Journal of Technology Research (JTR)
Volume 4, Issue 2, (2026), pp 111-134, ISSN 3005-639X

Large-Scale Benchmarking of Classical, Adaptive, Hybrid, and Multi-Criteria CPU Scheduling Algorithms Across Synthetic and Real-World Workloads

Khalifa Benghuzi*[1], Anwar Alhenshiri[1]

[1]Department of Computer Science, Faculty of Information Technology, Misurata University, Misurata, Libya.

*Corresponding author email: alhenshiri@it.misuratau.edu.ly

Received: 27-06-2026 | Accepted: 12-08-2026 | Available online: 20-08-2026 | DOI:10.26629/jtr.2026.10

ABSTRACT

CPU scheduling is one of the major challenges in operating systems, requiring a balance among efficiency, responsiveness, overhead reduction, fairness, priority compliance, and starvation prevention. This study presents a comprehensive evaluation of classical, adaptive, hybrid, and multi-criteria CPU scheduling algorithms under unified experimental conditions. A total of 93 scheduling configurations were evaluated, including 29 established algorithms and 64 HACS-DTQSAW configurations, using a custom simulator. Experiments covered diverse workloads and real-world datasets, with nine performance metrics analyzed using the Friedman and Nemenyi statistical tests. Results revealed performance variations depending on workload characteristics; SJF and Optimized SJF achieved strong performance in selected metrics, while PRIORITY provided the best priority compliance. HACS-DTQSAW demonstrated high robustness and balanced performance, highlighting the importance of multi-criteria evaluation for selecting suitable CPU scheduling algorithms.

Keywords: CPU Scheduling; Performance Benchmarking; Multi-Criteria Decision Making.

التقييم المعياري واسع النطاق لخوارزميات جدولة وحدة المعالجة المركزية التقليدية والتكيفية والهجينة ومتعددة المعايير عبر أحمال العمل الاصطناعية والواقعية

خليفة بن غزي[1] ، أنور الهنشيري[1]

[1] قسم علوم الحاسوب، كلية تقنية المعلومات، جامعة مصراتة، مصراتة، ليبيا

ملخـــــص البحـــــث

تُعد جدولة المعالج من أهم تحديات أنظمة التشغيل، حيث تتطلب تحقيق توازن بين الكفاءة، وسرعة الاستجابة، وتقليل الحمل، والعدالة، والالتزام بالأولويات، ومنع حرمان العمليات. تقدم هذه الدراســة تقييمًا واســعًا لخوارزميات الجدولة التقليدية والتكيفية والهجينة ومتعددة المعايير ضمن ظروف تجريبية موحدة. تم تقييم 93 إعدادًا للجدولة تشمل 29 خوارزمية معروفة و64 إعدادًا من خوارزمية HACS–DTQSAW باستخدام محاكي مخصص. شملت التجارب أحمال عمل متنوعة ومجموعات بيانات واقعية، مع تحليل تسعة مقاييس أداء واستخدام اختباري Friedman و Nemenyi للتحقق الإحصائي. أظهرت النتائج اختلاف الأداء حسب طبيعة الحمل؛ حيث تفوقت SJF و Optimized SJF في بعض مؤشرات الأداء، بينما حققت PRIORITY أفضل توافق مع الأولويات. كما أثبتت HACS–DTQSAW قدرة عالية على تحقيق أداء متوازن، مما يؤكد أهمية التقييم متعدد المعايير في اختيار خوارزميات الجدولة.

الكلمات المفتاحية: جدولة المعالج؛ التقييم المعياري للأداء ؛ اتخاذ القرار متعدد المعايير.

1. INTRODUCTION

CPU scheduling is a fundamental operating-system problem because scheduling policies directly affect waiting time, turnaround time, response time, fairness, scheduling overhead, priority compliance, and starvation resistance (Dhamdhere, 2009; Stallings, 2011). These objectives are inherently conflicting; improving one may adversely affect others. Therefore, evaluating schedulers using a single metric or limited workload provides only a partial view of their effectiveness.

Although numerous classical, adaptive, and hybrid scheduling algorithms have been proposed, they are often evaluated under different workloads, experimental settings, and performance measures. Recent developments in dynamic time-quantum, priority-based, and adaptive Round Robin scheduling further demonstrate the diversity of scheduling strategies and their varying performance characteristics (Iqbal et al., 2023; Zohora et al., 2024). Such methodological variation makes direct comparison difficult and limits assessment of whether a scheduler remains robust across heterogeneous workloads. Comprehensive comparisons of different scheduling families under a common framework and multiple competing objectives therefore remain necessary.

This study addresses this gap through a large-scale benchmark evaluating 29 established scheduling algorithms and 64 HACS-DTQSAW configurations, resulting in 93 scheduling configurations under unified experimental conditions. HACS-DTQSAW combines Simple Additive Weighting (SAW) for multi-criteria process selection with a Dynamic Time Quantum (DTQ) mechanism. Its 64 configurations systematically vary the relative importance of Arrival Time, Remaining Burst Time, and Priority, enabling evaluation of both performance and robustness across different scheduling preferences (Benghuzi & Alhenshiri, 2026). SAW provides a weighted aggregation mechanism for combining normalized criteria in multi-criteria decision problems (Kaliszewski & Podkopaev, 2016; Vafaei et al., 2022).

The benchmark uses four synthetic workloads containing 50, 100, 500, and 1000 processes and four real-world datasets: the Cloud Task Scheduling Dataset, Comprehensive Cloud Workload Performance Dataset, Google Cluster Traces v3, and PC Dataset. Performance is assessed using nine complementary metrics covering efficiency, responsiveness, scheduling overhead, fairness, priority compliance, and starvation resistance. Statistical analysis includes the Friedman test to determine whether observed performance differences are statistically significant.

The results reveal substantial performance variation across scheduling approaches and confirm that scheduler effectiveness depends on workload characteristics and evaluation objectives. While algorithms such as SJF and Priority achieve strong performance for specific objectives, their advantages do not consistently extend across heterogeneous workloads and competing criteria. HACS-DTQSAW demonstrates stronger cross-workload robustness, with the **A0.05-B0.475-P0.475** configuration achieving the highest overall benchmark position and seven HACS-DTQSAW configurations occupying the first seven positions. This concentration suggests that the observed advantage extends beyond a single weight combination to a relatively robust region of the HACS-DTQSAW weight space.

The main contribution of this study is a unified benchmark comparing classical, adaptive, hybrid, and multi-criteria scheduling approaches across heterogeneous workloads and competing objectives. By considering overall performance, cross-workload consistency, and statistical evidence rather than a single metric or workload, the study provides empirical evidence that multi-criteria adaptive scheduling can achieve strong overall

robustness while confirming that no scheduling policy is universally optimal.

The remainder of this paper is organized as follows. Section 2 reviews the evaluated scheduling approaches and establishes the research gap. Section 3 describes the experimental framework, workloads, metrics, and statistical procedures. Section 4 presents the experimental results and benchmark rankings. Section 5 provides statistical validation using the Friedman test. Section 6 discusses the principal findings, trade-offs, and cross-workload implications. Finally, Section 7 concludes the study and outlines future research directions.

2. RELATED WORK

CPU scheduling has evolved from classical policies toward adaptive, hybrid, and multi-criteria approaches that address competing requirements in processor allocation. The reviewed approaches differ mainly in how they prioritize processes, adapt the time quantum, combine scheduling mechanisms, or incorporate multiple decision criteria.

2.1 Classical CPU Scheduling Algorithms

First-Come-First-Served (FCFS), Shortest Job First (SJF), Priority Scheduling, and Round Robin (RR) provide the fundamental baselines for CPU scheduling evaluation. FCFS schedules processes according to arrival order but may suffer from the convoy effect. SJF favors short CPU bursts and can reduce average waiting and turnaround times, although long processes may experience extended waiting. Priority Scheduling explicitly favors higher-priority processes but may cause starvation of low-priority processes. RR provides time-sharing through a fixed time quantum, but its performance depends strongly on quantum selection and may incur additional context-switch overhead (Silberschatz et al., 2013; Stallings, 2011; Tanenbaum & Bos, 2015).

These algorithms form the classical baseline of the present benchmark, providing distinct reference points for evaluating adaptive and hybrid scheduling mechanisms.

2.2 Adaptive CPU Scheduling Algorithms

Adaptive scheduling primarily modifies the Round Robin time quantum or scheduling behavior according to process characteristics. Self-Adjusting RR adapts the quantum using burst-time information (Matarneh, 2009), while RR-HARM uses harmonic and arithmetic means to determine the quantum (Agha & Jassbi, 2013). RR-MMDM uses a min-max dispersion measure (Panda & Bhoi, 2014), and IRR-VTQ employs a varying time quantum (Mishra & Rashid, 2014).

Other adaptive developments include optimized dynamic-quantum RR (Dash & Samantra, 2016), Smart RR (Mody & Mirkar, 2019), Progressive RR (Paul et al., 2019), Amended Dynamic RR (Shafi et al., 2020), Intelligent RR (Sharma et al., 2021), Enhanced RR (Alhaidari & Balharith, 2021), Median-Average RR (Sharma et al., 2022), dynamic RR with priorities (Iqbal et al., 2023), and Enhanced Dynamic RR (Zohora et al., 2024). These approaches reduce dependence on a fixed quantum, but their effectiveness remains dependent on the adaptation mechanism and workload characteristics.

For the present benchmark, the adaptive group includes SATQRR, IRR Yadav, ORR, ARR-STS, Improved RR (IRR), RR-HARM, RR-MMDM, IRR-VTQ, ORR-DTQ, SRR, ADRR, ERR, MARR, and ERRDTQ, in addition to MLFQ. The cited literature directly supports several of these implementations; however, the final manuscript should provide an original source for each remaining algorithm label whose implementation is derived from a specific published method. The current reference list, for example, explicitly documents Matarneh (2009), Mishra (2012), Agha and Jassbi (2013), Panda and Bhoi (2014), Mishra and Rashid (2014), and the later adaptive RR studies.

