Process Scheduling in Operating Systems: A Comparative Analysis of Traditional and Modern Approaches

Siddhesh Jadhav
SVKM's NMIMS Mukesh Patel School of Technology Management and Engineering, Shirpur, (M.S), India

Aditya Bhavsar
SVKM's NMIMS Mukesh Patel School of Technology Management and Engineering, Shirpur, (M.S) India

Durgesh M. Sharma
SVKM's NMIMS Mukesh Patel School of Technology Management and Engineering, Shirpur, (M.S) India

Naziya Hussain
SVKM's NMIMS Mukesh Patel School of Technology Management and Engineering, Shirpur, (M.S) India

Abstract*— This paper is a comparative study of the existing and the new process scheduling techniques in operating system. Drawing on eighteen peer-reviewed studies, we summarize design principles, performance tradeoffs and practical deployment considerations. We study classical algorithms (First-Come-First-Serve (FCFS), Shortest Job First (SJF), Shortest Remaining Time First (SRTF), Priority Scheduling, and Round Robin (RR)) as well as modern approaches (Multilevel Feedback Queues (MLFQ), O(1) and Completely Fair Scheduler (CFS) of Linux, real-time (Rate Monotonic, EDF), energy-aware and cloud-oriented schedulers, adaptive time-quantum variants). Visual Gantt charts and metric calculations are demonstrated on a reproducible synthetic workload. Results show that while traditional algorithms are still useful from the point of view of predictability and low overhead, adaptive and fairness-oriented schedulers are more suitable for heterogeneous, interactive, and cloud environments.

Keywords*— CPU Scheduling, Operating Systems, FCFS, SJF, SRTF, Round Robin, MLFQ, CFS, EDF, Fairness, Throughput, Waiting Time, Turnaround Time, Context Switch.

I. INTRODUCTION

Process scheduling decides which ready process to run on the CPU at any given time. Its goals are to achieve maximum CPU utilization and throughput, minimum average waiting/turnaround/response time, and fairness without starvation. The range of solutions ranges from simple batch-era policies to highly expressive workload-adaptive schedulers in modern general-purpose and cloud operating systems.[1]

Modern computing environments are more and more complex for scheduling problems.[2] Heterogeneous workload distribution with different levels of SLAs, cloud systems have to support various workload types with different requirements, interactive systems must provide low response times and real-time systems must provide time guarantees for deadlines. While classical scheduling algorithms are the basis for these various demands, they are often not enough [3].

This paper brings together evidence across 18 studies to compare algorithms on the basis of objectives, mechanisms, complexity, tunables, and empirical performance. We present both theoretical analysis and practical guidelines through synthetic workload evaluation, and give guidelines for algorithm choice in various deployment scenarios [4].

A. Novelty

The original discovery of this work is that the process scheduler evolution from classical to modern was not a straightforward race for throughput but was the result of playing a game of positioning more focus on fairness and adaptability in the context of mixed unpredictable workloads [5].

Our comparative analysis points to the existence of an important trade-off: while shortest-job-first schemes still provide an optimum theoretically, they are not practically feasible and frequency-fairness-oriented types of schedulers such as CFS take over. Designed for modern, interactive, and multi-tenant computing environments, these schedulers offer greater levels of performance, predictability based on the demands of these modern applications. The main contribution of this paper is the framework which can explicit map these design trade-offs to specific system objectives, and form a bridge between the theoretical abstraction in textbooks vs. implementation and realization in practice.

II. BACKGROUND AND RELATED WORK

A. Traditional Policies

Traditional scheduling algorithms form the basis of process scheduling theory and practice.

