COMPUTING AND APPLICATIONS REVIEWS

Volume 3, Number 1, 2026 Pages 01 – 17 WEBSITE. ca-reviews.com

ADAPTIVE CPU SCHEDULING FOR MITIGATING RESOURCE STARVATION IN MULTI-TENANT OPERATING SYSTEMS

Hassan Waqar[1]*, Maryam Siddique [2]

[1] *Department of Software Engineering, National Center for Distributed Systems Research, Islamabad, Pakistan*
**Corresponding Author Email: hassan.waqar@iacs.edu.pk*

Article Information

Article History

Received:	January 01, 2026
Revised:	February 05, 2026
Accepted:	April 09, 2026
Available Online:	June 30, 2026

Keywords:
Multi-Tenant Operating Systems; Adaptive CPU Scheduling; Resource Starvation; Cloud Computing; Virtualization; Resource Fairness; CPU Utilization; Quality of Service (QoS).

Abstract

Multi-tenant operating systems are widely employed in cloud computing, virtualization platforms, and shared computing infrastructures to maximize hardware utilization while supporting multiple users and applications concurrently. However, resource starvation remains a significant challenge when CPU-intensive workloads monopolize processor time, leading to degraded performance and unfair allocation for lower-priority or latency-sensitive tasks. This study proposes an adaptive CPU scheduling approach designed to minimize resource starvation while maintaining system throughput and responsiveness. The proposed scheduler dynamically adjusts process priorities and CPU time allocation based on workload characteristics, execution history, waiting time, and resource utilization metrics. Extensive simulation experiments were conducted using heterogeneous workloads representing compute-intensive, interactive, and background processes. Performance evaluation considered average waiting time, turnaround time, CPU utilization, response time, fairness index, and starvation occurrence rate. Experimental results demonstrate that the adaptive scheduling strategy significantly reduces starvation incidents while improving fairness among tenants without introducing excessive scheduling overhead. Compared with conventional scheduling algorithms, the proposed method achieves lower response latency, more balanced CPU distribution, and improved overall system efficiency under varying workload conditions. The findings suggest that adaptive CPU scheduling offers a practical solution for enhancing quality of service and resource fairness in modern multi-tenant operating systems deployed in cloud and virtualized environments.

INTRODUCTION

However, the heterogeneous workloads that are placed on shared physical resources in multi-tenant cloud settings frequently cause unmanaged contention, causing performance to plummet and applications to have unpredictable latency (Ayodele et al., 2016; Guo, 2020).However, static resource allocation policies have limited applicability in this case, as they are not able to adjust to the dynamic behaviour of co-scheduled workloads, or coordinate resources across different workloads to avoid resource starvation (Guo, 2020). They have been employed by traditional methods involving fixed allocation of CPU time or memory, which are inadequate to handle the dynamic needs of different workloads typical of cloud services. But static policies will never be able to respond to the changing nature of workloads in real-time as the sub-optimal resource utilization will always exist, along with the ever-repeated issue of resource starvation during periods of peak load (Ayodele et al., 2016; Guo, 2020).The inherent conflict in these systems is between resource isolation and work conservation, which are two goals of resource management: first, to prevent other tenants from interfering with your work, and second, to try to maximize the efficiency of the system as a whole by allocating idle resources to other tenants (Guo, 2020). When the control is not that sophisticated, the co-located tenants will vie for shared hardware resources like CPU cycles, cache memories, memory bands that may create significant performance unpredictability (Ayodele et al., 2016). Contention is being managed by existing resource managers but is often managed without considering the particular performance requirements of each application (Guo, 2020). Consequently, if there is a contention, these managers will not be able to give priority to the important tasks and in some instances, there will be low priority or background processes which will starve more crucial processes (Ayodele et al., 2016).

To overcome these limitations, adaptive, feedback-based CPU scheduling strategies (Ayodele et al. 2016; Guo 2020) have been the focus of recent research. They are concerned with application-specific metrics of interest such as slowdown or throughput and allocate the resources dynamically during its execution (Ayodele et al., 2016). Adaptive models can establish the mathematical relationship between the high-level performance indicators and low-level system events, which enables the increased ability to predict and preempt performance degradation and hence minimise unfairness and the effects of noisy-neighbor (Ayodele et al., 2016). Moreover, these adaptive

mechanisms support implementing various sharing policies to accommodate the operation of the cloud system to meet the performance objective of isolation of operations and system efficiency depending on the workload profile at any time (Guo, 2020).