2.3 Hybrid CPU Scheduling Algorithms

Hybrid scheduling combines complementary mechanisms to exploit their respective strengths. SRBRR combines shortest-remaining-burst selection with Round Robin execution (Mohanty et al., 2010), Optimized SJF improves burst-oriented scheduling (Hamayun & Khurshid, 2015), and SJFDRR combines shortest-job selection with dynamic Round Robin execution (Gupta et al., 2016). HybridSJF-DVQ combines SJF and Round Robin using a dynamic variable quantum (Elmougy et al., 2017).

Priority-oriented hybrid approaches incorporate priority into Round Robin scheduling. P-RR is represented by priority-based Round Robin approaches (Rajput & Gupta, 2012; Arya et al., 2018), while PBRR explicitly integrates priority into Round Robin scheduling (Zouaoui et al., 2019). The present benchmark additionally evaluates IRR-P, PTRR, and OPRRP as priority-oriented hybrid variants.

These approaches demonstrate that combining scheduling mechanisms can improve specific objectives, but the resulting performance depends on the interaction between the integrated policies and workload characteristics.

2.4 Multi-Criteria CPU Scheduling and HACS-DTQSAW

Multi-criteria CPU scheduling formulates process selection as a decision problem involving multiple scheduling attributes rather than a single rule. In this study, HACS-DTQSAW (Hybrid Adaptive CPU Scheduling with Dynamic Time Quantum and Simple Additive Weighting), proposed by Benghuzi and Alhenshiri (2026), represents the multi-criteria scheduling approach evaluated in the benchmark. It combines a Simple Additive Weighting (SAW) layer for process prioritization with a Dynamic Time Quantum (DTQ) mechanism for adaptive CPU allocation.

At each scheduling cycle, the processes in the ready queue are evaluated using three criteria: Arrival Time AT, Remaining Burst Time BT, and Priority PR. After normalization, the SAW score of process i is calculated as:

$$\boldsymbol{Score}_{\mathrm{i}} = \boldsymbol{w_{AT}}\, \boldsymbol{r_{AT_{\mathrm{i}}}} + \boldsymbol{w_{RBT}}\, \boldsymbol{r_{RBT_{\mathrm{i}}}} + \boldsymbol{w_{PR}}\, \boldsymbol{r_{PR_{\mathrm{i}}}}$$

subject to:

$$\boldsymbol{w_{AT}} + \boldsymbol{w_{BT}} + \boldsymbol{w_{PR}} = \mathbf{1},\ \ \boldsymbol{w_{AT}}, \boldsymbol{w_{BT}}, \boldsymbol{w_{PR}} \geq \mathbf{0}$$

Processes are ranked in descending order of their SAW scores, and the highest-ranked process is selected for execution. While SAW determines which process to execute next, DTQ determines its execution interval. The DTQ is calculated from the current distribution of remaining burst times using the mean and median:

$$\boldsymbol{DTQ} = \frac{[\boldsymbol{Mean}(\boldsymbol{RBT}) + \boldsymbol{Median}(\boldsymbol{RBT})]}{\mathbf{2}}$$

The DTQ is applied during the current scheduling cycle and recalculated at the beginning of the next cycle after process completion, pre-emption, or the admission of newly arrived processes. This enables the execution interval to adapt to changes in workload characteristics.

At each cycle, the ready processes are evaluated using Min-Max normalization of AT, BT, and PR values, and the highest-scoring process is selected. The DTQ is then calculated from the current RBT distribution, and the selected process executes for $\boldsymbol{min}(\boldsymbol{DTQ}, \boldsymbol{RBT}_{\mathrm{i}})$. Completed processes are removed, while unfinished processes are returned to the ready queue with updated remaining burst times. Newly arrived processes are admitted before the next decision, and the procedure is repeated until all processes complete (Benghuzi & Alhenshiri, 2026).

2.5 Research Gap

The literature demonstrates substantial development of classical, adaptive, and hybrid CPU scheduling algorithms, particularly through improvements to Round Robin and

combinations of burst-time and priority-based mechanisms. However, many studies evaluate individual algorithms or small groups of alternatives under specific workloads and limited performance measures. Differences in workload characteristics, experimental environments, and evaluation procedures make direct comparison across scheduling families difficult.

Moreover, strong performance on one scheduling objective does not necessarily translate into strong overall performance. An algorithm that minimizes waiting or turnaround time may provide weaker fairness, while priority-oriented approaches may improve priority compliance at the expense of starvation resistance or efficiency. Similarly, adaptive and hybrid algorithms may perform well under particular workload structures but behave differently as workload size and characteristics change.

This study addresses these limitations through a unified large-scale benchmark comprising 29 established scheduling algorithms and 64 HACS-DTQSAW configurations, resulting in 93 scheduling configurations. The benchmark evaluates four synthetic workloads containing 50, 100, 500, and 1000 processes and four real-world datasets: the Cloud Task Scheduling Dataset, Comprehensive Cloud Workload Performance Dataset, Google Cluster Traces v3, and PC Dataset. Nine complementary metrics assess efficiency, responsiveness, scheduling overhead, fairness, priority compliance, and starvation resistance, with statistical validation used to distinguish systematic performance differences from workload-specific or random variation.

This unified framework provides a common basis for comparing classical, adaptive, hybrid, and multi-criteria scheduling approaches while evaluating not only individual metrics but also cross-workload consistency and overall benchmark performance.

3. METHODOLOGY

3.1 Experimental Environment

The experimental evaluation was conducted using a custom event-driven CPU scheduling simulator implemented in Python 3.9. The simulator was designed specifically for process-level CPU scheduling and provides a consistent environment for evaluating all scheduling approaches under identical execution conditions.

All experiments were performed on the same hardware and software platform, consisting of an Intel Core i5-1145G7 processor, 8 GB RAM, and Windows 11 (64-bit). Pandas was used for workload management, NumPy for numerical computation, SciPy for statistical analysis, and Matplotlib for visualization. Identical workload instances and fixed random seeds were used to ensure consistency and reproducibility.

3.2 Experimental Simulation Framework

The benchmark uses a single-processor, preemptive, event-driven simulation model to provide a consistent and reproducible environment for evaluating CPU scheduling algorithms. The simulation framework integrates diverse synthetic and real-world workloads, a comprehensive set of classical, adaptive, hybrid, and multi-criteria scheduling algorithms, and a common Python-based simulation engine that manages process arrivals, execution, preemption, time-quantum expiration, completion, and context switching. The framework evaluates scheduling performance across multiple dimensions, including efficiency, fairness, priority compliance, and starvation resistance, followed by statistical analysis using the Friedman test at a significance level of $\alpha = 0.05$ and an overall ranking of the evaluated algorithms. This unified design ensures that all scheduling approaches are evaluated under the same workload conditions, simulation procedures, performance measures, and statistical evaluation criteria, as summarized in Fig1.

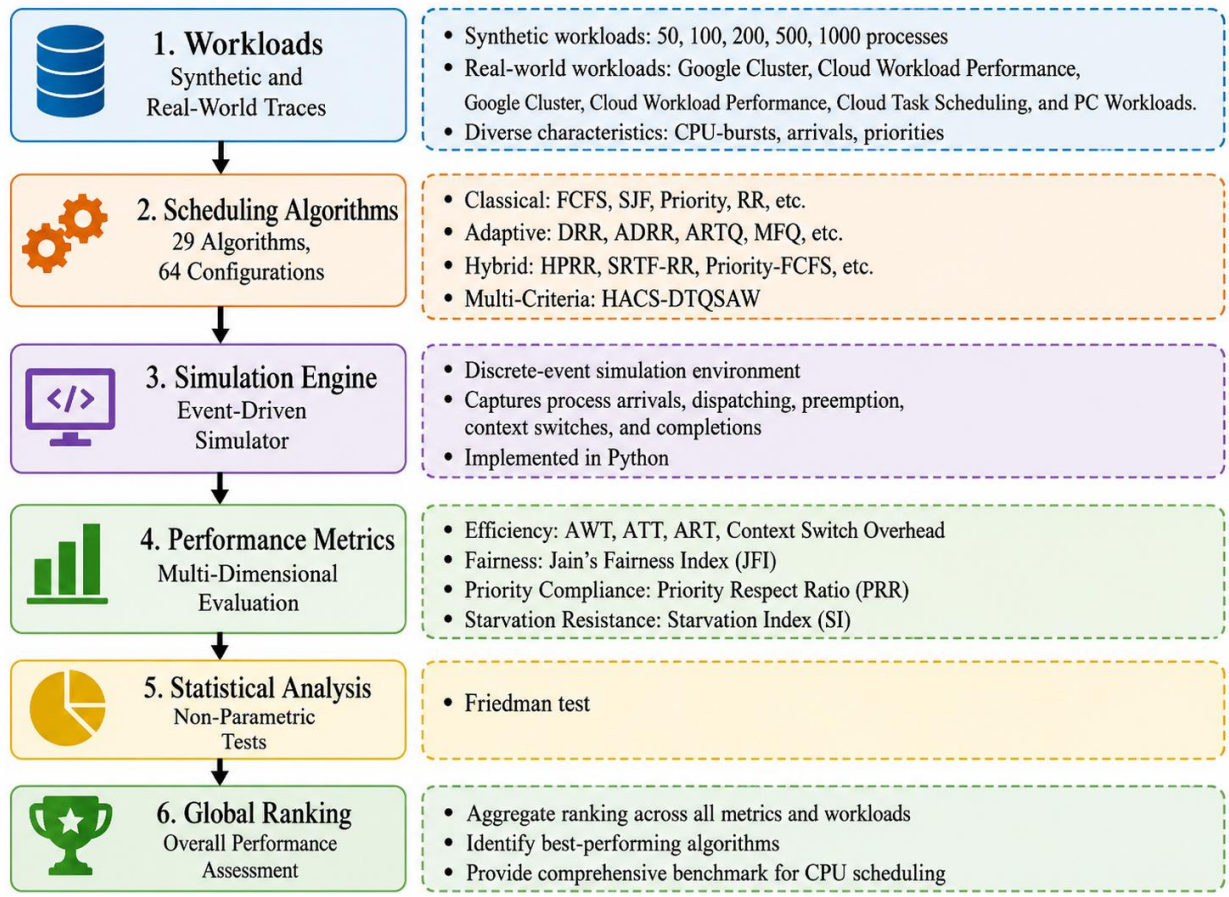

Fig1. Architecture of the Experimental Benchmark Framework.

