Journal of Information Technology and Digital World (ISSN: 2582-418X)
www.irojournals.com/itdw/

Investigating Process Scheduling Techniques for Optimal Performance and Energy Efficiency in Operating Systems

Samrrutha R S.[1], Stephi Jacob[2], Akalya A.[3], Karthika L.[4], Anisha C. D.[5]

[1-4]ME Student, [5]Assistant Professor Department of Computer Science and Engineering, PSG College of Technology, Coimbatore, Tamil Nadu, India

E-mail: [1]24mz37@psgtech.ac.in, [2]24mz39@psgtech.ac.in, [3]24mz31@psgtech.ac.in, [4]24mz03@psgtech.ac.in, [5]ani.c.dass@gmail.com

Abstract

Process scheduling is a critical component of operating systems, determining the order in which processes are allocated CPU time. Traditionally, scheduling algorithms have aimed to optimize performance metrics such as throughput, latency, and CPU utilization. However, with increasing emphasis on energy efficiency in modern computing, particularly in mobile devices and data centers, energy consumption has become a key factor in evaluating scheduling strategies. This survey explores various process scheduling algorithms, focusing on their impact on energy efficiency. A comparative analysis is provided between traditional algorithms, like the Linux Completely Fair Scheduler (CFS), and energy-aware alternatives such as the Energy Fair Scheduler (EFS), which extends CFS by incorporating energy considerations in the scheduling process. Trade-offs in performance, including run-time and waiting time, are discussed, with case studies evaluating the effectiveness of energy-efficient schedulers. Performance metrics such as total energy consumption and CPU utilization are analyzed to highlight EFS's potential in reducing energy overheads while maintaining system throughput. Findings suggest that energy-aware scheduling algorithms, like EFS, can significantly improve energy efficiency without compromising performance, providing promising solutions for both battery-powered devices and energy-intensive server

DOI: https://doi.org/10.36548/jitdw.2024.4.006
Received: 18-12-2024. received in revised form: 04-01-2025, accepted: 22-01-2025, published: 03-02-2025

environments. Future directions in energy-efficient process scheduling research emphasize the need for dynamic energy management in modern operating systems.

Keywords: Process Scheduling, Linux Completely Fair Scheduler, Energy-aware Scheduling, Battery-powered Devices, Energy-intensive Server Environments.

1. Introduction

Process scheduling is a critical function of operating systems, ensuring the efficient allocation of CPU resources among competing processes [11]. It significantly influences system performance and user responsiveness by determining the order and duration of process execution. Traditional scheduling algorithms focus primarily on optimizing performance metrics such as throughput, response time, and CPU utilization. However, with the increasing emphasis on sustainable computing, energy efficiency has emerged as a vital consideration in scheduler design. Conventional schedulers, such as the Linux Completely Fair Scheduler (CFS), prioritize fairness in CPU access by balancing process demands with overall system load. While CFS effectively maximizes CPU utilization and maintains load balance, it largely neglects the energy implications of scheduling decisions [12]This oversight is especially problematic in contexts ranging from battery-dependent mobile devices to energy-intensive data centers, where energy efficiency has become an operational imperative.

Despite advancements in energy-aware scheduling, significant challenges remain. Current energy-aware schedulers often face trade-offs between reducing power consumption and maintaining acceptable system performance. These challenges include balancing energy savings with throughput, managing overhead from real-time energy monitoring, and dynamically adapting to diverse workload characteristics. Additionally, many energy-aware approaches rely on complex models, which can increase implementation overhead and limit scalability. This research investigates advancements in process scheduling, emphasizing energy-awareness integration into scheduling mechanisms. It evaluates existing algorithms, analyzes their contributions and limitations concerning energy efficiency, and explores a case study to demonstrate the practical application of energy-aware scheduling. This study aims to advance adaptable and efficient scheduling solutions for diverse computing environments by addressing the dual objectives of performance optimization and energy efficiency. The objectives of the research are:

- Analyze traditional and modern scheduling approaches to assess their effectiveness in addressing energy efficiency alongside performance metrics.
- Investigate methods for incorporating energy consumption optimization into scheduling policies without compromising system performance.
- Present a case study to illustrate the implementation and benefits of energy-aware scheduling in real-world computing environments.