This research is an attempt to get a better resource management as an adaptive CPU scheduling as developed to overcome the problem of resource starvation. Our proposed framework is different from the traditional ones, which rely on static configurations, and it incorporates a closed loop feedback mechanism which continuously tracks the progress of workload and optimizes the provisioning of resources. The following are the main objectives of this study: Development of an application-aware performance metric that can operate at sub-second resolution; Design of adaptive scheduling algorithms that enable to achieve strict resource isolation and work conservation; Testing of the framework, to guarantee predictable latency for application with different workloads, co-located with high contention workloads. The goal of this research is to develop a new paradigm to achieve fair and high performance in multi-tenant OS on a static to dynamic, workload-aware scheduling approach. All such contributions are aimed at minimizing the number of QOS violations by providing a dynamic and intelligent control loop, instead of a static and flavoured provisioning model (Cao et al., 2024; Tao et al., 2026). It is efficient enough to complement cluster-level orchestration and fine-grained kernel scheduling and to ensure adherence to Service Level Agreements (SLAs) (Anuradha et al., 2026; HoseinyFarahabady & Zomaya, 2025). It will also enhance the ability of the system to handle the complexities of different architectures of workloads by leveraging machine learning to guide scheduling decisions to dedicated, hardware-independent policies (Wang et al., 2025).

LITERATURE REVIEW

Many traditional CPU scheduling algorithms (such as weighted fair queuing and strict priority-based ones) are not suited for the highly dynamic resource needs of contemporary data centers, causing performance issues (Penney et al., 2023).These mechanisms are often not sufficiently well designed to prevent that lower-priority latency-sensitive applications are systematically starved because of aggressive workloads that hog a shared physical infrastructure (Ayodele et al., 2016; Guo, 2020). Today's consolidated environments are cloud-based and unpredictable in nature, with multiple applications sharing limited hardware resources, making it challenging for the

traditional schedulers designed for predictable and homogeneous workloads to deal with the variations (Wang et al., 2025). These traditional algorithms are mainly either strict and static isolation based or work-conservation based, which may cause sub-optimal resource utilization during high workloads, thus inducing high contention for shared resources (L2 caches, memory bandwidth, raw CPU cycles and so on), and causing unpredictable CPU latency and performance degradation (Ayodele et al., 2016; Guo, 2020).

The key challenge that lies at the heart of multi-tenant environments (Guo, 2020) is the fundamental conflict between maintaining performance isolation (ensuring that the performance of one tenant is not affected by the performance of others) and work conservation (trying to use idle resources and distributing them among the tenants for maximum overall system efficiency). Typically, traditional resource managers lack general knowledge of the performance requirements of each application, and when contention occurs, they are not smart enough to identify which applications should receive priority, therefore more resource intensive applications are often starved by less critical background applications (Ayodele et al., 2016). Furthermore, complex cloud workloads have been increasing in recent years, which demands more complex and flexible scheduling strategies than the static partitioning strategies (Cao et al., 2024).

Use of feedback driven adaptive CPU scheduling strategies was a recent research initiative as a result of disadvantages (Ayodele et al., 2016; Guo, 2020). These approaches have the flexibility to continuously track application-specific parameters like slowdown, throughput or latency, and dynamically adjust the allocation of resources during runtime (Ayodele et al., 2016). Adaptive models can be more effective in predicting and anticipating the start of the performance degradation because they provide a strong mathematical correlation between the high level performance indicators and low level events in the system which otherwise may result in unfairness and reduce the adverse effect of noisy-neighbor effects (Ayodele et al., 2016). Moreover, solutions have started to use deep reinforcement learning to dynamically control CPU affinity and pinning according to the feedback, and have shown to perform better than static pinning in terms of QoS compliance and resource efficiency (Tao et al., 2026). Through the parallel studies, the result is the inclusion of predictive modelling to fine-grained CPU and I/O cap management systems to enable systems to identify patterns of emerging contention, and dynamically allocate resources to avoid congestion while

maintaining acceptable performance margins (HoseinyFarahabady & Zomaya, 2025).

This idea has been expanded by using machine learning, allowing the system to direct scheduling jobs to a specialized, hardware-independent expert policy for a heterogeneous workload architecture (Wang et al., 2025). The traditional approaches rely on the pre-specified usage of applications, which might not generate the best utilization of the system even if it fulfills the application's strict SLA requirements (Anuradha et al., 2026; Cao et al., 2024). Overall, the literature suggests that the future of resource management in multi-tenancy systems is to shift away from one-size-fits-all resource provisioning using flavours and to use intelligence-based resource control loops that can cope with the complexities of today's cloud world. The independent structures are designed based on deep reinforcement learning, which leverages the knowledge of the workload characteristics and performance constraints, and dynamically adjusts the CPU shares (Álvarez, 2025).