3.3 Simulator Verification

The simulator was verified using analytically known scheduling examples for FCFS, SJF, Priority Scheduling, and Round Robin. Execution sequences, Gantt charts, completion times, waiting times, turnaround times, response times, and context-switch counts were compared with manually derived results.

Repeated executions with identical inputs produced identical scheduling sequences and performance metrics, confirming deterministic simulator behavior and consistency of the implementation.

3.4 Workloads and Benchmark Datasets

The benchmark combines four synthetic workloads containing 50, 100, 500, and 1000 CPU-bound processes with four real-world datasets. The synthetic workloads were generated using fixed random seeds, Poisson-distributed arrival times, heterogeneous burst times, and six priority levels. The real-world datasets comprise the Cloud Task Scheduling Dataset, Comprehensive Cloud Workload Performance Dataset, Google 2019 Cluster Sample, and a PC Process Dataset collected from a Windows computer using a custom Python monitoring script based on the psutil library.

For consistency, all real-world datasets were transformed into a common process representation containing Arrival Time, Burst Time, and Priority. For the Google 2019 Cluster Sample, Arrival Time was obtained from start_time, Burst Time was calculated as end_time − start_time, and Priority was taken from the priority field; the resulting dataset contains 405,893 task records. For the Comprehensive Cloud Workload Performance Dataset,Task_Start_Time,

Task_Execution_Time, and Job_Priority were used as Arrival Time, Burst Time, and Priority, respectively, resulting in 5,000 task records. For the Cloud Task Scheduling Dataset, Arrival Time was derived as Completion_Time_S − Execution_Time_S − Waiting_Time_S, Burst Time was represented by Execution_Time_S, and Priority was obtained from Task_Priority, producing 6,000 task records. The PC Process Dataset was collected by monitoring active processes on a Windows computer. Arrival Time was obtained from process creation time, Burst Time from CPU Execution Time, and Priority from the Process Priority Class. After preprocessing and filtering, 182 processes (P1–P182) were retained.

The resulting workloads provide controlled scalability through the synthetic scenarios and heterogeneous real-world conditions through cloud, cluster, and desktop process traces. This combination enables consistent comparison of all scheduling algorithms under both controlled and realistic workload conditions.

Table 1: Workloads Used in the Experimental Evaluation.

Workload	Type	Description
Synthetic-50	Synthetic	50 CPU-bound processes
Synthetic-100	Synthetic	100 CPU-bound processes
Synthetic-500	Synthetic	500 CPU-bound processes
Synthetic-1000	Synthetic	1000 CPU-bound processes
Cloud Task Scheduling Dataset	Real-world	Cloud task scheduling workload
Comprehensive Cloud Workload Performance Dataset	Real-world	Cloud workload execution records
Google Cluster Traces v3 Dataset	Real-world	Production cluster workload traces
PC Dataset	Real-world	Desktop process scheduling workload

3.5 Evaluated Scheduling Algorithms and HACS-DTQSAW Configurations

A total of 93 scheduling configurations were evaluated, comprising 29 established scheduling algorithms and 64 HACS-DTQSAW configurations. The benchmark includes classical, adaptive, dynamic, and hybrid scheduling approaches selected to provide diverse reference points for evaluating efficiency, responsiveness, fairness, priority compliance, and starvation resistance. All benchmark algorithms were implemented according to their published scheduling principles and evaluated under identical simulation conditions.

Table 2 summarizes the 29 benchmark algorithms and their principal references. The classical group comprises FCFS, SJF, Priority Scheduling, and Round Robin (RR), while the remaining algorithms represent adaptive, dynamic-quantum, and hybrid scheduling strategies. The inclusion of these approaches provides a broad comparison between established scheduling policies and more recent mechanisms based on adaptive quantum selection, burst-time awareness, and priority integration.

Table 2. Evaluated CPU Scheduling Algorithms**.**

Category	Algorithms
Classical	FCFS; SJF; Priority Scheduling; RR
Adaptive / Dynamic RR	MLFQ; Self-Adjusting RR; Improved RR (Yadav); Optimized RR; Adaptive RR (Shortest Burst); Improved RR (Mishra); RR-HARM; RR-MMDM; IRR-VTQ; Optimized RR-DTQ; Smart RR (SRR); Progressive RR; Amended Dynamic RR (ADRR); Intelligent RR; Enhanced RR; Median-Average RR; Dynamic RR with Priorities; Enhanced Dynamic RR

Hybrid	SRBRR; Priority-RR; Optimized SJF; SJFDRR; Hybrid SJF-RR with Dynamic Variable Quantum; Priority-Based Improved RR; PBRR

HACS-DTQSAW model was evaluated using 64 systematically generated weight configurations. The configurations assign weights to three decision criteria: Arrival Time (A), Remaining Burst Time (B), and Priority (P), subject to (A+B+P=1). Three groups of 21 configurations were generated by progressively increasing the weight of one criterion from 0 to 1 in increments of 0.05 while equally distributing the remaining weight between the other two criteria. An additional equal-weight configuration was included, resulting in 64 configurations in total. This design enables sensitivity analysis of the proposed scheduler and evaluates whether its performance remains robust under different decision preferences.

Table 3. HACS-DTQSAW Weight-Configuration Design.

Configuration Group	Dominant Criterion	Dominant Weight Range	Remaining Weights	Number
Arrival-dominant	A	0.00–1.00	(B=P=(1-A)/2)	21
Burst-dominant	B	0.00–1.00	(A=P=(1-B)/2)	21
Priority-dominant	P	0.00–1.00	(A=B=(1-P)/2)	21
Balanced	A, B, P	1/3 each	Equal weights	1
Total				64

The resulting configurations range from single-criterion emphasis, such as ((1,0,0)), ((0,1,0)), and ((0,0,1)), to balanced configurations such as ((0.05,0.475,0.475)), ((0.475,0.05,0.475)), and ((0.475,0.475,0.05)), together with the equal-weight configuration ((1/3,1/3,1/3)). Thus, the configuration space systematically covers different preferences for arrival time, remaining execution demand, and priority without requiring an exhaustive enumeration of all possible weight combinations.

All 93 configurations were evaluated using the same workloads, simulator, execution model, performance metrics, and statistical procedures. This controlled design isolates the effect of the scheduling policy and HACS-DTQSAW weight selection while ensuring a consistent basis for cross-algorithm comparison.

3.6 Performance Metrics and Statistical Analysis

Performance was assessed using nine complementary metrics: Average Waiting Time (AWT), Average Turnaround Time (ATT), Average Response Time (ART), Context Switches (CS), Jain's Fairness Index for Waiting Time (JFI-WT), Jain's Fairness Index for Turnaround Time (JFI-TT), Jain's Fairness Index for Response Time (JFI-RT), Priority Respect Ratio (PRR), and Starvation Index (SI). Lower values are preferred for AWT, ATT, ART, and CS, whereas higher values are preferred for the three JFI measures, PRR, and SI. JFI was calculated separately for waiting, turnaround, and response times to quantify the distributional fairness of each measure. PRR was calculated as the proportion of scheduling decisions that preserved the defined priority ordering, while SI was calculated as one minus the proportion of processes whose waiting time exceeded the overall AWT. These metrics jointly capture scheduling efficiency, responsiveness, overhead, fairness, priority compliance, and starvation resistance.

The Friedman test was used to identify overall differences among the evaluated scheduling configurations. The resulting statistical findings were combined with benchmark rankings to assess overall performance and cross-workload consistency.

4. EXPERIMENTAL RESULTS

This section presents the results of the large-scale benchmark involving 93 scheduling

configurations, comprising 29 established scheduling algorithms and 64 HACS-DTQSAW configurations, across eight workload scenarios: four synthetic workloads and four real-world datasets. The analysis examines overall benchmark performance, individual performance metrics, and cross-workload consistency.

4.1 Overall Benchmark Ranking

Algorithm	50 Workload	100 Workload	500 Workload	1000 Workload	Cloud Task Scheduling	Comprehensive Cloud Workload Performance	Google Cluster Traces v3	PC Dataset	Final Rank
HACS-DTQSAW (A0.05-B0.475-P0.475)	4	1	2	3	4	1	7	4	1
HACS-DTQSAW (A0-B0.5-P0.5)	1	2	1	1	2	2	2	18	2
HACS-DTQSAW (A0.15-B0.425-P0.425)	5	8	4	4	8	12	11	2	3
HACS-DTQSAW (A0.2-B0.4-P0.4)	3	18	7	2	3	22	19	3	4
HACS-DTQSAW (A0.175-B0.65-P0.175)	13	7	10	15	9	6	4	19	5
HACS-DTQSAW (A0.25-B0.5-P0.25)	11	6	11	10	17	17	15	5	6
HACS-DTQSAW (A0.225-B0.55-P0.225)	9	5	5	6	25	23	12	8	7
HACS-DTQSAW (A0.1-B0.45-P0.45)	2	3	3	7	34	19	27	1	8
HACS-DTQSAW (A0.2-B0.6-P0.2)	10	4	8	8	24	20	14	15	9
HACS-DTQSAW (A0.15-B0.7-P0.15)	21	9	9	18	14	5	5	23	10
HACS-DTQSAW (A0.275-B0.45-P0.275)	8	29	13	9	18	18	21	6	11
HACS-DTQSAW (A0.25-B0.375-P0.375)	6	26	6	5	23	29	24	35	12
HACS-DTQSAW (A0.3-B0.4-P0.3)	7	21	16	11	26	21	23	46	13
...	...	...	...	...	...	...	...	...	...
PBRR	78	77	74	75	64	74	38	75	71
FCFS	76	76	75	74	74	72	62	90	75
RR	81	82	84	86	84	82	86	80	87
MLFQ	84	85	82	82	88	70	70	93	82
SARBR	93	92	87	88	92	92	31	82	84
PTRR	91	93	93	93	89	93	92	81	93

Fig 1. Overall Benchmark Ranking of CPU Scheduling Algorithms.

