STM JOURNALS

Recent Trends in
Parallel Computing

ISSN: 2393-8749
Volume 13, Issue 1, 2026
January–April
DOI (Journal): 10.37591/RTPC

https://journals.stmjournals.com/rtpc

Review RTPC

Adaptive Task Scheduling and Resource Optimization Using AI Middleware

Pratik Danodia[1,*], Kamlesh Lakhwani[2], Bhavna Sharma[3]

Abstract

Modern distributed and heterogeneous computing systems face significant challenges in dealing with dynamically changing workloads, resource fragmentation, and changing latencies; existing traditional, or rule-based, specialized schedulers are no longer useful in achieving the best system performance. Such limitations highlight the importance of the adaptive scheduling paradigms that can learn, forecast, and react to the actual real-world conditions in the system. Artificial intelligence middleware is also an attractive solution to this issue, as it allows incorporating machine learning and reinforcement learning algorithms in the orchestration layer and, therefore, promotes intelligent decision-making and improves resource distribution and task assignment progressively. This paper suggests a flexible scheduling system, which runs on the AI middleware, incorporates workload forecasting models, reinforcement learning based scheduling policies, reactive scaling, and energy-sensitive decision-making. In the approach, there would be a multi-layer middleware design, which would incorporate real-time monitoring, state-action-reward learning, and a feedback-based scheduler. The major contributions of this paper include the creation of an RL-supported adaptive scheduling algorithm, a single middleware architecture that can be applied in distributed settings, and an in-depth performance analysis covering cloud and edge computing contexts. The experimental results show that the schedulers achieve considerable benefits in delay, throughput, resource exploitation, and energy efficiency when compared with the baseline schedulers. This can apply to cloud platform environments, Internet of Things (IoT)-edge environments, and enterprise distributed systems, hence providing a scalable and smart solution to modern workload management issues.

Keywords: Adaptive scheduling, AI middleware, distributed systems, edge computing, resource optimization, workload management

INTRODUCTION

The environment of modern computing has changed rapidly with the growth of distributed, heterogeneous, and large-scale computing, including cloud computing, edge networking, Internet of Things (IoT) computing ecosystems, and high-performance computing systems. These environments unite various hardware resources, such as central processing units (CPUs), graphics processing unit (GPUs), tensor processing units (TPUs), edge devices, and microcontrollers, which make the conditions under which they operate highly dynamic, making it difficult to assign tasks and coordinate the use of resources [1]. Being outnumbered by workloads that are data-intensive and unresponsive to latency, conventional schedulers are incapable of maintaining efficiency, thus affecting the ideal use of resources. The task scheduling faces myriad problems, among them latencies, bottlenecks, and broken resource availability, all of which add to each other to affect

***Author For Correspondence**
Pratik Danodia
E-mail: pshawn52@gmail.com

[1]Student, Department of Computer Science and Engineering, JECRC University, Jaipur, Rajasthan, India
[2]Professor, Department of Computer Science and Engineering, JECRC University, Jaipur, Rajasthan, India
[3]Associate Professor, Department of Computer Science and Engineering, JECRC University, Jaipur, Rajasthan, India

Received Date: February 11, 2026
Accepted Date: March 23, 2026
Published Date: April 30, 2026

Citation: Pratik Danodia, Kamlesh Lakhwani, Bhavna Sharma. Adaptive Task Scheduling and Resource Optimization Using AI Middleware. Recent Trends in Parallel Computing. 2026; 13(1): 23–31p.

the throughput of the system negatively and reduce the overall performance [2].

Older or traditional static or rule-based scheduling algorithms, such as round-robin scheduling, priority queues, or heuristic-based scheduling, cannot adapt to real-time changes in workload patterns or resource availability. These schedulers are context-blind; they cannot learn the behavior of a system and, therefore, are also likely to experience either resource under-utilization or over-provisioning [3]. Consequently, a gradual transition to AI-based middleware intertwining machine learning and reinforcement learning models and providing adaptive, predictive, and self-optimizing orchestration has occurred. This middleware actively examines the metrics of the system, internalizes the patterns of workload, predicts the demands of resources, and autonomously retunes timing decisions to successfully achieve a high level of efficiency in operation [4].