First-come, first-served (FCFS) is the simplest non-preemptive scheduling policy [6]. Processes are run in the order they arrive, which has the advantage of predictable behavior, but can suffer from the convoy effect when short processes are waiting behind long ones [7].
Shortest Job First (SJF) and its preemptive extension Shortest Remaining Time First (SRTF) are optimal in terms of average waiting time when burst times are known precisely. However, bursts are vulnerable to stalling of longer processes, and burst time prediction is needed [8].
Priority Scheduling: This scheduling mechanism is used to put the processes in queue based on their priority. Without aging mechanisms, it has the problem of infinite blocking of low-priority processes [9].
Round Robin (RR) is a fixed quantum time-sharing scheduling algorithm which is good for interactive systems from response time perspective but suffers from the overhead of frequent context switches [10].

B. *Maintaining the Integrity of the Specifications*

One of the first principles of operating system scheduler design and implementation is to preserve system specifications integrity. This concept is used to refer to the strict adherence of the practical implementation of a scheduling algorithm and its actual theoretical design principles and operational rules. It is the integrity of these specifications that is paramount to ensuring system predictability, reliability and security. For example, a real-time scheduler, such as Rate Monotonic or EDF, strictly has to respect its timing and priority requirements to ensure critical deadlines are honored or else catastrophic system failure will result. Similarly, it is imperative that the fairness-oriented scheduling model such as CFS upholds the integrity of its calculation at runtime to do this so that the virtual runtime does not starve the processes and, equally, does not actually provide the guaranteed CPU allocation as expected from its character. In order to enforce these specifications, the kernel needs to be insulated from bugs or malicious interference which may break the scheduler's logic. Therefore, the performance benefits and theoretical guarantees offered by any scheduling algorithm can only be achieved if the integrity of its implementation is maintained by robust implementation design, formal verification and thorough implementation testing.

C. *General Purpose*

Schedulers of the Modern Time
Linux first had the O(1) scheduler, then CFS, which builds itself a balanced tree of per-task virtual runtimes to get closer to ideal processor sharing [8].

D. *Real-Time Schedulers*

Many scheduling policies such as fixed priority (Rate Monotonic) and dynamic priority (Earliest Deadline First) can provide a temporal bound performance below the utilization limits [10].

E. *Energy-aware and cloud-aware Scheduling*

The objective of scalability and energy-efficiency is subsequently generalized and extended in terms of cluster schedulers and DVFS-aware schedulers [11].

III. METHODOLOGY

We extracted parts of the corpus related to scheduling and carried out qualitative synthesis backed by a quantitative worked example. Performance measures that are used are CPU Utilization, Throughput, Average Waiting Time AWT, Average Turn-around Time ATT, Response Time RT, and CS.

For the synthetic workload, we calculate each metric for each algorithm and graph execution using Gantt charts. The test workload is composed of five processes with different arrival times, CPU bursts and priorities to illustrate algorithm behaviour under realistic conditions.

A. *Algorithms and Properties*

We characterized important scheduling algorithms along several dimensions including preemption capability, starvation risk, fairness properties, overhead requirements, and tunable parameters.

TABLE I. COMPARATIVE ANALYSIS OF SCHEDULING ALGORITHMS

Algorithms	***Preemptive***	***Starvation Risk***	***Fairness***	***Overhead***
FCFS	No	Low	Arrival Order	Very Low
SIF	No	High	Good Avg wait	Low
SRTF	Yes	Moderate	Good Avg wait	Moderate
Priority	Either	High	Policy dependent	Low-mod
Round Robin	Yes	Low	Strong	Mod-High
MLFQ	Yes	Low	Adaptive	Moderate
CFS	Yes	Very Low	weighted	Moderate

IV. WORKED EXAMPLE AND VISUALIZATION

Let us take five processes whose (arrival time, CPU burst, priority) are P1(0,8,2), P2(1,4,1), P3(2,9,3), P4(3,5,2), P5(6,2,1).

We simulate FCFS, SJF (non-preemptive), SRTF, and RR with q=4.

