of process. *International Journal of Engineering and Technology (UAE), 7*(4), 238–241.
4. **Dash, A. R., & Samantra, S. K. (2016).** An optimized round robin CPU scheduling algorithm with dynamic time quantum. *arXiv preprint arXiv:1605.00362*.
5. **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*(1), 12.
6. **Gupta, A. K., Yadav, N. S., & Goyal, D. (2016).** Design and performance evaluation of Smart Job First Dynamic Round Robin (SJFDRR) scheduling algorithm with smart time quantum. *Unpublished manuscript.*
7. **Hamayun, M., & Khurshid, H. (2015).** An optimized shortest job first scheduling algorithm for CPU scheduling. *Journal of Applied Environmental and Biological Sciences, 5*(12), 42–46.
8. **Hiranwal, S., & Saroj, D. K. (2011).** Adaptive round robin scheduling using shortest burst approach based on smart time slice. *Unpublished manuscript.*
9. **Iqbal, M., Ullah, Z., Khan, I. A., Aslam, S., Shaheer, H., Humayon, M., et al. (2023).** Optimizing task execution: The impact of dynamic time quantum and priorities on round robin scheduling. *Future Internet, 15*(3), 104.
10. **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*(10), 1831.
11. **Mishra, M. K. (2012).** An improved round robin CPU scheduling algorithm. *Journal of Global Research in Computer Science, 3*(6), 64–69.
12. **Mishra, M. K., & Rashid, F. (2014).** An improved round robin CPU scheduling algorithm with varying time quantum. *International Journal of Computer Science, Engineering and Applications, 4*(4), 1.
13. **Mody, S., & Mirkar, S. (2019).** Smart round robin CPU scheduling algorithm for operating systems. In *2019 4th International Conference on Electrical, Electronics, Communication, Computer Technologies and Optimization Techniques (ICEECCOT)* (pp. 309–316). IEEE.
14. **Mohanty, R., Behera, H. S., Patwari, K., & Dash, M. (2010).** Design and performance evaluation of a new proposed shortest remaining burst round robin (SRBRR) scheduling algorithm. In *Proceedings of the*

International Symposium on Computer Engineering & Technology (ISCET 17) (pp. 126–137).
15. **Panda, S. K., & Bhoi, S. K. (2014).** An effective round robin algorithm using min-max dispersion measure. *arXiv preprint arXiv:1404.5869*.
16. **Paul, T., Hossain, R., & Samsuddoha, M. (2019).** Improved round robin scheduling algorithm with progressive time quantum. *International Journal of Computer Applications, 178*(49), 30–36.
17. **Rajput, I. S., & Gupta, D. (2012).** A priority based round robin CPU scheduling algorithm for real time systems. *International Journal of Innovations in Engineering and Technology, 1*(3), 1–11.
18. **Shafi, U., Shah, M. A., Wahid, A., Abbasi, K., Javaid, Q., Asghar, M. N., & Haider, M. (2020).** A novel amended dynamic round robin scheduling algorithm for timeshared systems. *International Arab Journal of Information Technology, 17*(1), 90–98.
19. **Sharma, C., Sharma, S., Kautish, S., Alsallami, S. A., Khalil, E. M., & Mohamed, A. W. (2022).** A new median-average round robin scheduling algorithm: An optimal approach for reducing turnaround and waiting time. *Alexandria Engineering Journal, 61*(12), 10527–10538.
20. **Sharma, P. S., Kumar, S., Gaur, M. S., & Jain, V. (2021).** A novel intelligent round robin CPU scheduling algorithm. *International Journal of Information Technology, 14*(3), 1475–1482.
21. **Singh, A., Goyal, P., & Batra, S. (2010).** An optimized round robin scheduling algorithm for CPU scheduling. *International Journal on Computer Science and Engineering, 2*(7), 2383–2385.
22. **Stallings, W. (2011).** *Operating Systems: Internals and Design Principles*. Prentice Hall Press.
23. **Tanenbaum, A. S., & Bos, H. (2015).** *Modern Operating Systems* (4th ed.). Pearson.
24. **Yadav, R. K., Mishra, A. K., Prakash, N., & Sharma, H. (2010).** An improved round robin scheduling algorithm for CPU scheduling. *International Journal on Computer Science and Engineering, 2*(4), 1064–1066.
25. **Zohora, M. F., Farhin, F., & Kaiser, M. S. (2024).** An enhanced round robin using dynamic time quantum for real-time asymmetric burst length processes in cloud computing environment. *PLOS ONE, 19*(8), e0304517.
26. **Zouaoui, S., Boussaid, L., & Mtibaa, A. (2019).** Priority based round robin (PBRR) CPU scheduling algorithm. *International Journal of Electrical & Computer Engineering, 9*(1), 1–8.
27. **Benghuzi, K., & Alhenshiri, A. (2026).** A hybrid adaptive CPU scheduling algorithm using a dynamic time quantum and simple additive weighting. *Alqalam Journal of Science*. https://doi.org/10.69667/ajs.26608
28. Kolios, A., Mytilinou, V., Lozano-Minguez, E., & Salonitis, K. (2016). A comparative study of multiple-criteria decision-making methods under stochastic inputs. *Energies, 9*(7), 566. https://doi.org/10.3390/en9070566
29. Silberschatz, A., Galvin, P. B., & Gagne, G. (2013). *Operating System Concepts* (9th ed.). Wiley.
30. Ajithdari. (n.d.). *Comprehensive cloud workload performance*. Kaggle. https://www.kaggle.com/datasets/ajithdari/comprehensive-cloud-workload-performance
31. Derrickmwiti. (n.d.). *Google 2019 cluster sample*. Kaggle. https://www.kaggle.com/datasets/derrickmwiti/google-2019-cluster-sample
32. Programmer3. (n.d.). *Cloud task scheduling dataset*. Kaggle. https://www.kaggle.com/datasets/programmer3/cloud-task-scheduling-dataset