2. Related Work

The existing literature describes the role of real-time operating systems in ensuring timely and predictable resource allocation to meet process deadlines. Scheduling, a key innovation in RTOS, is classified by deadlines: soft deadlines offering flexibility, firm deadlines allowing occasional misses, and hard deadlines requiring strict adherence to avoid catastrophic failures, such as in air traffic control. Advanced scheduling algorithms optimize metrics like CPU usage, turnaround time, waiting time, and load average, ensuring efficient process management even under heavy workloads. The researchers emphasizes on innovations like low jitter in hard RTOS for predictability and fast dispatch latency to minimize delays between task readiness and execution. Symmetric Multiprocessing (SMP) enables parallel task processing, improving performance. Robust memory management ensures reliable task execution. Scalability, stability, and security are essential in RTOS, maintaining performance under increased load or feature integration while preventing crashes. Components like the function library facilitate seamless communication between applications and the kernel. These scheduling innovations are vital in mission-critical applications such as industrial automation, where precise timing and reliability are crucial. They enhance efficiency, dependability, and performance in real-time systems, making RTOS indispensable for applications requiring high precision and low latency.The role of process scheduling plays a major part in optimizing the performance of parallel applications on high-performance computing (HPC) systems. It explores the interaction between the Linux OS scheduler and OpenMP application-level scheduling techniques during the execution of multithreaded codes. The study highlights how OS-level scheduling decisions, such as context switches and thread migrations, can interfere with the performance of compute-bound applications, especially when threads are not pinned. It finds that while the Linux scheduler attempts to balance system load, it can inadvertently degrade performance by introducing additional overhead and imbalances. The research

emphasizes the importance of coordinating scheduling decisions at both the OS and application levels to minimize these negative effects. This coordination improves load balancing and reduces performance degradation in concurrently executing applications. The findings underscore the need for further research into integrating OS and application-level scheduling techniques for more efficient execution of multithreaded applications [6,7].

Kaur et al.[8] examines the effectiveness of various task scheduling algorithms in cloud computing, specifically focusing on minimizing makespan, which represents the total time needed to complete all tasks. A lower makespan indicates better performance for a task-scheduling algorithm. Based on the results, FCFS (First Come First Serve) exhibited the highest makespan of 5150.73, indicating the lowest performance among the algorithms tested. PSO (Particle Swarm Optimization), with the lowest makespan of 2778.76, demonstrated the best performance, showing its efficiency in dynamic resource allocation. Between these two extremes, SJF (Shortest Job First) and RR (Round Robin) ranked in terms of makespan and performance, with SJF achieving a lower makespan of 4291.14 compared to RR's 4825.69. This ranking implies that SJF performs better than RR and significantly outperforms FCFS. Overall, PSO leads in performance, followed by SJF, then RR, and finally FCFS. The results confirm that PSO is the most effective algorithm in minimizing makespan, making it the optimal choice for enhancing cloud computing efficiency.

Orhean et al., 2018[9] describes a reinforcement learning-based framework for solving the task scheduling problem in distributed systems. The system architecture includes three main components: the World Model, Task Classifier, and Reinforcement Learning Agent. The World Model dynamically tracks the state of nodes, capturing details such as load levels, task relationships (parent, sibling), and other status parameters. Tasks are classified by the Task Classifier based on their type, enabling better prioritization and organization. The Reinforcement Learning Agent develops a scheduling policy (π\piπ) by learning from interactions with the system. It allocates tasks to output queues while considering the heterogeneity of nodes and the task dependencies represented in a directed acyclic graph.The proposed solution improves execution time and optimizes resource utilization by tailoring scheduling policies to the characteristics of the distributed environment. Additionally, the work introduces a platform that implements this reinforcement learning algorithm, providing scheduling as a service to distributed systems. This approach addresses challenges like varying

node capabilities and inter-task dependencies, aiming to improve productivity, fault tolerance, and efficiency in distributed computing architectures.

