Priority Tuning Algorithm for Optimization of Process Scheduling and Memory Management in Linux

Praveen P, Kavya Swetha J, Rupesh A, Jay Adithya V
Department of Computer Technology
Madras Institute of Technology

***Abstract*—The efficient utilization of computing resources is critical for modern computer systems. In the domain of operating systems, process scheduling and memory management play a crucial role in optimizing system performance. Most of the operating systems use Completely Fair Scheduling, Round Robin, First Come First Serve and Priority-based scheduling. However, these algorithms prioritize either fairness or performance and do not consider user behavior. We propose a Dynamic Priority Tuning algorithm(DPT) for process scheduling and memory management in Linux operating systems. This algorithm aims to improve system performance by dynamically adjusting the priority of processes based on user behaviour. By continuously monitoring system resources and user activity, the algorithm can optimize process scheduling and memory management in real-time. Our algorithm is designed to prioritize performance over fairness while maintaining a reasonable level of fairness among user programs. This algorithm can benefit various domains that require high performance computing, such as scientific computing and real-time systems.**

***Index Terms*—Priority Tuning, User Behaviour, Performance Optimization, Reinforcement Learning, Process Scheduling.**

I. Introduction

Efficient utilization of computing resources is a critical aspect of modern computer systems. Operating systems play a crucial role in optimizing system performance through process scheduling and memory management. Most operating systems use a range of scheduling algorithms such as Completely Fair Scheduling, Round Robin, First Come First Serve, and Priority-based scheduling. However, these algorithms prioritize either fairness or performance and do not consider user behavior. The proposed research paper aims to address this limitation and proposes a priority tuning algorithm for optimizing process scheduling and memory management in the Linux operating system. The proposed algorithm takes into account the behavior of the user and dynamically adjusts the priorities of running processes to improve system performance. By continuously monitoring system resources and user activity, the algorithm can optimize process scheduling and memory management in real-time. The algorithm is designed to prioritize performance over fairness while maintaining a reasonable level of fairness among user programs. The proposed algorithm has the potential to benefit various domains that require high-performance computing, such as scientific computing and real-time systems. By prioritizing performance and dynamically adjusting priorities based on user behavior, the algorithm can optimize process scheduling and memory management, leading to improved system performance. The paper concludes by discussing the limitations of the proposed algorithm and suggesting future research directions to improve its effectiveness. In summary, we present a priority tuning algorithm for optimizing process scheduling and memory management in the Linux operating system. The algorithm is designed to prioritize performance over fairness while maintaining a reasonable level of fairness among user programs. The algorithm has the potential to benefit various domains that require high-performance computing, and the paper presents experiments to evaluate its effectiveness. The proposed algorithm represents an important step forward in optimizing system performance in modern computer systems.

II. Related Work

Negi and Kumar [1] applied machine learning techniques to improve Linux process scheduling. They used different algorithms, such as K-means clustering, decision trees, and Naive Bayes classifiers, to predict the behavior of a process based on past data. Mnaouer and Ragoonath [2] proposed an adaptive priority tuning system for optimized local CPU scheduling. The proposed system leverages feedback from system performance and user preferences to adjust the priority of tasks in real-time. Smith et al. [4] proposed a method for predicting application run times in parallel computing environments using historical information. They used linear regression to model the relationship between the application runtime and the system's workload characteristics. Oberthür and Böke [9] proposed a flexible resource management framework for self-optimizing real-time systems. The framework uses a feedback loop to adjust the resource allocation and scheduling based on the system's workload and resource availability. Cheng et al. [11] proposed a deep reinforcement learning-based approach for resource provisioning and task scheduling in cloud service providers. The approach uses a deep Q-network to learn the optimal allocation of resources and tasks based on the system's workload and resource availability.