HACS-DTQSAW (A0.05-B0.475-P0.475) achieved the first overall position, with workload ranks of 4, 1, 2, 3, 4, 1, 7, and 4 across the eight scenarios. Its consistently high placement demonstrates strong performance across both synthetic and real-world workloads. HACS-DTQSAW (A0-B0.5-P0.5) ranked second, with workload ranks of 1, 2, 1, 1, 2, 2, 2, and 18. Although its performance declined on the PC Dataset, its strong results across the remaining workloads secured the second overall position.

The comprehensive ranking reveals substantial differences in cross-workload performance among the evaluated scheduling configurations. The leading positions are predominantly occupied by HACS-DTQSAW configurations, indicating strong consistency across heterogeneous workloads rather than isolated superiority in a single scenario.

The third-ranked configuration, HACS-DTQSAW (A0.15-B0.425-P0.425), obtained ranks of 5, 8, 4, 4, 8, 12, 11, and 2. It was followed by A0.2-B0.4-P0.4, A0.175-B0.65-P0.175, A0.25-B0.5-P0.25, and A0.225-B0.55-P0.225, occupying positions four through seven. Thus, all seven leading positions were occupied by HACS-DTQSAW configurations, indicating strong cross-workload competitiveness of the multi-criteria scheduling approach.

Among the competing algorithms, ERRDTQ achieved the highest overall position at 16th,

with workload ranks of 36, 23, 25, 35, 22, 15, 6, and 31. Its strongest result was sixth position on Google Cluster Traces v3, although weaker performance on several synthetic workloads limited its overall ranking.

Among classical algorithms, Optimized SJF was the highest-ranked, occupying 36th overall, followed by SJF (54th), PRIORITY (60th), and FCFS (75th). These algorithms remained competitive in selected scenarios but showed less consistent performance across heterogeneous workloads.

Other adaptive and hybrid algorithms also exhibited considerable workload sensitivity. ORR-DTQ, ARR-STS, SJFDRR, PBRR, HybridSJF-DVQ, IRR-P, and SRBRR ranked 61st, 62nd, 69th, 71st, 73rd, 74th, and 76th, respectively. Although some achieved substantially better positions in individual datasets, their weaker performance elsewhere reduced their overall ranking.

The RR-based algorithms generally occupied the lower portion of the benchmark. P-RR, SATQRR, SRR, MLFQ, RR-MMDM, IRR-VTQ, RR, RR-HARM, Improved RR, and ORR ranked from 79th to 91st. The lowest positions were occupied by IRR Yadav (92nd) and PTRR (93rd).

These results demonstrate a clear distinction between workload-specific performance and cross-workload robustness. Algorithms that perform strongly in one workload may lose their advantage under different workload characteristics, whereas the leading HACS-DTQSAW configurations generally maintained high positions across multiple scenarios.

4.2 Individual Performance Metric Rankings

Fig.2 presents the rankings of the evaluated scheduling configurations across the nine performance metrics and eight workload scenarios. The metric-wise analysis demonstrates that algorithm performance varies substantially according to the optimization objective. No single algorithm consistently achieves the best position across all metrics and workloads.

Algorithm	AWT Rank	ATT Rank	ART Rank	CS Rank	FWT Rank	FTT Rank	FRT Rank	PRR Rank	SI Rank	Final Rank
Proposed(A0-B0.5-P0.5)	17	11	16	18	50	46	51	14	21	1
Proposed(A0.25-B0.5-P0.25)	16	10	14	22	53	53	54	41	11	2
Proposed(A0.175-B0.65-P0.175)	13	8	11	13	58	56	57	48	17	3
Proposed(A0.2-B0.4-P0.4)	29	17	29	28	41	42	47	23	25	4
Proposed(A0.15-B0.7-P0.15)	12	12	10	15	60	59	58	49	14	5
Proposed(A0.05-B0.475-P0.475)	20	49	21	12	49	57	52	16	18	6
Proposed(A0.225-B0.55-P0.225)	15	23	13	24	55	55	55	45	12	7
Proposed(A0.2-B0.6-P0.2)	14	14	12	23	57	58	56	47	16	8
Proposed(A0.3-B0.4-P0.3)	26	15	26	29	47	43	49	37	32	9
Proposed(A0.275-B0.45-P0.275)	18	21	17	25	51	51	53	43	27	10
...	...	...	...	...	...	...	...	...	...	...
RR	79	85	83	11	72	71	71	73	92	84
ORR	91	45	85	17	76	81	76	82	87	85
MARR	73	82	62	77	59	74	66	79	70	86
RR-HARM	78	84	28	86	78	68	75	71	77	87
OPRRP	84	88	86	84	64	67	90	12	75	88
SRR	85	89	73	7	71	77	79	91	81	89
Improved RR (IRR)	87	69	88	16	79	78	78	75	85	90
ADRR	80	87	76	34	70	90	74	77	83	91
IRR Yadav	92	92	81	20	81	87	82	76	91	92
PTRR	90	91	89	87	77	76	69	68	86	93

Rank Scale (Lower is Better)
1 Best — Mid Performance — Worst High Rank

Fig 2. Rankings across nine performance metrics**.**

4.2.1 Average Waiting Time (AWT)

Average Waiting Time represents the time processes spend waiting in the ready queue before receiving CPU service. Lower values indicate better performance.

The AWT results show a clear advantage for SJF and its closely related variants across the synthetic workloads. SJF achieves the lowest waiting times for 50, 100, 500, and 1000 processes, confirming the effectiveness of burst-time-oriented scheduling when waiting-time minimization is the primary objective. Optimized SJF also performs strongly and remains close to SJF across all four workload sizes.

Several adaptive and dynamic-quantum algorithms achieve intermediate performance. ARR-STS and ORR-DTQ are among the stronger alternatives, whereas conventional Round Robin and several variants generally produce higher waiting times. This difference becomes more pronounced as workload size increases, suggesting that repeated preemption and quantum-based execution can introduce additional waiting overhead under large workloads.

The real-world datasets show a less consistent ranking. The relative performance of SJF, Optimized SJF, ERRDTQ, ORR-DTQ, ARR-STS, and other algorithms varies across datasets, with some dynamic and hybrid approaches becoming more competitive than in the synthetic workloads. This indicates that algorithm effectiveness cannot be inferred from synthetic workloads alone.

HACS-DTQSAW configurations occupy different positions in the AWT rankings depending on the workload. Some approach the stronger competing algorithms, whereas others trade waiting-time performance for other scheduling objectives. Thus, the AWT results do not establish HACS-DTQSAW as the best approach for waiting-time minimization; rather, they show its performance relative to conventional, adaptive, hybrid, and dynamic approaches.

Overall, SJF remains the strongest algorithm when AWT is considered independently, particularly on the synthetic workloads. However, its AWT advantage does not imply overall superiority, as other metrics reveal different performance characteristics. This finding reinforces the need for a comprehensive multi-metric benchmark.

Table 4. Top 20 Algorithms Ranked by AWT.

Algorithm-AWT	Workload-50	Workload-100	Workload-500	Workload-1000	Cloud Task	Comprehensive Cloud	Google Cluster Traces v3	PC Dataset	AWT Rank
ERRDTQ	49052.144	95568.224	471131.1239	942084.4712	170914.7576	4175771.938	37557893.94	23237.29327	1
Optimized SJF	49052.144	95568.224	471131.1239	942084.4712	170914.7576	4175771.938	422103614.9	23237.29327	2
SJF	47555.051	94292.0165	470309.6229	941508.336	171101.8798	4304088.172	422031056.9	28022.34066	3
Proposed(A0-B1-P0)	72880.3077	143300.9568	722946.2303	1454025.319	239026.2109	5937140.415	785205631.4	25540.25706	4
Proposed(A0.025-B0.95-P0.025)	72944.19522	143433.3482	723551.9515	1455202.148	239089.8952	5940022.938	785229354.7	46920.45429	5
Proposed(A0.05-B0.9-P0.05)	73022.7759	143586.2143	724372.1859	1456709.498	239314.8185	5949250.855	785275878.1	49956.65421	6
ARR-STS	73589.19869	143780.4508	721384.4345	1452890.26	239347.4487	5920638.532	793001802.4	22082.95624	7
ORR-DTQ	69271.259	136898.284	680560.773	1362931.565	239527.443	5878008.969	807070652.1	21542.47734	8
Proposed(A0.075-B0.85-P0.075)	73131.49864	143859.3362	725683.6758	1459286.63	239746.1895	5966372.562	785345940.3	52097.5414	9

Proposed(A0.1-B0.8-P0.1)	73313.3111	144344.1711	727749.0381	1463436.256	240458.0063	5993386.55	785481766.7	55089.84608	10

4.2.2 Average Turnaround Time (ATT)

The ATT results show a pattern broadly similar to AWT because turnaround time is strongly influenced by the waiting experienced by processes. SJF achieves highly competitive results across the synthetic workloads, followed closely by Optimized SJF and several burst-oriented or dynamic-quantum algorithms.

As workload size increases, the differences between the algorithms become more pronounced. Algorithms that efficiently prioritize short processes maintain relatively favorable ATT values, whereas several Round-Robin-based methods experience larger increases in turnaround time. Nevertheless, the real-world datasets produce different relative rankings, with some adaptive and hybrid algorithms becoming more competitive.