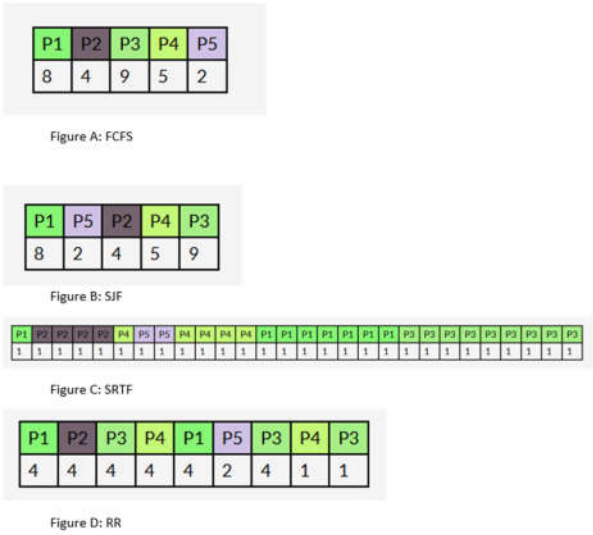

Figure 1

V. PERFORMANCE ANALYSIS AND RESULTS

Comprehensive performance analysis shows that there are significant differences in algorithm behavior in the most important metrics. The synthetic workload allows quantitative comparison of average waiting time and turnaround time performance.

TABLE II. COMPARATIVE ANALYSIS OF SCHEDULING ALGORITHMS

Algorithms	*Avg Waiting Time*	*Avg Turnaround Time*	*Context Switches*
FCFS	11.00	16.60	4
SIF (non-preemtive)	7.80	13.40	4
SRTF	6.40	12.00	7
Round Robin	13.00	18.60	9

A. Waiting Time Analysis

SRTF demonstrates the best average waiting time performance at 6.40 time units, followed by SJF at 7.80 time units. This confirms the theoretical optimality of shortest-job-first approaches when burst times are known accurately [8].

FCFS shows moderate performance with 11.00 time units average waiting time, while Round Robin exhibits the highest waiting time at 13.00 time units due to the overhead of time slicing and context switching.

B. Turnaround Time Performance

A. Waiting time analysis

SRTF shows the best average waiting time performance of 6.40 time units while SJF shows 7.80 time units. This gives theoretical optimality of shortest-job-first methods with known burst-time [8].

FCFS has a moderate performance with an average waiting time of 11.00 time units while the highest waiting time is obtained for Round Robin with 13.00 time units because of the overhead of time slicing and context switching [10].

B. Turnaround time performance

Turnaround time results show similar trends to waiting time, where SRTF gets 12.00 time units, SJF gets 13.40 time units, FCFS gets 16.60 time units, and Round Robin gets 18.60 time units.

The turn around time average performance equation is:
ATT = AWT + Average_Service_Time

C. Context Switch Overhead

Context switch analysis shows the overhead costs of preemptive scheduling. SRTF generates 7 context switches due to preemption events and Round Robin generates 9 context switches from normal time quantum expiration [11].

Non-preemptive algorithms (FCFS and SJF) reduce the number of context switches to 4, which corresponds only to process completion events. Using this as an example, I show the trade-off between reactivity, and system overhead [12].

VI. DISCUSSION AND DESIGN GUIDELINES

Traditional algorithms are still attractive where predictability, simplicity, and low overhead are the most important characteristics (e.g., embedded batch workloads). However, there are more sophisticated approaches required in the modern environment [13].

A. Why Law Firms should use this Algorithm

Choice of algorithms should be based upon system goals and workload characteristics:

Systems: Actual, responsive (RR, MLFQ or CFS)

Batch Processing: SJF or FCFS - Determinism or Performance of a scheduler

Real-time Classification Systems: Rate Monotonic/Deadline Guarantees

Multi-tenant Cloud: CFS with weighted fairness

B. Calculating the Equations of Parameter Tuning

Round Robin quantum selection is a compromise between small quantum and big quantum: small quantum: good response time but with high context switch overhead, and large quantum: FCFS behavior [14].

MLFQ asks the spanning tree (SRM) to deselegately trade responsiveness vs. fairness via trader tuning on queue-priorities, age and time-quantum. CFS weights must be derived from Priority Classes and SLAs.