Reuther et al., 2018[10] describes an in-depth analysis of process scheduling in operating systems, focusing on job schedulers critical to modern big data architectures and supercomputing systems. Job schedulers serve as the backbone of parallel processing, allocating computing resources and controlling the execution of processes. Traditionally designed for supercomputers handling long-running computations, schedulers now face the challenge of managing big data workloads characterized by numerous short computations that process massive datasets. The study emphasizes the importance of scheduler efficiency as a fundamental factor limiting the performance of large-scale computing systems. A theoretical model is proposed to analyze scheduler latency, identified as a key performance metric for big data workloads. The model introduces two critical parameters: marginal latency and a nonlinear exponent. Experimental benchmarking is conducted on four widely-used schedulers Slurm, Son of Grid Engine, Mesos, and Hadoop YARN highlighting their performance under varying workloads. The findings reveal that computing system utilization drops significantly for computations lasting only a few seconds, falling below 10%. However, the study demonstrates that multi-level schedulers, such as LLMapReduce, can enhance utilization to over 90% by aggregating short computations, offering a practical solution to improve scheduling efficiency for short-duration tasks.

3. Survey on Various Process Scheduling Algorithms

First come first serve is one of the process scheduling algorithm which is very effective for the processes which has less burst time. It can make the waiting time of the process to escalate in a regular and even manner. This makes FCFS less optimal for complex processes. SJF is able to result in least average waiting time and turnaround time, which makes SJF as most efficient algorithm, especially in non-preemptive environment [5]. As SJF deals with shortest job first, so if reducing turnaround time and waiting time is priority, SJF is the best choice. Round robin is similar to FCFS and SJF where it is adjusted by quantum time. One of the biggest advantage of RR it ensures fair CPU time for all processes. It carefully balances the waiting time and fairness. If being fair to each process and time sharing is expected from a process scheduling algorithm RR will be the strong contender. Priority algorithm is suitable for the application with inconsistent resource demands, where the priority is determined by user or

the system. When it comes to high-load environments the challenge is to manage the dynamic priority allocation [1].

A process can be divided into foreground and background processes. The foreground process usually has a higher priority than the background process. Every process will have different scheduling needs and response-time demands. In multilevel queue scheduling (MLQ) system, the ready queue is segregated into multi-levels where the higher priority process occupies the top position. But this is a high possibility of starvation, if the process is pushed to the bottom of the queue. In a similar system, another idea is analyzed to solve the issue of starvation. A multi-level feedback queue (MLFQ) tries to move the processes that don't finish at the top level to a lower level in the queue. The issues here are deciding the number of queues to use, how long the time quantum should be for each queue, and the priority assignment of each process to avoid starvation. In MLFQ, lower priority queues get longer time quantum and higher priority queues receive shorter time quantum. The system aims to improve the turnaround time and reduce response [2]

CPU scheduling involves several performance metrics to evaluate efficient process execution. CPU utilization aims to keep the CPU occupied at all time. Throughput means the number of processes completed per time unit. The time of entry of a process is arrival time. Burst time is the total time required to complete the whole execution of the process. The completion time marks the moment when the execution gets completed. Turnaround time is the total time required to complete the process.

$$Turnaround\ time = completion - arrival\ time \quad (1)$$

$$Waiting\ time = Turnaround\ time - Burst\ time \quad (2)$$

Waiting time is the total time a process waits to complete the task. Response time is the duration from the submission of a request to the generation of the first response. Lastly, fairness ensures that all processes receive a fair share of CPU time. These metrics collectively help in evaluating the performance and efficiency of CPU scheduling algorithms [3]

Table 1. AT and BT of Processes

Process ID	Arrival time(ms)	Burst Time (ms) (quantum 5)	Priority
P0	0	12	3
P1	1	2	1
P2	2	3	3
P3	3	2	4
P4	4	6	2

Table 2. Average TT and WT Calculation

Process ID	TURNAROUND TIME (ms)				WAITING TIME (ms)			
	FCFS	SJF	RR	Priority	FCFS	SJF	RR	Priority
P0	12	25	25	20	0	13	13	8
P1	14	2	7	2	12	0	5	0
P2	17	7	10	23	14	4	7	20
P3	19	4	12	25	17	2	10	23
P4	25	13	23	8	19	7	17	2
AVG	17.4	10.2	15.4	15.6	12.4	5.2	10.4	10.6

The Table.2 justifies that FCFS and SJF are suitable for batch operating system because of its simplicity and efficiency in minimizing overall process completion time. RR and Priority algorithm are suitable for time sharing system because they aim for fairness and responsiveness. RR promises equal time for all process while priority prioritizes tasks based on urgency [4].