III. Methodology

A. Dynamic Priority Tuning Algorithm

1) Collect the process statistics from the /proc directory using popen command and psutil function at regular

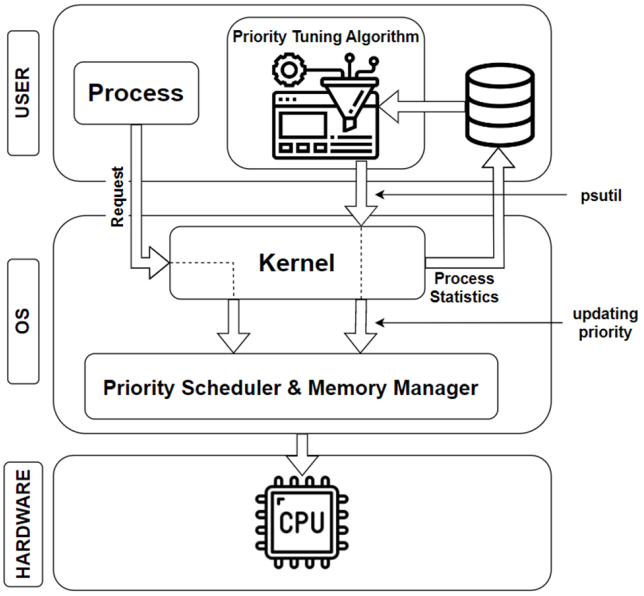

Fig. 1. Architecture design for DPT algorithm

intervals and store it in state space.

2) Define the action space as a list of integer values representing priority changes for a process.
3) Initialize the q_table, hyperparameters and reward function.
4) For each process in the state space, do the following:
5) Pass the state to the Reinforcement Learning agent.
6) Get the current priority of the process.
7) Choose an action using the epsilon-greedy policy function.
8) Apply the chosen action to the process priority and update it.
9) Observe the new state by collecting the next process in the state space.
10) Update the q_table with the new Q-value using the Q-learning formula.

IV. Results

The reinforcement learning agent is executed on an Ubuntu virtual machine. The agent adapted according to the system's environment and changed the process priorities. It is observed that the priorities are increased for processes that are frequently used by the user and require more system resources. The agent also decreased the priority of unwanted and resource-consuming processes. Due to increased priority values, important processes get more CPU time and memory. This results in increased system performance and an efficient use of system resources which is shown in Fig. 2 and Fig. 3.

V. Conclusion

In conclusion, the dynamic priority tuning algorithm that has been proposed for process scheduling and memory management has yielded promising results. The algorithm is able to dynamically adjust process priorities and memory allocation