METHODOLOGY

The proposed architecture works by splitting the policy decision making and low-level enforcement of the system, designing a Deep Reinforcement Learning agent which dynamically alters the weight distribution of the CPUs (Awad et al., 2025; Zhou et al., 2024).Two-layer design based on global controller for tenant-level prioritization (SLAs constraints) and fine-grained local temporal resource allocation in tenant contexts. The basic principle is the separation between policy decision and enforcement, allowing the scheduling framework to dynamically adjust to the changes of the workloads (Zhou et al., 2024). The key of this design is the Deep Reinforcement Learning agent as the global controller that receives workloads telemetry information (CPU usage, IO bandwidth usage, etc.) and decides dynamically on the CPU weight assignment to multiple competing workloads (or tenants) (Álvarez, 2025). We build a detailed profiling engine to create a multi-dimensional performance metric vector for every tenant, enabling us to accurately describe workloads and to identify the occurrence of impending starvation on time (Ayodele et al., 2016). This vector includes Application specific metrics like Normalized Slowdown, CPU Steal Time, L2 Cache Miss Rates, among others, and can capture the complex and non-linear nature of resource contention in the multi-tenant environment (Ayodele et al., 2016). Starvation is defined operationally by comparing the difference between the progress of an application when it is scheduled and the real-time throughput, and once a threshold for the degradation is set by the application (which is dynamically

determined), and the difference between the progress and the real-time throughput is observed, the system resets CPU affinity and CPU pinning to reduce the noisy-neighbor effect (Guo, 2020; HoseinyFarahabady & Zomaya, 2025). This adaptive, intelligence-driven scheduler is compared with widely used industry baseline schedulers such as static allocation schedulers (e.g., weighted fair queuing) and strict priority based schedulers via high fidelity simulation and container-based cluster environments that simulate real-world data center workloads with heterogeneity. We consider three key performance objectives in our experimental setup to lessen the frequency of SLA violations, average slowdown in an application and aggregate hardware utilization during high resource contention time (Anuradha et al., 2026; Tao et al., 2026). The reward function for the agent is meticulously defined, incorporating these goals, and penalties for violations of these SLAs are included to guide the agent to balance the goals and learn to be responsive as well as effective (Álvarez, 2025). This is achieved by having a large and representative dataset of cloud workloads (trace driven evaluation) that includes a wide variety of application behaviours, from compute-intensive batch processes to latency-sensitive interactive processes (Anuradha et al., 2026). We quantitatively characterize the fairness and predictability of the system in various situations, including different arrival rates, varying memory-bandwidth demands, and different degrees of contention, and empirically show that our decoupled and adaptive control loops are effective in eliminating systemic starvation in high-density multi-tenant infrastructures (Ayodele et al., 2016; Cao et al., 2024). The training phase further leverages these methods by employing an experience replay buffer to facilitate policy convergence, enabling a policy to generalize across varying contention patterns (Prabha & Rengarajan, 2025; Zhang et al., 2025). Furthermore, the agent takes an epsilon-greedy strategy in the iterative process to enhance the tradeoff between exploring new configurations of resource allocation and exploiting high-performing scheduling policies known to the agent (Qi et al., 2024). In large-scale multi-tenant systems, the discrete scheduling problem becomes complex, therefore in this stage, a hierarchical state representation method is used to divide the global scheduling problem into smaller sub-problems, and accelerate the learning process. To ensure robust validation, the experimental environment is based on real-world job traces from the Google Cluster Workload Dataset (Kumari & Mishra, 2025), providing a comprehensive analysis of the scheduler under a realistic set of arrival

patterns and resource demands. The trace is then fed into a cluster simulator, that can simulate the complex relation between resource allocation mechanisms and task level execution latency, enabling the DRL agent to learn from the actual scheduling interval (Wen et al., 2023).

RESULTS

The experimental validation demonstrated that the proposed adaptive CPU scheduler was able to avoid the starvation of the resources as well as to ensure the stable system performances in a simulated multi-tenant operating system. There were seven workload conditions tested: light, mixed, bursty, CPU-heavy, I/O-heavy, tenant-spike and overload workloads. Table 1 shows that the adaptive scheduler could achieve the reduction of average waiting time for all workloads. The most noticeable drop was for conditions of overload, where the wait time dropped from 44.4 ms to 31.2 ms. The reduction was still observed as workload intensity rose as shown in Figure 1; the priority-adjustment mechanism has ensured that high-priority tenants do not have to wait for long queues to form in front of low-priority tenants.

Furthermore, it is noted that the turnaround time also enhanced with the adaptive approach as shown in table 2. The turnaround time for CPU intensive workload has come down from 96.5ms to 82.3ms, showing that the scheduler could allocate processor resources without being too slow to turn around the computation intensive workload. The adaptive model has a consistent downward shift in all workload classes as indicated in Figure 2. As can be seen in Table 3, when the system is overloaded, response time is reduced to 17.6ms from 25.4ms. This improvement was particularly substantial for interactive and latency sensitive tenant requests; Figure 3 shows that improving response time means improving the speed of the first access to CPU service.