Tables 1 and 2 provide important insights into the performance of various scheduling algorithms by presenting key metrics: Turnaround Time (TT) and Waiting Time (WT). These

metrics are fundamental to assessing the efficiency of process scheduling. For instance, the Shortest Job First (SJF) algorithm achieves the lowest average TT (10.2 ms) and WT (5.2 ms), indicating its efficiency in minimizing delays for shorter processes. However, SJF's impracticality in real-time systems arises from its requirement for precise knowledge of job durations.

Round Robin (RR) offers better fairness across processes due to time-slicing but results in higher average TT (15.4 ms) and WT (10.4 ms), reflecting its overhead from frequent context switching. The Priority scheduling algorithm exhibits a trade-off: it effectively lowers TT and WT for high-priority tasks (e.g., P4) but increases these metrics for lower-priority ones (e.g., P3), as evident from its uneven distribution of results. First-Come-First-Served (FCFS) performs reasonably well overall but struggles with fairness for longer processes, as seen in its relatively high WT (12.4 ms) and TT (17.4 ms).

4. Case Study

The development of the Energy Fair Scheduler (EFS) involved careful modifications to the Linux Completely Fair Scheduler (CFS) to introduce energy-aware scheduling. To achieve this, the methodology focused on enhancing CFS's load-balancing mechanism to incorporate energy consumption as a key metric in scheduling decisions. The EFS adjusts the weight assigned to processes, dynamically prioritizing tasks with lower energy footprints while ensuring that CPU utilization remains high. This integrates the energy-monitoring capabilities, to track the power consumption of processes in real-time.

Parameter tuning played a vital role in optimizing EFS's performance. Parameters such as energy consumption thresholds, CPU load thresholds, and process weighting factors were calibrated to achieve an optimal balance between performance and energy efficiency. Extensive simulation experiments were conducted to evaluate EFS under diverse workloads, including both CPU-intensive and mixed workloads. The simulations involved benchmarking tools and synthetic workloads to measure energy usage and CPU utilization. The results were compared to the unmodified CFS to quantify improvements in energy efficiency and performance trade-offs. This systematic approach ensured that EFS maintained throughput and minimized latency while reducing energy consumption, making it a practical solution for devices ranging from battery-powered systems to high-performance servers. The advantages

and the disadvantages of the traditional and the energy-aware scheduler are depicted in Table 3.

Table 3. Advantages and Limitations of Traditional and Energy-Aware Schedulers

Aspect	Traditional schedulers	Energy-Aware Schedulers
Advantages	Optimized for throughput, low latency, and high CPU utilization. Simple, well-established algorithms with low computational overhead. Effective for performance-critical tasks and homogeneous workloads.	Reduces overall energy consumption, enhancing efficiency. Prolongs battery life for portable devices. Balances energy efficiency with acceptable performance. Suitable for diverse systems, including energy-constrained environments.
Limitations	Inefficient in energy-critical scenarios, leading to higher power consumption. Does not optimize for idle periods, resulting in energy waste. Unsuitable for systems where energy efficiency is a priority (e.g., mobile or IoT devices).	May introduce higher latency or reduced throughput for certain workloads. Requires complex energy monitoring and scheduling mechanisms. Implementation is more resource-intensive and may reduce scalability.

4.1 Process Scheduling and Energy Efficiency

Process scheduling is an essential component of an operating system that determines the order in which processes access the CPU. In modern computing, scheduling has primarily aimed to optimize performance metrics like throughput, latency, and CPU utilization. However, with the increasing importance of energy efficiency in computing from mobile devices to large data centers schedulers are now being evaluated on their energy consumption impact as well. Traditional schedulers, like the Linux Completely Fair Scheduler (CFS), focus on providing equal CPU access among processes, prioritizing fair distribution based on process needs and system load. CFS uses a load-balancing mechanism to maintain high CPU

utilization, making it suitable for maximizing computational performance, but it lacks provisions for actively managing energy consumption.