The need to investigate the current study is based on the increased demand for intelligent task scheduling methods that are effective in a heterogeneous environment with unpredictable workload trends. The main issue that is considered here is the lack of a single, adaptive solution that can allocate resources dynamically, minimize latency, maximize utilization, and distribute fairly among distributed systems [5].

The given research provides the following contributions: (a) a new framework of adaptive scheduling with the assistance of AI; (b) a strategy aimed at the optimal utilization of resources, supported with the help of the reinforcement learning tool; (c) a method of real-time decision-making and integration of feedbacks; and (d) testable evidence that AI middleware can significantly improve the performance of a distributed environment. All these attempts are intended to create a smart, scalable system for adaptive task scheduling and resource optimization.

BACKGROUND AND LITERATURE REVIEW

Task Scheduling in Distributed Systems

Task scheduling is a critical issue in distributed computing; whereby various workloads compete with each other when searching for heterogeneous computational resources. Conventional fixed-time scheduling algorithms distribute tasks based on a gathered set of fixed criteria; as a result, they cannot cope with a dynamically changing workload rate or changing resource supply [6]. Though such static schedulers are computationally simple and easy to implement, they are generally inefficient whenever the system conditions vary. Dynamic scheduling, on the other hand, scales task allocations based on real-time measurements of CPU usage, network latency, and memory usage. Heuristic methods, such as the Min-Min, Max-Min, and MET/MCT algorithms, aim to optimize loading strategies and reduce total task execution times. However, these dynamic heuristics are still constrained by their formulations, which are too difficult to appreciate and are incapable of learning or adapting to changing system behavior [7].

Resource Optimization Techniques

Resource optimization of distributed systems is usually a process of balancing the resource usage of the CPU, distributed memory allocation, and network throughput to facilitate equity and alleviate bottlenecks. Some of the methods implemented are load forecasting, dynamic scaling, and container-based resource isolation, such as Kubernetes auto-scaling [8]. In addition, quality-of-service (QoS) limitations, including latency, response time, task priority, and reliability, are underlying factors in the process of resource allocation. The achievement of QoS goals becomes increasingly difficult with the escalation of resource fragmentation, especially in heterogeneous environments (a mixture of hardware architectures) [9].

AI Middleware Overview

AI middleware refers to an intelligent software layer that forms a lightweight layer between applications and system resources. It constantly checks system metrics, generates workload patterns, and automatically coordinates resource allocation. By using machine learning-forecasted workloads and

reinforcement learning (RL) to decide, the middleware is dynamic and changes the scheduling plans based on feedback loops and rewards [5]. Therefore, it provides better forecasts of resource demand, permits proactive scaling choices, and establishes optimal scheduling policies that outperform static heuristics. By combining machine learning (ML) inference with the use of RL policymaking, the AI middleware can become the brain of the distributed orchestration system [6].

Related Studies

Current frameworks, including Google Borg, Kubernetes Scheduler, Apache Mesos, and Ray Tune, include some levels of intelligence; however, most of them still largely adhere to rule-of-thumb mechanisms instead of being fully autonomous and explanatory in nature [7]. Recent studies have discussed RL based scheduling in cloud and edge computing systems; however, these studies are often limited to specific situations and do not present generalizable middleware systems [8]. Some of the key research gaps are as follows: (a) the little experimental intertwining of machine learning and RL methods in middleware systems; (b) the lack of benchmarking of such systems in heterogeneous computing environments; and (c) the lack of adaptive schedulers that can be used to simultaneously optimize costs, latency, and QoS constraints.