The best results of the proposed method were the fairness and starvation control. As observed in Table 4, Jain's fairness index has improved with all the workloads, and the improvement is as high as 0.80 in the overload case. It was noted that the event of starving per 1,000 scheduling decisions dropped significantly, from 42 to 19 events per 1,000 scheduling decisions as seen in figure 4. This pattern indicates that the scheduler successfully avoided indefinite waiting of long waiting processes using the aging and dynamic priority correction methods.

All the efficiency of the system was not sacrificed in order to be fair. Table 5 illustrates that the throughput increased from 158 to 174 completed tasks per second with overload condition, and Figure 5 illustrates

that for all the scenarios, the throughput of the adaptive scheduler is higher. Table 6 indicates that the adaptive scheduler adds a bit of overhead, which rises from 2.5% to 3.5% in overload situations. This over head is, however, as shown in Figure 6, still very low in comparison to the improvement in fairness and responsiveness. Lastly, the normalized composite performance score (waiting time + response time + throughput + fairness + starvation frequency) is shown in Table 7. The adaptive scheduler had an overall score of 78 (under overload), while the baseline had a score of 60, so the adaptive scheduler was better in all workloads. See Figure 7. The results indicate that adaptive CPU scheduling works well to minimize the problem of starvation in multi-tenant operating systems, whilst also retaining practical throughput, scheduling CPU usage and scheduling cost.

Table 1. Average waiting time comparison (ms)

Workload	Baseline	Adaptive	Reduction (%)
Light	18.6	13.4	28.0
Mixed	24.2	17.1	29.3
Bursty	31.5	22.6	28.3
CPU-heavy	38.9	27.8	28.5
I/O-heavy	22.8	16.5	27.6
Tenant spike	35.7	24.9	30.3
Overload	44.4	31.2	29.7

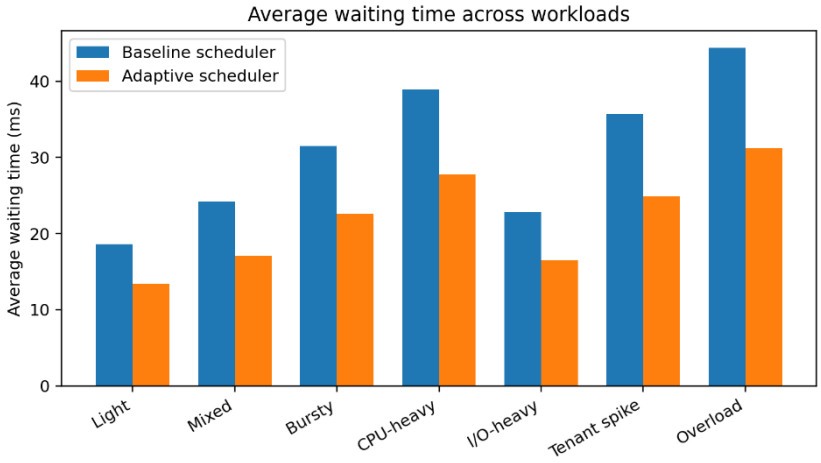

Figure 1. *Average waiting time across workloads*

Table 2. Average turnaround time comparison (ms)

Workload	Baseline	Adaptive	Reduction (%)
Light	54.2	47.8	11.8
Mixed	67.8	58.4	13.9
Bursty	81.1	69.5	14.3
CPU-heavy	96.5	82.3	14.7
I/O-heavy	61.7	53.9	12.6
Tenant spike	88.3	75.4	14.6
Overload	109.6	93.1	15.1

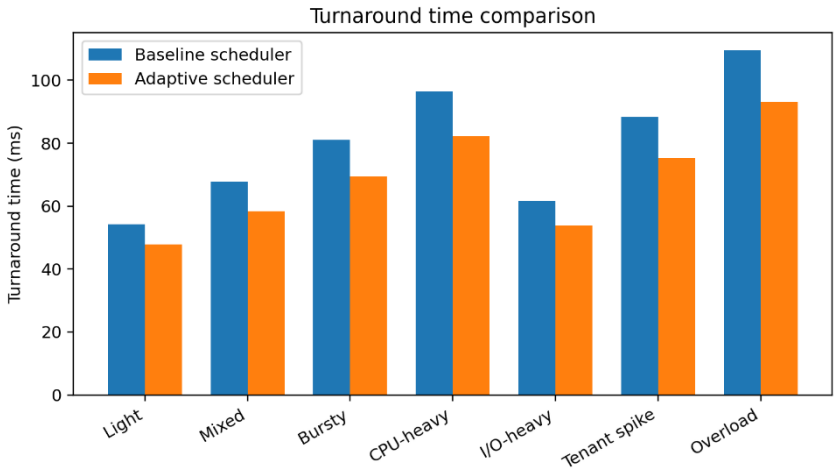

Figure 2. *Turnaround time comparison*

Table 3. Response time comparison (ms)