The results therefore indicate that burst-time-oriented scheduling is particularly effective for minimizing turnaround time, but this advantage is not sufficient to establish universal dominance. Algorithms achieving strong ATT may simultaneously produce weaker fairness or priority-compliance results. The complete benchmark consequently provides a more balanced interpretation of algorithm quality.

Table 5. Top 20 Algorithms Ranked by ATT.

Algorithm-ATT	Mean-50	Mean-100	Mean-500	Mean-1000	Cloud Task	Comprehensive Cloud	Google Cluster Traces v3	PC Dataset	ATT Rank
SJF	51318.132	97992.7335	473978.6724	945175.9736	171182.1948	4306597.509	42225818 8.8	34222.57486	1
Proposed(A0.025-B0.95-P0.025)	76707.27622	147134.0652	727221.001	1458869.786	239170.2101	5942532.274	78545648 6.7	53120.6885	2
ARR-STS	77352.27969	147481.1678	725053.484	1456557.898	239427.7636	5923147.869	79322893 4.4	28283.19045	3
Proposed(A0.05-B0.9-P0.05)	76785.8569	147286.9313	728041.2354	1460377.135	239395.1335	5951760.192	78550301 0.1	56156.88842	4
ORR-DTQ	73034.34	140599.001	684229.8225	1366599.202	239607.7579	5880518.306	80729778 4.1	27742.71154	5
Proposed(A0.1-B0.8-P0.1)	77076.3921	148044.8881	731418.0876	1467103.894	240538.3213	5995895.886	78570889 8.7	61290.08028	6
Proposed(A0.125-B0.75-P0.125)	77393.74257	148629.067	734498.6525	1473257.09	241642.6319	6034975.342	78606541 6.9	62058.23738	7
Proposed(A0.175-B0.65-P0.175)	78447.37701	150634.5038	744971.8677	1493957.833	245776.7435	6163970.104	78763655 1.5	62987.93885	8
SJFDRR	87531.79761	168370.6858	822712.4465	1663963.257	286515.9441	6474840.015	79685937 2.6	83374.58792	9
Proposed(A0.25-B0.5-P0.25)	81917.6813	157394.1869	778507.229	1559906.321	261016.2304	6531195.56	79235246 9.1	63880.7758	10

4.2.3 Average Response Time (ART)

The ART comparison demonstrates greater diversity among the algorithms than the waiting- and turnaround-time results. Algorithms such as SRBRR, ARR-STS, ORR-DTQ, and several adaptive RR variants show competitive response-time behavior in different workloads, while SJF performs strongly in some scenarios but does not consistently dominate all datasets.

The results indicate that rapid initial CPU access depends not only on burst-time ordering but also on preemption and process-selection behavior. Consequently, some Round-Robin and hybrid approaches can obtain competitive response times even when their average waiting times are higher.

The real-world datasets further change the relative ranking of the algorithms. This confirms that response-time performance is highly workload-dependent. An algorithm that performs well under a controlled synthetic workload may lose its advantage when exposed to the more irregular characteristics of real-world traces.

The proposed HACS-DTQSAW configurations appear within the broader distribution of algorithms rather than occupying a universally dominant position. Their performance should therefore be interpreted in relation to the competing algorithms and the other evaluation metrics.

Table 6. Top 20 Algorithms Ranked by ART.

Algorithm-ART	Mean-50	Mean-100	Mean-500	Mean-1000	Cloud Task	Comprehensive Cloud	Google Cluster Traces v3	PC Dataset	ART Rank
SRBRR	23115.64116	46274.99459	228418.3734	453581.2261	111280.3622	2559787.882	183500669.8	16691.85668	1
HybridSJF_DVQ	26434.42387	50520.09433	242216.4025	485498.0555	122439.5974	2913481.887	182893514.3	16992.95098	2
Proposed(A0-B1-P0)	36127.2121	70747.62945	352001.2625	700897.2209	153355.9756	3664935.885	378316705.1	9017.516025	3
Proposed(A0.025-B0.95-P0.025)	36177.78162	70855.90549	352512.4544	701890.0988	153381.0166	3666112.293	378339786.5	29689.82658	4
Proposed(A0.05-B0.9-P0.05)	36221.77362	70943.64914	353004.6249	702782.3177	153468.3608	3670086.66	378385192.3	32201.48246	5
Proposed(A0.075-B0.85-P0.075)	36281.59772	71089.91364	353745.3139	704136.2014	153640.1325	3677516.932	378449407.7	34048.08903	6
Proposed(A0.1-B0.8-P0.1)	36374.10208	71349.96698	354873.9683	706250.5831	153929.2513	3689338.123	378528283.6	35766.09881	7
Proposed(A0.125-B0.75-P0.125)	36532.66691	71662.83439	356468.0837	709322.0603	154383.3968	3706447.684	378638037.8	36291.75896	8
ARR-STS	36872.30786	73105.97499	362141.3883	720895.9109	153422.7781	3728777.217	333317316.5	18350.47161	9
Proposed(A0.15-B0.7-P0.15)	36773.01396	72101.02031	358694.3139	713650.8565	155070.6823	3730164.952	378800296.3	36836.93086	10

4.2.4 Context Switches (CS)

Context switches provide an indication of scheduling overhead. Lower values generally indicate lower switching overhead.

The results show a clear advantage for non-preemptive algorithms such as FCFS, SJF, PRIORITY, and Optimized SJF, which generally produce fewer context switches. In contrast, Round Robin and several adaptive RR-based algorithms produce higher switching frequencies because their execution mechanisms rely more heavily on preemption and quantum expiration.

The hybrid and dynamic-quantum algorithms occupy intermediate positions depending on their quantum-selection mechanisms. The results therefore illustrate a fundamental trade-off: increasing preemption can improve responsiveness but generally increases scheduling overhead.

Across the larger workloads, this distinction becomes more important because the absolute number of context switches grows substantially. The comparison confirms that low context-switch overhead alone does not imply superior scheduling performance, since algorithms with fewer switches may sacrifice response time or fairness.

Table 7. Top 20 Algorithms Ranked by CS.

Algorithm-CS	Mean-50	Mean-100	Mean-500	Mean-1000	Cloud Task	Comprehensive Cloud	Google Cluster Traces v3	PC Dataset	CS Rank
FCFS	49	99	499	999	5999	4999	4886	181	1
Optimized SJF	49	99	499	999	5999	4999	4886	181	2
PRIORITY	49	99	499	999	5999	4999	4886	181	3
SJF	49	99	499	999	5999	4999	4886	181	4
ERRDTQ	50	100	500	1000	6000	5000	3.463211939	182	5
IRR	57.2	112.1	558.45	1110.6	6716	5002	4887	186	6
SRR	72.65	145.55	721.3	1439.1	10738	8719	8835	184	7
ORR-DTQ	87.85	174.65	869.75	1736.2	11984	9861	29121	185	8
SJFDRR	93.4	183.55	932.9	1874.95	11027	9554	11797	198	9
Priority-Based RR (P-RR)	90.4	184.7	934.25	1865.45	9007	7490	8332	509	10

4.2.5 Jain's Fairness Index for Waiting Time (JFI-WT)

The JFI-WT results reveal a substantially different ranking from the efficiency metrics. Algorithms that minimize waiting time are not necessarily those that distribute waiting time fairly.

Several conventional algorithms, particularly SJF, obtain relatively low fairness values despite their strong AWT performance. This reflects the unequal waiting-time distribution produced by prioritizing short jobs. In contrast, several adaptive, hybrid, and multi-criteria approaches achieve considerably higher fairness values.

The results demonstrate that fairness-oriented performance is distributed across different algorithmic families rather than being concentrated in one class. Some RR-based approaches provide improved fairness because CPU service is distributed more evenly, while certain adaptive algorithms show workload-dependent improvements.

The HACS-DTQSAW configurations also achieve relatively high JFI-WT values in many workloads, but the results show that some competing algorithms can achieve equal or higher fairness in individual scenarios. Therefore, the contribution of the proposed method should be interpreted through its overall balance across metrics rather than through an assertion of universal fairness superiority.

Table 8. Top 20 Algorithms Ranked by JFI-WT.

Algorithm- JFI-WT	Mean-50	Mean-100	Mean-500	Mean-1000	Cloud Task	Comprehensive Cloud	Google Cluster Traces v3	PC Dataset	FWT Rank
Proposed(A0.5-B0-P0.5)	0.747747967	0.735881906	0.72787293	0.73093138	0.854508687	0.8061264	0.909012329	0.332609894	1
Proposed(A1-B0-P0)	0.751834768	0.734577272	0.727486832	0.731942333	0.912957277	0.804415675	0.88629404	0.300992638	2
Proposed(A0-B0-P1)	0.746029434	0.734604272	0.727520054	0.729520671	0.844929617	0.807162473	0.933923575	0.353405454	3
Proposed(A0.95-B0.025-P0.025)	0.746415339	0.730482333	0.722539854	0.726646147	0.908685424	0.776751625	0.880940522	0.285257292	4
Proposed(A0.475-B0.05-P0.475)	0.73456614	0.720742758	0.712947571	0.715512766	0.844489846	0.787008769	0.891345311	0.289241022	5
Proposed(A0.9-B0.05-P0.05)	0.741536975	0.725042306	0.716509552	0.7203499	0.903314062	0.776141738	0.874774889	0.27173761	6
SATQRR	0.999700764	0.999932784	0.999995234	0.999994781	0.999930427	0.999999555	0.999999985	0.21124147	7
Proposed(A0.85-B0.075-P0.075)	0.73702666	0.71930742	0.709873852	0.713482418	0.896666631	0.776032277	0.868283779	0.262835176	8
Proposed(A0.025-B0.025-P0.95)	0.720832626	0.705318847	0.696521913	0.69829076	0.765875693	0.768168577	0.912697202	0.281462767	9
Proposed(A0.45-B0.1-P0.45)	0.718725883	0.70439011	0.695597914	0.698027586	0.83276501	0.783784335	0.870352425	0.276208479	10

4.2.6 Jain's Fairness Index for Turnaround Time (JFI-TT)

The JFI-TT results reinforce the distinction between efficiency and fairness. SJF, despite its strong turnaround-time efficiency, records very low fairness values for several synthetic workloads. This indicates that minimizing average turnaround time can result in considerable inequality among individual processes.