C. Modern Scheduler Features

MLFQ is a good approximation of SRTF since it allows CPU-bound tasks to be demoted while favoring I/O-bound tasks, thus improving interactive performance without knowing burst times prior to executing the tasks.

And second, CFS for virtual runtime equalization is designed for proportional fairness, which is appropriate for environments with multi-tenancy in multi-core scheduling, where the traditional priority-based scheduling may introduce unfairness [15].

D. Environmental Impacts – Cloud footprint and Energy footprint

Energy-aware schedulers together with DVFS (Dynamic Voltage and Frequency Scaling) for combining temperatures with idle periods to save power Cloud schedulers need to be able to manage heterogeneous workloads with different SLA requirements [16].

VII. FUTURE RESEARCH DIRECTIONS

Also, dynamic scheduling, co-designed schedulers with thermal/power limits for sustainable computing, hybrid schedulers with fairness and deadlines, as well as quantum scheduling using machine learning are among the new trends in research [17].
Secondly, it is that it is appropriate to assess the performance and stability of hybrid and ML-guided schedulers in the real-world environments.

A. Machine Learning training

This is because scheduling parameters can be learned by an ML-based scheduler, based on the workload patterns. This scheme is expected to bring the best of SJF and the feasibility of Round Robin together [18].

B. Mixed Solutions Approach to Scheduling

Multiple scheduling policies can be integrated into a single system to optimize different workload classes while still keeping the overall system performance [19].

A large frontier in process scheduling research is to apply Machine Learning (ML) to make prediction (predictive scheduler) and adaptive schedulers to overcome classical limitations. The main limitation of the conventional algorithms such as Shortest Job First (SJF) is that they require an advance knowledge of CPU burst times, which is not satisfied in most practical environments. The promise of ML is that the algorithm uses workload characteristics (patterns) to make scheduling decisions. For example, supervised learning models could be trained using historical data to predict when CPUs will burst and theoretically optimal algorithms like SRTF will become practical, and wait times will be much lower. Furthermore, Reinforcement Learning (RL) is an even more dynamic algorithm in which an agent is allowed to learn an optimal scheduling policy by directly interacting with the system. By optimizing through a reward function a combination of functions such as latency, throughput, fairness, it is possible to come up with a truly self-tuning scheduler, which can adapt parameters in-flight based on real-time conditions. This learning-based approach is inherently extensible to complex multi-objective optimization for cloud and data center environments through schedulers, as well, as they may need to account for energy consumption and energy thermal limits and multi-tenant service level agreements (SLAs).

VIII. THREATS TO VALIDITY

Our synthetic example is illustrative rather than exhaustive. Results are sensitive to workload characteristics and parameter settings Qualitative synthesis may be biased by the omission of figures or equations from text extracted from the uploaded corpus.

The evaluation is done for CPU scheduling and does not take into account I/O scheduling interactions, memory management effects, or multi-core scaling behavior. All of these things can have a dramatic effect on real-world performance.

We thus provide methodological transparency and invite replication with other workloads representing different application scenarios and system configurations.

IX. CONCLUSION

While classical scheduling algorithms such as SJF and SRTF have been proven to have mathematical optimum to reduce the waiting time, this perfection is practically irrelevant in modern computing. The "so what" is that the main goal of scheduling has changed. The problem is no longer to arrive at an absolute minimal cost for clearing off a queue: it is to provide a system working with a mixture of orphaned jobs while maintaining responsiveness and fairness.

It could be proven that modern schedulers (like MLFQ and CFS) are not necessarily "better" than SRTF on a single metric, provided that they are isolated from each other. Instead, they are superior because they are designed to account for the variety

and the uncertainty of real-world workloads (something that the classical algorithms simply cannot do effectively).

We draw a clear line from the comparison: traditional schedulers are better suited to predictable, single-tenant setups, whereas new fairness- and feedback-based schedulers are better suited to interactive, heterogeneous and multi-tenant setups.