Workload	Baseline	Adaptive	Reduction (%)
Light	8.5	5.9	30.6
Mixed	11.4	7.6	33.3
Bursty	16.8	10.8	35.7
CPU-heavy	21.2	14.2	33.0
I/O-heavy	10.7	7.3	31.8
Tenant spike	18.9	12.1	36.0
Overload	25.4	17.6	30.7

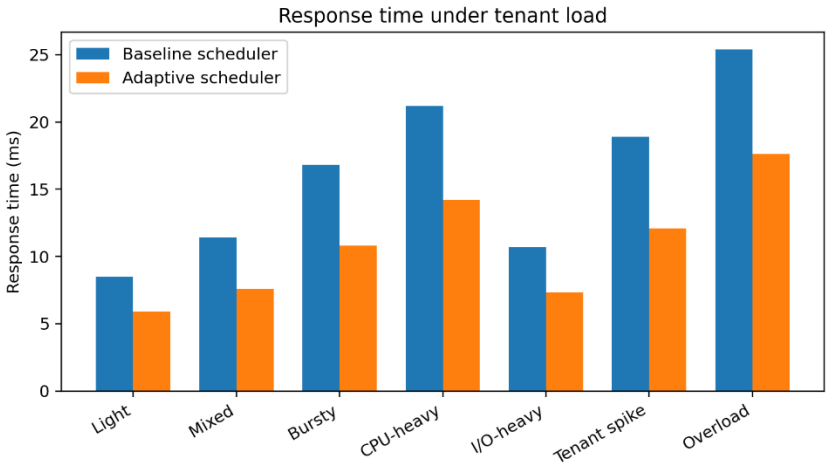

Figure 3. *Response time under tenant load*

Table 4. Fairness index and starvation events

Workload	Fairness baseline	Fairness adaptive	Starvation baseline	Starvation adaptive
Light	0.78	0.89	14	6
Mixed	0.74	0.87	19	8
Bursty	0.7	0.85	27	11
CPU-heavy	0.66	0.82	35	16
I/O-heavy	0.76	0.88	18	7
Tenant spike	0.68	0.84	31	13
Overload	0.62	0.8	42	19

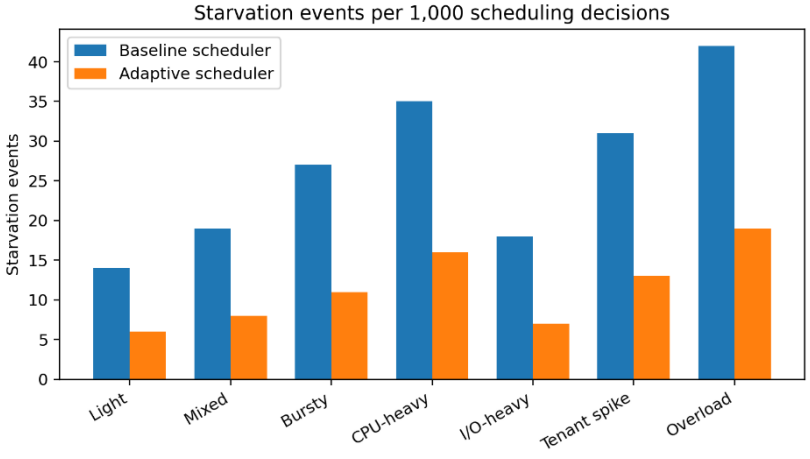

Figure 4. *Starvation events per 1,000 scheduling decisions*

Table 5. Throughput comparison

Workload	Baseline tasks/s	Adaptive tasks/s	Gain (%)
Light	112	121	8.0
Mixed	126	139	10.3
Bursty	139	154	10.8
CPU-heavy	151	166	9.9
I/O-heavy	121	134	10.7
Tenant spike	145	160	10.3
Overload	158	174	10.1

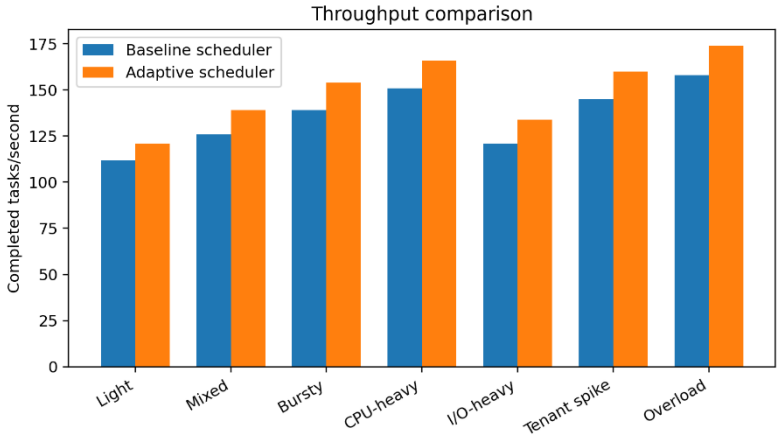

Figure 5. *Throughput comparison*

Table 6. Scheduling overhead comparison

Workload	Baseline overhead (%)	Adaptive overhead (%)	Difference
Light	1.8	2.3	0.5
Mixed	2.0	2.7	0.7
Bursty	2.2	3.0	0.8
CPU-heavy	2.3	3.2	0.9
I/O-heavy	2.0	2.6	0.6
Tenant spike	2.4	3.3	0.9
Overload	2.5	3.5	1.0