Several adaptive and RR-based approaches demonstrate more balanced turnaround-time distributions. SATQRR, in particular, achieves extremely high JFI-TT values across many workloads, indicating strong consistency in turnaround-time allocation. Other algorithms provide intermediate levels of fairness.

The proposed configurations also achieve substantially higher JFI-TT than many traditional algorithms in the synthetic workloads. Nevertheless, the complete results demonstrate that they are part of a broader group of algorithms exhibiting strong fairness characteristics rather than being universally dominant.

Table 9. Top 20 Algorithms Ranked by JFI-TT.

Algorithm- JFI-TT	Mean-50	Mean-100	Mean-500	Mean-1000	Cloud Task	Comprehensive Cloud	Google Cluster Traces v3	PC Dataset	FTT Rank
Proposed(A0.5-B0-P0.5)	0.736785039	0.729755565	0.726477994	0.730272197	0.854515315	0.806134778	0.908999384	0.272820641	1
Proposed(A0.95-B0.025-P0.025)	0.735000388	0.722527865	0.720772098	0.725727945	0.908683103	0.77661218	0.880750243	0.216708593	2
Proposed(A0.85-B0.075-P0.075)	0.721356539	0.710781 33	0.708022447	0.712591822	0.896666371	0.776048356	0.868097559	0.198131127	3
Proposed(A0-B0-P1)	0.734938 96	0.728874722	0.726125307	0.728862626	0.8449379	0.807170783	0.933908889	0.289736681	4
Proposed(A1-B0-P0)	0.741008986	0.728288177	0.726355858	0.731426717	0.912954229	0.804424577	0.886283438	0.246632421	5
Proposed(A0.475-B0.05-P0.475)	0.721187527	0.714296562	0.711077156	0.714626275	0.844498309	0.787022135	0.891333687	0.219180866	6
Proposed(A0.9-B0.05-P0.05)	0.727942732	0.717059644	0.714709253	0.719444822	0.903312668	0.776157795	0.874586576	0.204987638	7
SATQRR	0.999993 34	0.999992882	0.999992 62	0.999992563	0.999929393	0.99999947	0.999999977	0.210696281	8
Proposed(A0.8-B0.1-P0.1)	0.713126119	0.703338 74	0.700659753	0.705083508	0.888347262	0.776008421	0.860800257	0.192301 75	9
Proposed(A0.45-B0.1-P0.45)	0.705898951	0.697065674	0.693805489	0.697215935	0.832775666	0.783798231	0.870342991	0.209194676	10

4.2.7 Jain's Fairness Index for Response Time (JFI-RT)

The JFI-RT results show that algorithms with low response times do not necessarily provide an equitable distribution of response times. The conventional algorithms exhibit considerable variation, with several achieving low fairness values despite competitive response-time performance.

The proposed and adaptive algorithms generally provide higher JFI-RT values in many workloads, while SATQRR and several other fairness-oriented approaches also demonstrate strong results. The rankings vary considerably across the synthetic and real-world datasets, confirming the sensitivity of response-time fairness to workload characteristics.

These findings reinforce the importance of evaluating both the magnitude of response time and its distribution among processes. A scheduler that provides rapid service to some processes while significantly delaying others may achieve a favorable average while still exhibiting poor fairness.

Table 10. Top 20 Algorithms Ranked by JFI-RT**.**

Algorithm- JFI-RT	Mean-50	Mean-100	Mean-500	Mean-1000	Cloud Task	Comprehensive Cloud	Google Cluster Traces v3	PC Dataset	FRT Rank
Proposed(A0.5-B0-P0.5)	0.756667291	0.753695354	0.7508365	0.749450746	0.765456913	0.748771303	0.75928952	0.711914853	1
Proposed(A0-B0-P1)	0.756751061	0.755257898	0.7486346	0.747974118	0.761349808	0.747958377	0.770500603	0.752143435	2
Proposed(A1-B0-P0)	0.752877459	0.749953062	0.7479329	0.749296098	0.781710631	0.749249453	0.746198497	0.701543482	3
Proposed(A0.95-B0.025-P0.025)	0.74898871	0.746851895	0.7444312	0.745325933	0.780771527	0.734975376	0.742237354	0.701156866	4
Proposed(A0.9-B0.05-P0.05)	0.743516216	0.743380993	0.7405087	0.740862951	0.779502342	0.734732821	0.737509805	0.71349079	5
Proposed(A0.475-B0.05-P0.475)	0.745634692	0.743272292	0.740262	0.738343097	0.761752451	0.739770778	0.748313014	0.672498227	6
Proposed(A0.85-B0.075-P0.075)	0.738700098	0.739532051	0.7361722	0.73597959	0.777836438	0.734705872	0.732450748	0.711505534	7
Proposed(A0.8-B0.1-P0.1)	0.732606601	0.734672822	0.731257	0.730487814	0.775603805	0.734707677	0.726672662	0.710863922	8
Proposed(A0.75-B0.125-P0.125)	0.727025471	0.72928128	0.7256502	0.724452147	0.772647883	0.734989771	0.72022064	0.712984653	9
Proposed(A0.025-B0.025-P0.95)	0.731732024	0.731041 86	0.7241784	0.723965978	0.727895411	0.728615993	0.76018075	0.669028972	10

4.2.8 Priority Respect Ratio (PRR)

The Priority Respect Ratio (PRR) is calculated by measuring the proportion of scheduling decisions that preserve the intended priority ordering relative to the total number of scheduling decisions. For each context switch between two different processes, an indicator is assigned a value of 1 when the priority order is preserved, meaning that the previously executing process has a higher or equal priority than the newly selected process; otherwise, the indicator is assigned a value of 0. Here, a smaller numerical priority value represents a higher priority. The PRR is then obtained by dividing the total number of priority-respecting scheduling decisions by the total number of scheduling decisions, as defined in Equation (3.8). The resulting value ranges from 0 to 1, where a value of 1 indicates that all scheduling decisions respect the intended priority order, while lower values indicate a higher frequency of priority violations. As an original metric proposed in this study, PRR therefore provides a direct quantitative assessment of priority compliance during scheduling transitions, complementing conventional performance and fairness metrics.

PRR produces one of the clearest distinctions among scheduling philosophies. PRIORITY achieves the highest PRR across the evaluated workloads, as expected from its explicit priority-based decision mechanism. P-RR and PBRR also achieve very high priority-compliance values, while IRR-P performs strongly on several real-world datasets.

The conventional FCFS, SJF, RR, and MLFQ algorithms generally achieve moderate PRR values because priority is not their primary scheduling criterion. Their PRR values remain relatively stable but substantially below the explicitly priority-oriented methods.

The HACS-DTQSAW configurations show a wide range of PRR values. Configurations emphasizing priority achieve values approaching those of the priority-based algorithms, while configurations emphasizing other criteria produce lower PRR. This confirms that the ability to respect process priorities is strongly related to the scheduling decision policy.

The key finding from the complete comparison is therefore that PRIORITY is the strongest algorithm for PRR, but this advantage is specific to priority compliance and does not automatically translate into superior waiting time, response time, fairness, or starvation resistance.

Table 11. Top 20 Algorithms Ranked by PRR.

Algorithm- PRR	Mean-50	Mean-100	Mean-500	Mean-1000	Cloud Task	Comprehensive Cloud	Google Cluster Traces v3	PC Dataset	PRR Rank
PRIORITY	1	1	1	1	1	1	1	1	1
Priority-Based RR (P-RR)	0.972071702	0.985543541	0.99680979	0.998441807	0.999888975	0.999732977	0.999879981	0.997890295	2
Proposed(A0-B0-P1)	0.950992088	0.970526999	0.991872198	0.995516362	0.999002079	0.998796751	0.999829584	0.995215311	3
Proposed(A0.05-B0.05-P0.9)	0.950976628	0.970520037	0.991872198	0.995516362	0.999002079	0.998796751	0.856339468	0.995215311	4
Proposed(A0.025-B0.025-P0.95)	0.950976628	0.970520037	0.991872198	0.995516362	0.999002079	0.998796751	0.88070893	0.995215311	5
Proposed(A0.075-B0.075-P0.85)	0.950976628	0.970520037	0.991872198	0.995516362	0.999002079	0.998796751	0.843387866	0.995215311	6
Proposed(A0.1-B0.1-P0.8)	0.936842046	0.954730348	0.971974586	0.974095729	0.999002079	0.998796751	0.817655078	0.995215311	7
Proposed(A0.125-B0.125-P0.75)	0.895804676	0.904781959	0.911089107	0.908038793	0.999002079	0.998796751	0.797119973	0.990430622	8
Proposed(A0.15-B0.15-P0.7)	0.841108588	0.840375009	0.846543392	0.84297405	0.999002079	0.998796751	0.786213361	0.990430622	9
Proposed(A0.175-B0.175-P0.65)	0.784259685	0.792419924	0.795346986	0.790661641	0.999002079	0.9917778	0.755112474	0.980861244	10

4.2.9 Starvation Index (SI)

The Starvation Index (SI) is calculated by first identifying the processes whose waiting time exceeds the system's Average Waiting Time (AWT). The number of such processes is denoted by S, while n represents the total number of processes. The SI is then obtained by applying the proposed SI formulation, where the resulting value ranges from 0 to 1. A higher SI value indicates better resistance to starvation, as it means that a larger proportion of processes have waiting times at or below the system average. Conversely, a lower SI value indicates that a greater proportion of processes experience excessive waiting and are therefore more vulnerable to starvation. Thus, unlike conventional waiting-time metrics, SI directly evaluates the extent to which a scheduling algorithm prevents excessive waiting across processes and provides a complementary measure of scheduling fairness and starvation resistance.