This study explores energy-aware scheduling approaches, specifically focusing on modifying the Completely Fair Scheduler (CFS) to develop an Energy Fair Scheduler (EFS). The EFS aims to introduce energy fairness into process scheduling by adapting scheduling policies to minimize unnecessary energy consumption while maintaining system performance. Parameters such as total energy consumption and CPU utilization are examined to evaluate the EFS's capability to optimize energy usage without compromising CPU performance. The findings demonstrate the potential of the EFS as a versatile scheduling mechanism for diverse devices, ensuring efficient CPU task handling and reducing power usage a critical factor for both battery-powered devices and energy-intensive server environments.

4.2 The Role of Energy Fairness in Process Scheduling

Energy fairness in process scheduling involves adjusting the scheduling algorithms to account for each process's energy impact, thereby distributing energy use more equitably across tasks and achieving an overall reduction in system power consumption. This contrasts with traditional schedulers that focus primarily on fairness in terms of CPU access time, often ignoring the energy implications of different types of workloads. The Energy Fair Scheduler (EFS) builds on the principles of the Completely Fair Scheduler (CFS) in Linux by integrating energy-awareness into its decision-making process. The EFS scheduler seeks to prioritize processes based not only on CPU demands but also on their energy footprints, adjusting scheduling decisions dynamically to reduce total system energy consumption.

In evaluating EFS, two main performance metrics are considered energy consumption and CPU utilization. By modifying the scheduling algorithm to manage power consumption more actively, EFS aims to reduce the energy overhead of idle periods and CPU-intensive processes without reducing CPU or system throughput. This ensures that the CPU remains busy with minimal wasted power, achieving a balance between performance and energy efficiency. In doing so, EFS could lead to significant energy savings across various systems, from low-power devices to data-intensive servers, while maintaining the processing capabilities necessary for high-performance applications.

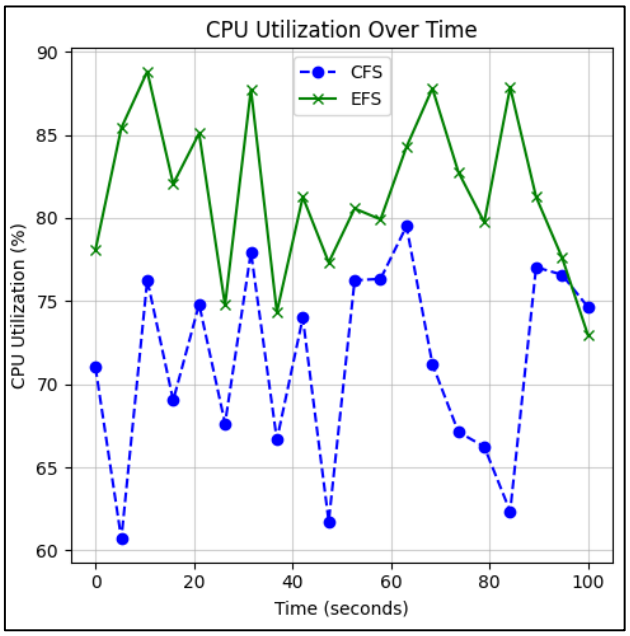

Figure 1. CPU Utilization Comparison: CFS vs EFS

Figure.1 compares CPU utilization trends over time for two schedulers, CFS and EFS. The blue dashed line represents the Completely Fair Scheduler (CFS), while the green solid line shows the Energy Fair Scheduler (EFS). The EFS maintains consistently higher CPU utilization, often between 75% and 90%, whereas CFS fluctuates significantly between 60% and 80%. This demonstrates EFS's ability to keep the CPU more active, minimizing idle periods and enhancing energy efficiency without compromising system throughput. The smoother trend in EFS utilization suggests improved scheduling stability, which is essential for performance in energy-efficient systems.

Figure.1 illustrates the CPU utilization trends over time for the Completely Fair Scheduler (CFS) and the Enhanced Fair Scheduler (EFS). The CFS exhibits pronounced fluctuations, reflecting inconsistent workload balancing and higher variability in resource allocation. Conversely, EFS demonstrates a significantly smoother utilization trend, indicative of more stable and efficient scheduling. This stability arises from EFS's ability to minimize sudden spikes and dips, ensuring more predictable CPU usage. The smoother trend directly translates to practical advantages, including improved system responsiveness, reduced latency under varying workloads, and enhanced energy efficiency due to the elimination of abrupt transitions in CPU demand.