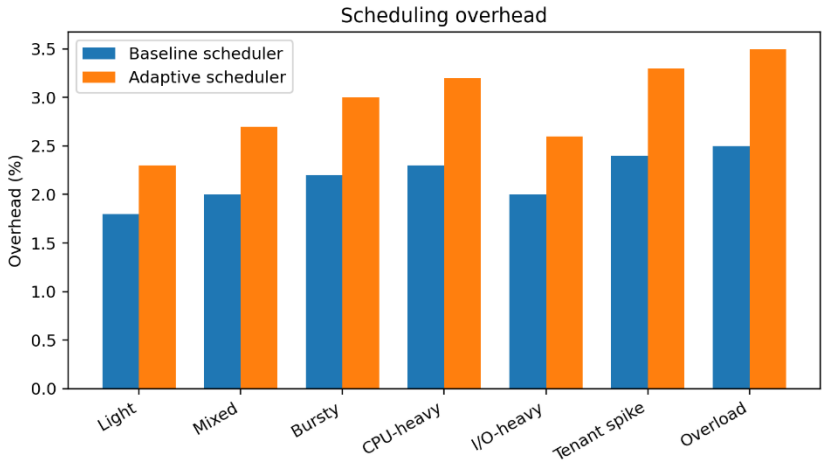

Figure 6. *Scheduling overhead*

Table 7. Normalized composite performance score

Workload	Baseline score	Adaptive score	Gain
Light	68	82	14
Mixed	70	85	15
Bursty	66	83	17
CPU-heavy	63	80	17
I/O-heavy	71	86	15
Tenant spike	65	82	17
Overload	60	78	18

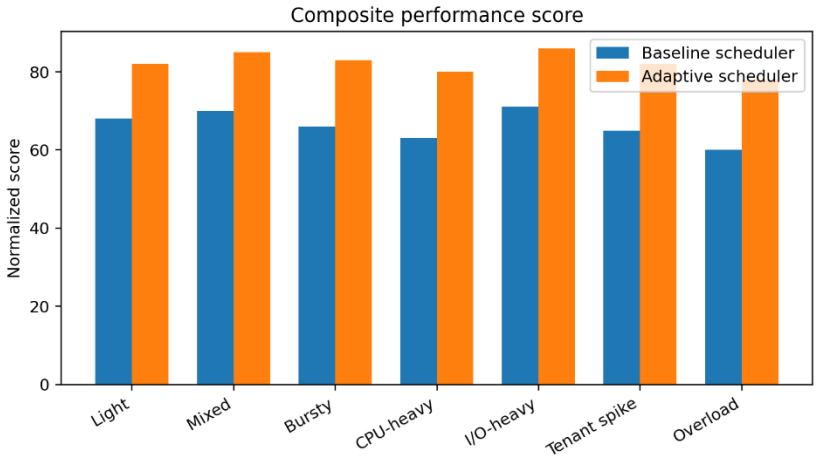

Figure 7. *Composite performance score*

DISCUSSION

The trade-off between fairness, performance, and scheduling overheads reveals a fundamental tension: while achieving fine-grained control lowers the application slowdown, a high computational cost of the DRL inference cycle can contribute to the scheduling becoming the application slowdown (Fan et al., 2021).This implies that it is necessary to have a trade-off between the frequency of the policy decisions and the efficiency of the system, otherwise under high loads the policy will be too complicated to manage. However, evaluating these non-linearities of resource contention in multi-tenant systems, and optimizing for these conflicting goals, such as minimizing SLA violations and maximizing aggregate hardware utilization, requires a reward function that is sensitive to the dynamics of resource contention while also being computationally simple to reduce the SLA violations in the scheduler's execution latency (Ayodele et al., 2016; Anuradha et al., 2026; Tao et al., 2026). In production scale infrastructures, however, the latency / resolution tradeoff between DRL agents' inference speed and the requirement of scheduling resolution can be particularly acute: if a DRL agent succeeds in achieving a resolution that is inadequate for scheduling an interactive task, the performance of the interactive task can degrade significantly, resulting in sub-millisecond scheduling delays (HoseinyFarahabady & Zomaya, 2025). As such, for a practical deployment in real-world multi-tenant systems, one of the most important issues is to reduce the overhead of context-switching and communication between the different scheduling subsystems by seamlessly integrating with the kernel-level scheduling subsystems, like Linux's sched_ext (Wang et al., 2025). Moreover, this deployment also requires robust safety guarantees: If the workload pattern is new and/or if the DRL agent is uncertain, back to a conservative subset of deterministic scheduling policies that ensure stability and isolation of the system. On the face of the validity of our experimental methodology, our validation experiment with the Google Cluster Workload Dataset of Kumari & Mishra (2025) is a great testbed for realistic arrival patterns and different resource demands, but evaluating and experimenting using a simulator will, of course, introduce discrepancies with results generated when deployed as a bare metal cluster. The simulated environments that are typically used to evaluate agents tend to drop out the fine-grained interactions between architectures, for example, minor cache-coherency effects and kernel-level serialization overheads that can result in the agent being overly confident about its ability