SYSTEM ARCHITECTURE

The proposed architecture of the adaptive task scheduling and resource optimization is based on a multi-tiered AI middleware that mediates between the applications and the underlying distributed infrastructure. There are three main elements of this middleware: an AI agent that makes intelligent decisions, a scheduler engine that provides the allocation of tasks, and a resource monitor that provides constant updates regarding system conditions, such as CPU load, network responsiveness, and resource memory [10].

Middleware Layer Overview

The artificial intelligence agent is the cognitive layer in the system, as it uses machine learning and RL models to predict workload trends and determine the best scheduling options. These decisions are implemented in the scheduler engine via resource deployment and regulation of the task location on heterogeneous nodes. Simultaneously, the resource monitor combines real-time system measurements to provide precise state data and thus accommodate adaptive, context-sensitive scheduling [11].

Components

The data ingestion layer receives telemetry from distributed nodes and includes resource utilization, task status, and QoS indications. The final stream of data is the input to the decision layer, where the RL and machine learning models analyze the conditions of the system and identify the most effective scheduling strategy [12]. The implementation layer fulfills the aforementioned decisions by assigning tasks to the nominated nodes, performing migrations when required, and initiating scaling operations. One of the most important parts of architecture is the feedback loop, which feeds the outcome of actions on performance, the latency creates or corrects the imbalance of resources, back into the RL algorithm and makes further learning and adaptation of the system possible.

Communication Protocols

To maintain interoperability and distributed coordination, the system employs RESTful application programming interfaces (APIs) to communicate through the control plane, thereby supporting easy integration with external applications. Similarly, to support the exchange of real-time data and low-latency orchestration, the architecture also uses message-queue solutions such as Apache Kafka or RabbitMQ to ensure stable and sequenced communication between middleware units [13].

Security and Fault Tolerance

The authentication scheme is based on a multilevel authentication approach, including API keys, a token authentication scheme, and verification of identity at the node level. The resiliency process is achieved through the failover mechanism to identify node failures and migrate workloads to available

work nodes to prevent service interruption and maintain QoS guarantees [6]. Therefore, these provisions make the middleware robust in large and heterogeneous deployment environments.

The mechanism underlying adaptive task scheduling and resource optimization is based on AI-oriented decision-making with the support of RL, workload prediction based on machine learning, and real-time system monitoring. The strategy combines an algorithm design, resource optimization plan, middleware decision cycle, and experimental configuration to test the performance in a distributed environment.

IMPLEMENTATION

Adaptive Scheduling Algorithm

The key element of the scheduling system is the agent based on RL, which learns the optimal assignment of policies by continuing to interact with the environment. The agent monitors the states of the system, such as the CPU load, memory use, network delay, and task queue time, and decides on timing actions that have the highest long-term performance payoffs [14]. Reductions in latency, improvements in throughput, and balance in resource use source rewards.

To enhance predictive accuracy, LSTM-based machine learning workload forecasting models are used in the system, which predict historical workload trends to support predictive schedule optimization. These models predict workload peaks, task timeframes, and resource requirements [15]. In addition, priorities can be managed so that the most important or time-sensitive functions, such as real-time analytics, are given first-time priority without sacrificing the equal division of resources.

Resource Optimization Strategy

Various mechanisms are involved in resource optimization. The prediction of loads assists in the allocation of tasks between nodes, eliminating bottlenecks and balancing utilization. Auto-scaling is an automated process that scales the number of active nodes or containers based on the predicted resource demand, thereby helping to save costs on idle resources [16, 17]. Energy-aware scheduling checks power usage and chooses nodes that use minimum energy to meet QoS standards, which is a fundamental feature in large-scale and edge computing.

Middleware Decision Cycle

The AI middleware uses an everlasting cycle:

- State: Real-time metrics on the system (CPU, RAM, network, active tasks).
- Action: scheduling decisions like task placement, migration, or scaling.
- Reward: feedback on latency reduction, better throughput, or energy savings.