The SI results demonstrate another major difference between scheduling strategies. SJF and Optimized SJF achieve relatively high SI values on the synthetic workloads, while several Round-Robin and adaptive approaches obtain lower values. However, the ranking changes considerably across the real-world datasets.

For example, SJF achieves approximately 0.63 SI across the synthetic workloads and approximately 0.79 on the PC Dataset, whereas SATQRR performs poorly on several synthetic and cloud workloads but achieves a much higher value on Google Cluster Traces v3. SJFDRR also demonstrates strong SI on the Cloud Task, Comprehensive Cloud, and PC datasets. These results clearly demonstrate that starvation resistance is highly dependent on workload characteristics.

The proposed configurations generally produce moderate SI values, with differences among configurations being smaller than those

observed for some other metrics. Consequently, HACS-DTQSAW does not dominate starvation resistance. Instead, its role in the overall benchmark is to provide a balance between starvation prevention and the other scheduling objectives.

Table 12. Top 20 Algorithms Ranked by SI**.**

Algorithm- SI	Mean-50	Mean-100	Mean-500	Mean-1000	Cloud Task	Comprehensive Cloud	Google Cluster Traces v3	PC Dataset	SI Rank
Optimized SJF	0.631	0.635	0.634	0.63215	0.568166667	0.5756	0.531000614	0.818681319	1
ERRDTQ	0.631	0.635	0.634	0.63215	0.568166667	0.5756	0.418459244	0.818681319	2
SJF	0.629	0.634	0.6337	0.632	0.568333333	0.5756	0.530795989	0.785714286	3
Proposed(A0.475-B0.475-P0.05)	0.525	0.5345	0.5364	0.53235	0.482333333	0.4746	0.2946593	0.862637363	4
Proposed(A0.5-B0.5-P0)	0.524	0.5355	0.5363	0.5323	0.483333333	0.4764	0.2946593	0.862637363	5
Proposed(A0.45-B0.45-P0.1)	0.526	0.534	0.5365	0.53195	0.475666667	0.4772	0.2946593	0.857142857	6
ORR-DTQ	0.608	0.6245	0.6322	0.6307	0.495666667	0.515	0.281767956	0.802197802	7
SJFDRR	0.535	0.57	0.5434	0.51405	0.6065	0.6046	0.282177205	0.884615385	8
Proposed(A0.425-B0.425-P0.15)	0.525	0.534	0.5354	0.53125	0.469	0.4778	0.294454676	0.818681319	9
Proposed(A0-B1-P0)	0.519	0.525	0.5227	0.5207	0.498666667	0.5026	0.294863925	0.818681319	10

4.3 Overall Interpretation of Benchmark Results

The combined results demonstrate that CPU scheduling algorithms exhibit distinct performance profiles rather than universal dominance. SJF and Optimized SJF are particularly effective for minimizing waiting and turnaround times while maintaining low context-switch overhead. PRIORITY provides the strongest priority compliance, whereas several RR-based, adaptive, and hybrid algorithms demonstrate competitive fairness or response-time performance.

The comparison between synthetic and real-world workloads is particularly important. Algorithm rankings frequently change across the Cloud Task Scheduling, Comprehensive Cloud Workload Performance, Google Cluster Traces v3, and PC datasets. Similarly, increasing the synthetic workload size from 50 to 1000 processes changes the relative performance of several algorithms. These findings demonstrate that workload characteristics and scalability substantially influence scheduling effectiveness.

The metric-wise results also reveal clear trade-offs. The algorithm with the lowest AWT is not necessarily the fairest; the algorithm with the highest PRR is not necessarily the most responsive; and the algorithm with the fewest context switches may not provide the most equitable service distribution. Therefore, single-metric evaluation provides only a partial assessment of scheduler performance.

The comprehensive benchmark ranking addresses this limitation by considering performance across all eight workloads and the complete set of evaluation objectives. The leading HACS-DTQSAW configurations demonstrate that strong overall performance can be achieved through balanced multi-criteria decision-making rather than optimization of a single metric. Importantly, however, their results should be interpreted comparatively rather than as evidence of universal superiority. Overall, the benchmark establishes a clear hierarchy: HACS-DTQSAW configurations occupy the leading overall positions, ERRDTQ

is the strongest non-proposed competitor, and Optimized SJF is the highest-ranked classical algorithm. The results support the value of large-scale evaluation across both synthetic and real-world workloads for identifying scheduling approaches that provide consistent performance rather than isolated advantages.

5. STATISTICAL VALIDATION

To assess whether the observed performance differences among the 93 scheduling configurations were statistically significant, non-parametric statistical tests were applied.

5.1 Friedman test

The Friedman test was conducted independently for each performance metric and synthetic workload size.

Table 12. Friedman Test Results Across Performance Metrics and Workload Sizes.

Metric	Workload 50		Workload 100		Workload 500		Workload 1000	
	Friedman $\chi^2(92)$	p-value	Friedman $\chi^2(92)$	p-value	Friedman $\chi^2(92)$	p-value	Friedman $\chi^2(92)$	p-value
AWT	1683.041773	1.2766E−290	1683.827908	8.7999E−291	1696.293788	2.4072E-293	1705.618461	2.9087E-295
ATT	1683.041773	1.2766E−290	1683.827908	8.7999E−291	1696.293788	2.4072E-293	1705.618461	2.9087E-295
ART	1639.632274	1.0527E−281	1663.478034	1.3368E−286	1654.182369	1.0845E-284	1670.711614	4.3652E-288
CS	1200.72729	1.7782E−192	1252.981747	5.4217E−203	1284.035888	2.9408E-209	1384.578778	1.2788E-229
FWT	1670.619536	4.5596E−288	1754.553737	2.4529E−305	1781.521338	6.7839E-311	1786.111575	7.6728E-312
FTT	1737.777118	7.0005E−302	1754.232997	2.8560E−305	1776.131019	8.7664E-310	1790.879117	0
FRT	1785.358336	1.0972E−311	1786.954848	< 0.001	1699.150655	6.223E-294	1801.856906	0
PRR	1151.103803	1.5932E−182	1265.138904	1.9170E−205	1404.135083	1.3611E-233	1630.87083	6.6106E-280
SI	750.919969	5.9241E−104	721.767487	2.14558E−98	936.1901659	6.9167E-140	1046.09546	1.3765E-161

All Friedman tests were statistically significant at $\alpha = 0.05$, with $p < 0.001$ for every metric and workload size. This indicates that the evaluated scheduling configurations differ significantly in their performance and that these differences persist across increasing synthetic workload sizes.

5.2 Nemenyi Post-Hoc Analysis

Following the significant Friedman tests, the Nemenyi post-hoc procedure was applied to identify the pairwise differences among the evaluated scheduling configurations. Because the benchmark comprises 93 configurations, reporting all 4,278 pairwise comparisons for each metric would substantially increase the size of the manuscript. Therefore, the main text reports the statistically and practically relevant comparisons involving the leading algorithms and HACS-DTQSAW configurations, while the complete pairwise results are retained in the supplementary statistical material.

For the 50-process workload, the Nemenyi analysis revealed objective-dependent differences among the scheduling approaches. For average waiting time (AWT) and average turnaround time (ATT), SJF achieved the best average rank (1.55), followed by ERRDTQ and Optimized SJF (2.225 each). However, the leading HACS-DTQSAW configuration, A0-B1-P0, was not significantly different from SJF, ERRDTQ, or Optimized SJF (adjusted $p = 1.000$), indicating statistically comparable efficiency performance among these configurations.

For average response time (ART), SRBRR obtained the best average rank (1.20), whereas A0-B1-P0 ranked second (5.45). The difference

between A0-B1-P0 and SRBRR was not statistically significant (adjusted p = 1.000), indicating that the proposed configuration achieved response-time performance statistically comparable with the best-performing benchmark for this workload.

A stronger distinction was observed for context-switch overhead (CS). The HACS-DTQSAW configuration A0.05-B0.475-P0.475 showed significant differences from FCFS, SJF, Priority Scheduling, and Optimized SJF (adjusted p = 0.04285 for each comparison). It also differed significantly from several dynamic and hybrid approaches, including PBRR ($p = 1.32 \times 10^{-6}$), SATQRR ($p = 2.78 \times 10^{-6}$), OPRRP ($p = 5.77 \times 10^{-6}$), and HybridSJF-DVQ ($p = 1.51 \times 10^{-5}$).

For fairness, SATQRR achieved the best average rank for both FWT and FTT (1.00). Nevertheless, A1-B0-P0 was not significantly different from SATQRR (p = 1.000), while it differed significantly from SJF ($p < 0.001$). For response-time fairness (FRT), A0.5-B0-P0.5 achieved the best average rank among the evaluated configurations and was significantly different from SJF ($p < 0.001$) and ERRDTQ ($p = 9.11 \times 10^{-10}$), while remaining statistically comparable with several other leading HACS-DTQSAW configurations.

For priority compliance, Priority Scheduling achieved the best average rank (1.05). The priority-dominant HACS-DTQSAW configuration A0-B0-P1 was not significantly different from Priority Scheduling or PBRR (p = 1.000), demonstrating comparable priority-respect performance. For starvation resistance, A0-B0.5-P0.5 was statistically comparable with ERRDTQ (p = 0.842), SJF (p = 0.855), and Optimized SJF (p = 0.842), while significant differences were observed against several other RR-based approaches.

Overall, the Nemenyi analysis for the 50-process workload indicates that the proposed HACS-DTQSAW model does not exhibit universal statistical superiority across all metrics. Instead, different weight configurations achieve statistically competitive performance for different scheduling objectives. This objective-dependent behavior is consistent with the multi-criteria design of the proposed scheduler and provides statistical evidence that its performance is influenced by the relative weighting of arrival time, burst time, and priority.