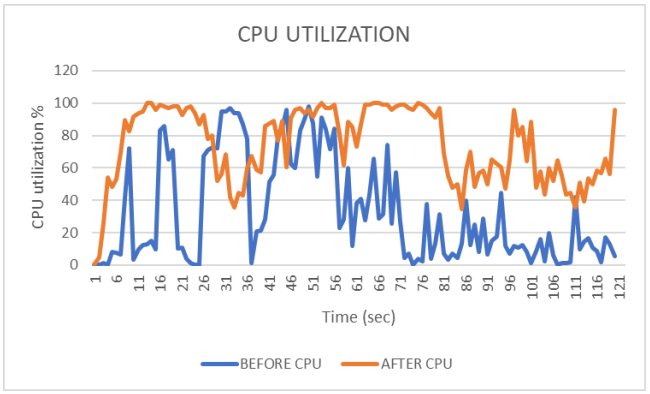

Fig. 2. CPU Utilization vs. Time

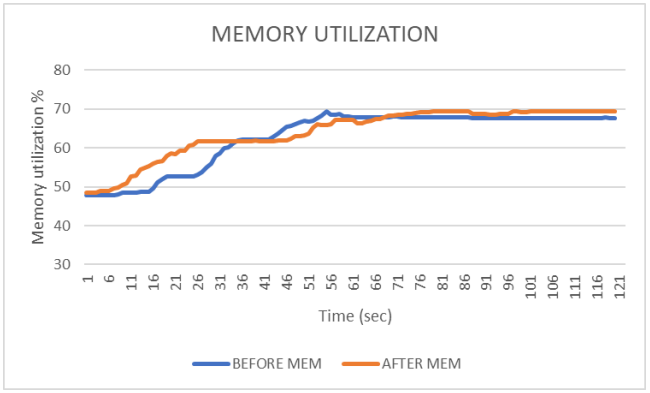

Fig. 3. Memory Utilization vs. Time

to optimize system performance by utilizing reinforcement learning techniques and historical data. When compared to existing scheduling algorithms, the algorithm's evaluation showed appreciable improvements in system performance. The dynamic priority tuning algorithm described in this paper has the potential to significantly improve the operating system's process scheduling and memory management performance.

References

[1] A. Negi and P. K. Kumar, "Applying Machine Learning Techniques to Improve Linux Process Scheduling", TENCON 2005 - 2005 IEEE Region 10 Conference, Melbourne, VIC, Australia, 2005, pp. 1-6, doi: 10.1109/TENCON.2005.300837.

[2] A. B. Mnaouer and C. Ragoonath, 'An Adaptive Priority Tuning System for Optimized Local CPU Scheduling using BOINC Clients', Journal of Physics: Conference Series, vol. 256, no. 1, p. 012019, Nov. 2010.

[3] T. Aivazian, Linux Kernel 2.4 Internals, The Linux Documentation Project, August, 2002.

[4] W. Smith, I. T. Foster, and V. E. Taylor. 1998. Predicting Application Run Times Using Historical Information. In Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing (IPPS/SPDP '98). Springer-Verlag, Berlin, Heidelberg, 122–142.

[5] S. Suranauwarat and H. Taniguchi, "The Design, Implementation and Initial Evaluation of An Advanced Knowledge-based Process Scheduler", ACM SIGOPS Operating Systems Review, volume: 35, pp: 61-81, October, 2001.

[6] Y. Xiao, S. Nazarian, and P. Bogdan, "Self-Optimizing and Self-Programming Computing Systems: A Combined Compiler, Complex Networks, and Machine Learning Approach," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 27, pp. 1-12, 2019, doi: 10.1109/TVLSI.2019.2897650.

[7] J. Torrellas, C. Xia and R. L. Daigle, "Optimizing the instruction cache performance of the operating system," in IEEE Transactions on Computers, vol. 47, no. 12, pp. 1363-1381, Dec. 1998, doi: 10.1109/12.737683.
[8] K. Ecker, D. Juedes, L. Welch, D. Chelberg, C. Bruggeman, F. Drews, D. Fleeman, and D. Parrott, "An optimization framework for dynamic, distributed real-time systems," in International Parallel and Distributed Processing Symposium (IPDPS03), April 2003, p. 111b.
[9] S. Oberthür and C. Böke, "Flexible resource management - a framework for self-optimizing real-time systems," in Proceedings of IFIP Working Conference on Distributed and Parallel Embedded Systems (DIPES'04), B. Kleinjohann, G. R. Gao, H. Kopetz, L. Kleinjohann, and A. Rettberg (Eds.), Kluwer Academic Publishers, Aug. 2004, pp. 23-26.
[10] V. Kalogeraki, P. Melliar-Smith, and L. Moser, "Dynamic Scheduling for Soft Real-Time Distributed Object Systems," Proceedings of the Third IEEE International Symposium on Object-Oriented Real
[11] M. Cheng, J. Li, and S. Nazarian, "DRL-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2018, pp. 129-134. doi: 10.1109/ASP-DAC.2018.8297294.
[12] P. Kishore Kumar and A. Negi, "Characterizing Process Execution Behaviour Using Machine Learning Techniques," in *DpROM WorkShop Proceedings, HiPC 2004 International Conference*, December 2004.