Our results suggest that using SRTF provides the best average waiting time, if burst times are known, whereas Round Robin provides fairness at high overhead cost. Modern schedulers, such as MLFQ and CFS deliver an easy-to-use combination that eliminates the trade-offs between competing design criteria.

The design criteria relate to the importance of matching scheduling policy to workload characteristics and system goals. While not the red herring that some would suggest, careful consideration of the tradeoffs between latency, throughput, and fairness in Performance Tradeoffs: Fine Tuning the Performance Tradeoffs is important for performance with parameter tuning.

Energy-aware policies together with ML-guided scheduling and hybrid techniques offer a huge promise for achieving future maturity in the field of process scheduling technology.[17] It's the fundamentals of classical algorithms, and performance and fairness needs today require extra techniques, beyond those classical fundamentals.

X. REFERENCES

1. S. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 10th ed., Hoboken, NJ: John Wiley & Sons, 2018.
2. A. S. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th ed., Boston: Pearson, 2014.
3. C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46-61, Jan. 1973.
4. N. C. Audsley, A. Burns, R. I. Davis, K. W. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, vol. 8, no. 5, pp. 284-292, Sept. 1993.
5. J. Leung and J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks," *Performance Evaluation*, vol. 2, no. 4, pp. 237-250, Dec. 1982.
6. L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Transactions on Computers*, vol. 39, no. 9, pp. 1175-1185, Sept. 1990.
7. S. Baruah, A. Burns, and R. I. Davis, "Response-time analysis for mixed criticality systems," *IEEE Real-Time Systems Symposium (RTSS)*, pp. 34-43, Dec. 2011.
8. C. S. Wong, I. K. T. Tan, R. D. Kumari, J. W. Lam, and W. Fun, "Fairness and interactive performance of O(1) and CFS Linux kernel schedulers," *International Symposium on Information Technology (ITSIM)*, 2008.
9. J. Bouron, S. Chevalley, B. Lepers, and W. Zwaenepoel, "The Battle of the Schedulers: FreeBSD ULE vs. Linux CFS," *USENIX Annual Technical Conference (ATC)*, 2018.
10. "Investigating Process Scheduling Techniques for Optimal Performance and Energy Efficiency in Operating Systems," *Journal of Information Technology and Digital World*, vol. 6, no. 4, pp. 388-400, 2024.
11. Y. C. Jani "Survey on Different Scheduling Algorithms of Processes in Operating System," *International Journal for Scientific Research and Advanced Technology (IJSART)*, vol. 4, no. 4, 2018.
12. Prof. N. Pawar, P. Sawade, S. Pathare, "Review on CPU Scheduling Algorithms," *International Journal of Scientific Research and Engineering Development (IJSRED)*, vol. 8, no. 5, 2025.
13. W. Pan, "Comparison and Analysis of Scheduling Algorithms: Exploring Performance, Time, Fairness, and Applicable Scenarios," *Highlights in Science, Engineering and Technology (HSET)*, 2023.
14. S. Kumari and G. Kumar, "Survey on Job Scheduling Algorithms in Grid Computing," *International Journal of Computer Applications*, vol. 115, no. 15, 2015.
15. H. Asghar and E.-S. Jung, "A Survey on Scheduling Techniques in the Edge Cloud: Issues, Challenges and Future Directions," *arXiv preprint arXiv:2202.07799*, 2022.
16. J. A. Stankovic, M. Spuri, K. Ramamritham, and G. Buttazzo, *Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms*, Springer, 1998.
17. D. Chen, A. K. Mok, and T.-W. Kuo, "Utilization Bound Revisited," *IEEE Transactions on Computers*, vol. 52, no. 3, pp. 351-361, 2003.
18. D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, "Job Scheduling Strategies for Parallel Processing," *Lecture Notes in Computer Science*, Springer, 2004.
19. Sinha, P., Prabadevi, B., Dutta, S., Deepa, N., & Kumari, N. (2020, February). Efficient process scheduling algorithm using RR and SRTF. In 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE) (pp. 1-6). IEEE.