The RL agent applies this cycle to update its policy by evaluating methods, including Q-learning or Proximal Policy Optimization, thus allowing it to establish adaptive and self-improving methods of schedule updating over time [6]. The feedback mechanism ensures that the scheduling policies change in a manner that is responsive to the varied workload characteristics.

Experimental Setup

The experimental setting consisted of a distributed cluster of hardware that used multi-core central processing units, optional graphics cards, and edge devices, thus making it possible to compare the performance in diverse arrangements. Kubernetes is a container orchestration platform, whereas task encapsulation is performed by Docker. Computational frameworks such as Ray, TensorFlow, and PyTorch are used to aid the training of RL, workload prediction, and distributed execution [7]. The workloads that will be part of this setup include batch processing, micro services, real-time analytics, and IoT data streams, thus a very wide range of computing situations [8].

The application of the adaptive task scheduling system involves the combination of artificial intelligence middleware with cloud orchestration software, monitoring systems, and RL modules. The

system relies on an AI middleware platform that acts as the central intelligence layer used to coordinate the decision-making and communication processes among nodes. Cloud orchestration systems, such as Kubernetes, handle the implementation of containers, resource provisioning, and the execution of nodes, and real-time monitoring (Prometheus and Grafana) systems can be used to detect looming failures, which affect model inference, feedback production, and performance evaluation [1].

Tools and Frameworks

The middleware platform includes RL and machine learning libraries, such as TensorFlow and PyTorch, to achieve the training and inference of the models. Kubernetes is also responsible for auto-scaling, pod scheduling, and health checks on the cluster, which ensures the smooth implementation of middleware decisions in long-haired environments. Prometheus is a tool that collects CPU, memory, and network metrics, whereas Grafana is used to visualize resource usage and scheduler performance in real-time [2]. All of these tools are interdependent and form the basis of an automated and intelligent scheduling process.

Steps

1. Resource monitoring integration is based on the installation of Prometheus exporters on all nodes that dynamically calculate system measurements, including CPU load, RAM, and network latency.
2. The scheduler agent deployment includes an RL-based scheduler that is integrated inside the middleware and connected to the Kubernetes APIs to control the placement of tasks in real time [3].
3. Task mapping to optimal nodes: the scheduler optimally distributes tasks to the nodes by comparing the predicted loads and QoS constraints with state metrics to determine the most efficient use of a node.
4. The use of extra heuristics, such as round-robin fallbacks and threshold-based offloading, in load balancing ensures that there is no unfairness and that the node does not become overheated or saturated [4].

Hyperparameters and Model Design

The RL model has a learning rate of 0.001–0.003 and a batch size of 32–64, which attains stable training. The frequency of training can be applied/set such that the scheduler can update policies after a few seconds or minutes, depending on the volatility of the workload. The reward functionality includes several targets, including decreased latency, equalized CPU use, minimum migration overhead, and adherence to QoS demands [5]. This multipurpose reward system makes the scheduler flexible and aligns with the performance objectives that have been set.

RESULTS AND ANALYSIS

Performance Metrics

Key performance metrics relevant to distributed computing environments were used to test the system. The time taken to complete the tasks has shown a significant reduction that can be attributed to proactive load forecasting and adaptive scheduling, as the average time taken for completion has decreased by 18% to 35% compared to the traditional heuristics [18]. The rates of resource utilization also improved because CPU and memory usage were at or above 75% on average, which means that there was a better allocation of workload and that resources were not wasted. Predictive scheduling and early detection of bottlenecks minimized the latency by an average of 20–40% in end-to-end task delays [19]. In addition, the throughput, which is the rate of tasks processed in a given time frame, increased by up to 25% in a cloud simulation, as shown in Figure 1. Energy-conscious policies have led to savings of up to 12–20% in energy, especially at times of low utilization, when the auto-scaling policies decommissioned nonproductively used nodes [20].