The results provide evidence that the overall performance distributions differ significantly among the evaluated configurations. This pattern suggests that the strong performance of HACS-DTQSAW is not limited to a single weight combination but extends across a region of its configuration space. Overall, the statistical analysis confirms that the performance differences observed in the benchmark are statistically significant rather than attributable solely to random variation.

6. DISCUSSION

The benchmark demonstrates that CPU scheduling performance is inherently multi-dimensional and strongly dependent on workload characteristics. Across the evaluated classical, adaptive, hybrid, and multi-criteria approaches, no scheduling strategy consistently dominated all performance dimensions. The results therefore indicate that scheduler effectiveness cannot be adequately characterized by a single metric or workload.

The efficiency results reveal clear trade-offs among scheduling objectives. Algorithms that prioritize short processes can achieve low waiting and turnaround times, whereas preemptive and adaptive approaches may improve responsiveness at the cost of additional context-switch overhead. Conversely, algorithms with lower scheduling overhead may provide less responsive service. These trade-offs become more pronounced as workload size increases, demonstrating that efficiency improvements in one dimension do not necessarily translate into superior overall performance.

The starvation results further demonstrate the importance of workload-dependent analysis. SJF and Optimized SJF achieved relatively high SI values among conventional algorithms on the

synthetic workloads, with values around 0.63, but their relative performance changed across real-world datasets. Similarly, SATQRR showed substantially different SI behavior across workload types, while SJFDRR performed strongly on the Cloud Task, Comprehensive Cloud, and PC datasets but less strongly on Google Cluster Traces v3. These variations indicate that starvation resistance depends on the interaction between scheduling policy and workload structure rather than on algorithm category alone. Accordingly, SI provides an important complementary perspective to conventional efficiency measures.

The adaptive and hybrid results also show that adaptation does not guarantee consistent superiority. Algorithms that perform well under one workload may lose their advantage when arrival patterns, burst characteristics, or workload scale change. This finding is particularly relevant for dynamic-quantum approaches, whose effectiveness depends on how their adaptation mechanism responds to the underlying workload. Hybrid algorithms similarly benefit from combining complementary scheduling mechanisms, but the resulting trade-offs may favor one performance objective over another.

The HACS-DTQSAW configurations provide a further illustration of this trade-off. Their performance varies according to the relative weights assigned to Arrival Time, Remaining Burst Time, and Priority, confirming that criterion preferences influence scheduling behavior. Nevertheless, the leading configurations in the overall benchmark demonstrate that strong performance is achievable across heterogeneous workloads without requiring a single universally optimal weighting scheme. The concentration of several HACS-DTQSAW configurations among the leading overall positions therefore suggests a relatively robust region of the configuration space rather than an isolated optimum.

The comparison between synthetic and real-world workloads is particularly important. Synthetic workloads provide controlled scalability analysis from 50 to 1000 processes, whereas the real-world datasets introduce workload characteristics that are not fully captured by controlled generation. The changes in algorithm rankings across these datasets demonstrate that results obtained from a single synthetic workload may not generalize to practical computing environments. The use of heterogeneous workloads within a unified simulation framework therefore strengthens the external relevance of the benchmark.

Taken together, the findings show that scheduling algorithms should be evaluated according to the objectives and workload conditions under consideration rather than by algorithm category alone. The observed trade-offs among efficiency, responsiveness, scheduling overhead, fairness, priority compliance, and starvation resistance explain why single-metric comparisons can produce incomplete or misleading conclusions. The Friedman test further strengthen this interpretation by providing statistical evidence that the observed differences are systematic rather than attributable solely to experimental variation.

Overall, the benchmark provides evidence that no universally optimal CPU scheduling algorithm exists. The appropriate scheduling strategy depends on workload characteristics and the relative importance assigned to competing performance objectives. Consequently, large-scale evaluation across heterogeneous workloads, multiple performance dimensions, and statistical validation provides a more reliable basis for assessing scheduling algorithms than isolated comparisons based on a single metric or dataset.

7. CONCLUSION

This study presented a large-scale benchmark of CPU scheduling algorithms under heterogeneous workload conditions. A total of 29 established scheduling algorithms and 64 HACS-DTQSAW configurations were evaluated within a unified event-driven

simulation framework using four synthetic workloads and four real-world datasets. Nine complementary performance metrics were considered, covering efficiency, responsiveness, scheduling overhead, fairness, priority compliance, and starvation resistance. Statistical validation was performed using the Friedman test.

The results confirm that CPU scheduling is inherently a multi-objective problem and that no single algorithm consistently provides the best performance across all objectives and workloads. SJF and Optimized SJF performed particularly well in waiting time, turnaround time, and context-switch overhead, whereas PRIORITY achieved the strongest priority compliance. Other adaptive and hybrid algorithms demonstrated competitive performance for specific metrics or workload conditions. These findings highlight the limitations of single-metric evaluation and demonstrate that improvements in one performance dimension do not necessarily imply overall superiority.

Workload characteristics also had a substantial effect on scheduling performance. Algorithm rankings changed across synthetic and real-world workloads, and increasing the workload size from 50 to 1000 processes altered the relative performance of several algorithms. This demonstrates that workload diversity and scalability are essential for reliable evaluation and that results obtained from a single workload may not generalize to heterogeneous computing environments.

Within the overall benchmark, HACS-DTQSAW (A0.05-B0.475-P0.475) achieved the highest overall position, while seven HACS-DTQSAW configurations occupied the first seven positions. This concentration of leading configurations indicates that the observed performance is not dependent on a single weight combination but extends across a relatively robust region of the configuration space.

Overall, the findings support a shift from single-objective evaluation toward comprehensive multi-objective benchmarking in CPU scheduling research. HACS-DTQSAW does not achieve universal superiority in every individual metric; rather, its main strength lies in maintaining strong overall performance across heterogeneous workloads through the combination of multi-criteria process selection and Dynamic Time Quantum. The principal contribution of this study is therefore the empirical demonstration that reliable CPU scheduling evaluation requires simultaneous consideration of multiple objectives, workload diversity, scalability, and cross-workload consistency. At the same time, the observed workload-dependent behavior confirms that no universally optimal scheduling policy exists; the appropriate choice ultimately depends on workload characteristics and the relative importance of efficiency, responsiveness, overhead, fairness, priority compliance, and starvation resistance.

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, 7*(4), 238–241.

[4] Benghuzi, K., & Alhenshiri, A. (2026). A hybrid adaptive CPU scheduling algorithm using a dynamic time quantum and simple additive weighting. *Alqalam Journal of Science, 2*(6), 609–620. doi:10.69667/ajs.26608

[5] Dash, A. R., & Samantra, S. K. (2016). An optimized round robin CPU scheduling algorithm with dynamic time quantum. *arXiv preprint* arXiv:1605.00362.

[6] Dhamdhere, D. M. (2009). *Operating systems: A concept-based approach* (1st ed.). McGraw-Hill Education.

[7] 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.

[8] 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*.

[9] 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.

[10] Hiranwal, S., & Saroj, D. K. (2011). Adaptive round robin scheduling using shortest burst approach based on smart time slice. *Unpublished manuscript*.

[11] 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.

[12] Jain, R. K., Chiu, D. M. W., & Hawe, W. R. (1984). *A quantitative measure of fairness and discrimination for resource allocation in shared computer systems* (DEC Research Report TR-301). Digital Equipment Corporation, Eastern Research Laboratory.

[13] Kaliszewski, I., & Podkopaev, D. (2016). Simple additive weighting—A metamodel for multiple criteria decision analysis methods. *Expert Systems with Applications, 54*, 155–161.

[14] 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), 1831–1837.

[15] Mishra, M. K. (2012). An improved round robin CPU scheduling algorithm. *Journal of Global Research in Computer Science, 3*(6), 64–69.

[16] 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), 1–11.

[17] Mody, S., & Mirkar, S. (2019). Smart round robin CPU scheduling algorithm for operating systems. In *Proceedings of the 2019 4th International Conference on Electrical, Electronics, Communication, Computer Technologies and Optimization Techniques (ICEECCOT)* (pp. 309–316). IEEE.

[18] 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 and Technology (ISCET)* (pp. 126–137).

[19] Panda, S. K., & Bhoi, S. K. (2014). An effective round robin algorithm using min-max dispersion measure. *arXiv preprint* arXiv:1404.5869.

[20] 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.

[21] 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.

[22] 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 time-shared systems. *International Arab Journal of Information Technology, 17*(1), 90–98.

[23] 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.

[24] 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.

[25] Silberschatz, A., Galvin, P. B., & Gagne, G. (2013). *Operating system concepts* (9th ed.). John Wiley & Sons.

[26] 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.

[27] Stallings, W. (2011). *Operating systems: Internals and design principles* (7th ed.). Prentice Hall.

[28] Tanenbaum, A. S., & Bos, H. (2015). *Modern operating systems* (4th ed.). Pearson.

[29] Vafaei, N., Ribeiro, R. A., & Camarinha-Matos, L. M. (2022). Assessing normalization techniques for simple additive weighting method. *Procedia Computer Science, 199*, 1229–1236. doi:10.1016/j.procs.2022.01.156

[30] 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.

[31] 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.

[32] Zouaoui, S., Boussaid, L., & Mtibaa, A. (2019). Priority based round robin (PBRR) CPU scheduling algorithm. *International Journal of Electrical and Computer Engineering, 9*(1), 1–8.

[33] Datasets

[34] Ajithdari. (n.d.). *Comprehensive cloud workload performance* [Dataset]. Kaggle.

[35] Derrickmwiti. (n.d.). *Google 2019 cluster sample* [Dataset]. Kaggle.

[36] Programmer3. (n.d.). *Cloud task scheduling dataset* [Dataset]. Kaggle.