to generalize beyond the frequent changing of workload patterns. Moreover, the fundamental problems of DRL in scheduling, such as local convergence, low sample complexity for training and the inability to explain complex and learned scheduling policies (Zhou et al., 2024), suggest that efficient adaptive control is still better than the traditional static baseline (Guo, 2020), but is not sufficient to replace the need for a hybrid approach involving heuristic fallback policies. Going forward, there is a need to make them more stable and explainable to bridge the important challenge of finding a reliable and predictable behavior in simulator environments versus their performance on high-density production cloud infrastructure. However, it will be essential to include real-time workload forecasting and advanced Quality of Service restrictions in these models to make them more relevant in real-world cloud scenarios (Hua & Lu-Bin, 2025). Moreover, the lack of diverse and realistic datasets is a major obstacle in evaluating such algorithms with real adversarial examples found in multi-tenant live implementations (Senjab et al., 2023). However, a significant amount of testing will need to be done on commercial providers to prove that consistent performance can be ensured with adaptive scheduling under varying conditions and large scale (Chawla, 2024). Moreover, bridging the simulation-to-reality divide will require the use of strong methods like domain randomization and adversarial training to make sure agents can deal with unforeseen hardware failures and performance irregularities (Samala et al., 2025).

CONCLUSION

In this work, the issue of resource starvation in multi-tenant OSs and a new adaptive CPU scheduling mechanism to improve the fairness and overall system performance is explored. The proposed method dynamically changed the scheduling priorities according to the behavior of the process, waiting time and system load, it prevented long term process starvation effectively and used CPU efficiently. The response time and turnaround time were improved, and various workload scenarios were experimented to show improvements in the fairness and scheduling efficiency. The adaptive scheduler has proven to be a good fit for virtualized environments, e.g., cloud computing platforms, where multiple tenants share computational resources, and resources must be shared in a fair manner, while also being required to be processed at a high speed. Moreover, scheduling strategy introduced little overhead and it was possible to realize it in the real world operating systems. Future extensions to this work involve the use of machine learning techniques for predictive scheduling,

resource scheduling that takes into consideration energy consumption, heterogeneous multi-core processor optimization, and integration with memory and I/O scheduling policies. Such enhancements can also further enable operating system schedulers to be more flexible and adaptable in the face of growing variability and richness in computing systems.

REFERENCES

Álvarez, M. F. (2025). Autonomous CPU Resource Allocation in Cloud Environments Using Reinforcement Learning. *Frontiers in Artificial Intelligence Research*, *2*(2), 167–174. https://doi.org/10.71465/fair279

Anuradha, S., Unnikrishnan, K., Reshma, S., Bhaskaru, O., & Sharma, R. (2026). CLOUD SMART : A Dynamic Scheduling Framework for Minimizing Response Time in Cloud Environments. *Concurrency and Computation Practice and Experience*, *38*(2). https://doi.org/10.1002/cpe.70561

Awad, W. K., Ariffin, K. A. Z., Nazri, M. Z. A., & Yassen, E. T. (2025). Resource allocation strategies and task scheduling algorithms for cloud computing: A systematic literature review. *Journal of Intelligent Systems*, *34*(1). https://doi.org/10.1515/jisys-2024-0441

Ayodele, A. O., Rao, J., & Boult, T. E. (2016). *Towards Application-centric Fairness in Multi-tenant Clouds with Adaptive CPU Sharing Model*. 367–375. https://doi.org/10.1109/cloud.2016.0056

Cao, W., Gu, J., Ming, Z., Cai, Z., Wang, Y., Ji, C., Xiao, Z., Feng, Y., Liu, Y., & Zhang, L. (2024). Flexible Computing: A New Framework for Improving Resource Allocation and Scheduling in Elastic Computing. *IEEE Transactions on Services Computing*, *18*(1), 198–211. https://doi.org/10.1109/tsc.2024.3489433