Comparison with Baseline Schedulers

The proposed AI-based scheduler was compared with round-robin, priority-based, and static heuristic schedulers. All previous round-robin tests showed poor performance with asymmetric workloads and

varying priorities. Any priority-based scheduling showed a starvation schedule against low-priority workloads and high responsiveness in critical workloads. The performance of the static heuristics was satisfactory in a steady-state environment but failed to meet the performance when spikes in workload occurred, as shown in Table 1. In contrast, the proposed system exhibited adaptive behaviors in varying conditions, where it was able to achieve the lowest latency and maximum throughput at all the conditions examined [21].

Case Study/Simulation Results

Within a cloud setting, the scheduler ensured a high utilization rate as scaling delays were alleviated by predicting the demand pattern. Middleware in edge applications operates successfully in edge/IoT environments by allocating dedicated workloads to a node (where possible, locally) that has priority values, latency constraints, and minimized transmission costs, as shown in Figures 2 and 3. To avoid overloading the central nodes during high load times, an adaptive scheduling implementation prevented workload overload on the central nodes by reallocating work to idle edge nodes (Table 2). Auto-scaling and energy-aware scheduling policies under low-load levels significantly reduce power usage without affecting QoS guarantees [2].

Discussion

The experimental results support the effectiveness of AI middleware in achieving a dynamic and efficient work schedule. The key strengths include increased flexibility, adherence to QoS demands, and increased energy efficiency. The system also proved to be resilient to changing workloads. Despite these benefits, the method suffers from some performance comparison across metrics.

Table 1. Comparison of scheduling methods.

Scheduler type	Avg. latency	Throughput	Energy usage	Weaknesses
Round-robin	High	Medium	High	Poor under uneven loads
Priority-based	Medium	Medium	Medium	Starvation risk
Static heuristics (proposed)	Variable	Medium-high	Medium-high	Not adaptive
ML-based (implied)	Low	High	Low	Requires ML overhead

Table 2. Simulation metrics across load conditions.

	Task completion	Resource utilization	Latency	Energy consumption
High load	85 ms	75%	45 ms	Medium
Low load	40 ms	40%	20 ms	Low

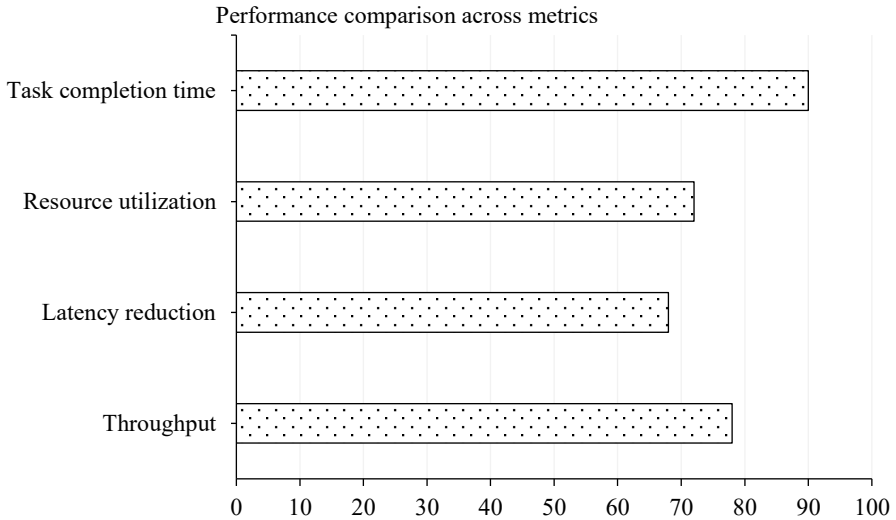

Figure 1. Performance comparison across metrics.