5. Conclusion

This research highlights the importance of integrating energy efficiency into process scheduling to address the growing demand for sustainable computing. Traditional schedulers like the Linux Completely Fair Scheduler excel in fairness and CPU utilization but overlook energy implications, limiting their suitability for energy-sensitive environments. Energy-aware scheduling bridges this gap by minimizing power consumption alongside traditional performance metrics, offering viable solutions for power-constrained devices and sustainable data centers. The study evaluates existing algorithms, identifying their strengths and limitations, and demonstrates the practical benefits of energy-aware strategies through a case study.

Future work will focus on developing adaptive scheduling mechanisms that dynamically balance energy efficiency and performance in real-time. Incorporating machine learning techniques to predict workload patterns and optimize energy utilization could further enhance scheduler design. Additionally, extending energy-aware scheduling to heterogeneous systems, such as those involving GPUs and specialized accelerators, presents a promising avenue for achieving broader applicability and improved sustainability.

References

[1] Mehta, Shubh, and Harshad Mehta. "Detailed Analysis and Simulation of Various Process Scheduling Algorithms." International Journal of Algorithms Design and Analysis 6, no. 2 (2020): 43-52.

[2] Harki, Naji, Abdulraheem Ahmed, and Lailan Haji. "CPU scheduling techniques: A review on novel approaches strategy and performance assessment." Journal of Applied Science and Technology Trends 1, no. 1 (2020): 48-55.

[3] Omar, Hoger K., Kamal H. Jihad, and Shalau F. Hussein. "Comparative analysis of the essential CPU scheduling algorithms." Bulletin of Electrical Engineering and Informatics 10, no. 5 (2021): 2742-2750.

[4] Goel, Neetu, and R. B. Garg. "A comparative study of cpu scheduling algorithms." arXiv preprint arXiv:1307.4165 (2013).

[5] Ali, Shahad M., Razan F. Alshahrani, Amjad H. Hadadi, Tahany A. Alghamdi, Fatimah H. Almuhsin, and Enas E. El-Sharawy. "A review on the cpu scheduling algorithms: Comparative study." International Journal of Computer Science & Network Security 21, no. 1 (2021): 19-26.

[6] Korndörfer, Jonas H. Müller, Ahmed Eleliemy, Osman Seckin Simsek, Thomas Ilsche, Robert Schöne, and Florina M. Ciorba. "How do os and application schedulers interact? an investigation with multithreaded applications." In European Conference on Parallel Processing, pp. 214-228. Cham: Springer Nature Switzerland, 2023.

[7] Ismael, G. A., Salih, A. A., AL-Zebari, A., Omar, N., Merceedi, K. J., Ahmed, A. J., ... & Yasin, H. M. (2021). “Scheduling Algorithms Implementation for Real Time Operating Systems: A Review.” Asian Journal of Research in Computer Science, 11(4), 35-51.

[8] Kaur, Rajbhupinder, Vijay Laxmi, and Balkrishan. "Performance evaluation of task scheduling algorithms in virtual cloud environment to minimize makespan." International Journal of Information Technology (2022): 1-15.

[9] Orhean, Alexandru Iulian, Florin Pop, and Ioan Raicu. "New scheduling approach using reinforcement learning for heterogeneous distributed systems." Journal of Parallel and Distributed Computing 117 (2018): 292-302.

[10] Reuther, Albert, Chansup Byun, William Arcand, David Bestor, Bill Bergeron, Matthew Hubbell, Michael Jones et al. "Scalable system scheduling for HPC and big data." Journal of Parallel and Distributed Computing 111 (2018): 76-92.

[11] Omar, Hoger K., Kamal H. Jihad, and Shalau F. Hussein. "Comparative analysis of the essential CPU scheduling algorithms." Bulletin of Electrical Engineering and Informatics 10, no. 5 (2021): 2742-2750.

[12] Yu, Teng, Runxin Zhong, Vladimir Janjic, Pavlos Petoumenos, Jidong Zhai, Hugh Leather, and John Thomson. "Collaborative heterogeneity-aware os scheduler for asymmetric multicore processors." IEEE Transactions on Parallel and Distributed Systems 32, no. 5 (2020): 1224-1237