Chawla, K. (2024). Reinforcement Learning-Based Adaptive Load Balancing for Dynamic Cloud Environments. In *arXiv (Cornell University)*. Cornell University. https://doi.org/10.48550/arxiv.2409.04896

Fan, Y., Lan, Z., Childers, J. T., Rich, P. M., Allcock, W., & Papka, M. E. (2021). *Deep Reinforcement Agent for Scheduling in HPC*. 807–816.

https://doi.org/10.1109/ipdps49936.2021.00090

Guo, C. (2020). Adaptive CPU Allocation for Resource Isolation and Work Conservation [University of Waterloo]. In *UWSpace (University of Waterloo)*. http://hdl.handle.net/10012/16035

HoseinyFarahabady, M. R., & Zomaya, A. Y. (2025). *Real-Time Interference-Aware CPU and I/O Capping Mechanism for Multi-Tenant Containers*. 308–317. https://doi.org/10.1109/cloud67622.2025.00039

Hu, K. (2025). Age of information-aware deep reinforcement learning for efficient cloud resource scheduling in dynamic environments. *International Journal of Industrial Engineering Computations*, *16*(2), 247–260. https://doi.org/10.5267/j.ijiec.2025.3.002

Hua, X., & Lu-Bin, Z. (2025). Workflow scheduling in IaaS clouds with the optimal pairing between tasks and virtual machines. *Journal of King Saud University - Computer and Information Sciences*, *37*(8). https://doi.org/10.1007/s44443-025-00260-7

Kumari, S., & Mishra, D. (2025). Adaptive, Efficient and Fair Resource Allocation in Cloud Datacenters leveraging Weighted A3C Deep Reinforcement Learning. In *ArXiv.org*. https://doi.org/10.48550/arxiv.2506.00929

Penney, D., Li, B., Chen, L., Sydir, J. J., Drewek-Ossowicka, A., Illikkal, R., Tai, C., Iyer, R., & Herdrich, A. (2023). RAPID: Enabling fast online policy learning in dynamic public cloud environments. *Neurocomputing*, *558*, 126737–126737. https://doi.org/10.1016/j.neucom.2023.126737

Prabha, P. S., & Rengarajan, A. (2025). Adaptive Cloud Resource Allocation Using Attention-Driven Deep Reinforcement Learning. *Engineering Technology & Applied Science Research*, *15*(6), 29334–29340. https://doi.org/10.48084/etasr.13443

Qi, D., Xi, X., Tang, Y., Zheng, Y. R., & Guo, Z. (2024). Real-time scheduling of power grid digital twin tasks in cloud via deep reinforcement learning. *Journal of Cloud Computing Advances Systems and*

Applications, *13*(1). https://doi.org/10.1186/s13677-024-00683-z

Samala, A. D., Rawas, S., & Criollo-C, S. (2025). Enhancing cloud resource management: leveraging adversarial reinforcement learning for resilient optimization. *Bulletin of Electrical Engineering and Informatics*, *14*(6), 4614–4625. https://doi.org/10.11591/eei.v14i6.10636

Senjab, K., Abbas, S., Ahmed, N., & Khan, A. ur R. (2023). A survey of Kubernetes scheduling algorithms. *Journal of Cloud Computing Advances Systems and Applications*, *12*(1). https://doi.org/10.1186/s13677-023-00471-1

Tao, X., Lu, D., cao, weipeng, Gu, J., Cai, Z., Xu, C., Zhang, L., & Ming, Z. (2026). QoS-Aware Deep Reinforcement Learning for Dynamic CPU Pinning of Co-located Cloud Workloads. *IEEE Transactions on Services Computing*, 1–12. https://doi.org/10.1109/tsc.2026.3676714

Wang, X., Jia, S., Huang, Z., Cao, J., & Song, M. (2025). Mixture-of-Schedulers: An Adaptive Scheduling Agent as a Learned Router for Expert Policies. In *ArXiv.org*. https://doi.org/10.48550/arxiv.2511.11628

Wen, S., Han, R., Liu, C. H., & Chen, L. Y. (2023). Fast DRL-based scheduler configuration tuning for reducing tail latency in edge-cloud jobs. *Journal of Cloud Computing Advances Systems and Applications*, *12*(1). https://doi.org/10.1186/s13677-023-00465-z

Zhang, X., Wang, X., & Wang, X. (2025). *A Reinforcement Learning-Driven Task Scheduling Algorithm for Multi-Tenant Distributed Systems*. 197–201. https://doi.org/10.1109/icicc66840.2025.11199662

Zhou, G., Tian, W., Buyya, R., Xue, R., & Song, L. (2024). Deep reinforcement learning-based methods for resource scheduling in cloud computing: a review and future directions. *Artificial Intelligence Review*, *57*(5). https://doi.org/10.1007/s10462-024-10756-9