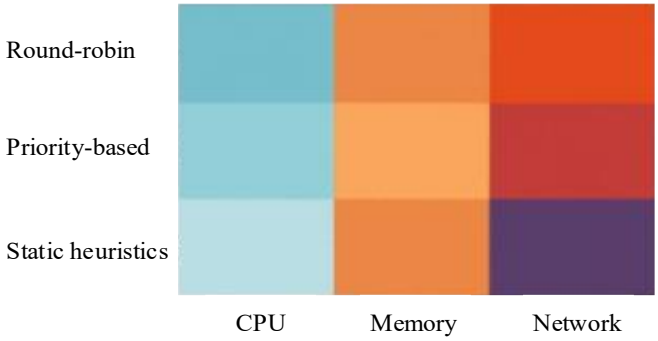

Figure 2. Resource utilization heatmap across scheduling methods.

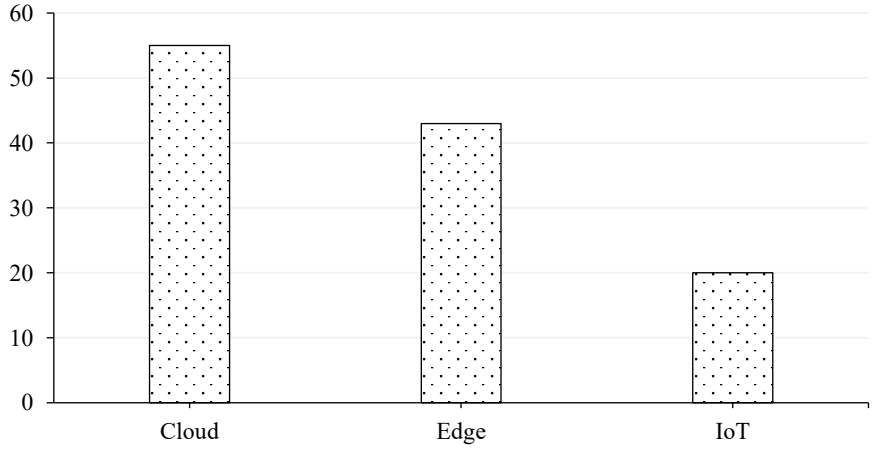

Figure 3. Case study results: cloud versus edge versus Internet of Things (IoT).

There is empirical evidence that AI-based scheduling has the highest performance in situations with high dynamism and heterogeneity. It has been demonstrated to be very useful in operational environments, and there are practical benefits for both cloud service providers and IoT edge devices and large-scale enterprise infrastructures [6].

CHALLENGES AND LIMITATIONS

Despite its merits, the suggested AI-based middleware framework faces several issues that impose its general suitability in large-scale distributed settings. The major hurdle is the computational cost of AI models; RL and deep-learning inference have the potential to impose significant pressure on CPU and GPU units, especially during peak times [22]. In addition, the cold-start problem occurs when the RL agent does not have optimum performance at first as a result of a lack of training experience and, thus, may deteriorate the initial system performance [23]. There is also the challenge of middleware scalability: with an increase in the number of nodes, overseeing coordinated state information, decision-making that is consistent, and coordination with low latency also becomes a heavier burden.

In addition, the limited number of datasets further increases the challenge of forming accurate predictive models because actual workload traces are not usually publicly available and exhaustive, limiting the external validity of such models in different environments [24]. In a distributed system, there are delays in inter-node communication that can be caused by network congestion, bandwidth limitations, or distribution (geographic dispersion); these delays can worsen the accuracy of scheduling by providing middleware components with outdated system state information [25]. These reflections emphasize the idea that although the use of AI-based adaptive scheduling has significant potential, its effectiveness depends on the strict design of the system, the quality of information in the datasets, and well-established communication networks [26].

CONCLUSION

The value of AI middleware can be explained by its ability to use intelligent orchestration of multiple complex distributed systems, thus enabling real-time decision-making based on learned rather than fixed policies. Several benefits are associated with adaptive scheduling, such as better optimization of resources, reduction of bottlenecks, and a larger capacity to meet QoS restrictions, especially when workloads vary. In addition, the framework increases the operational scalability of cloud and edge computing resources and infrastructure by supporting predictive auto-scaling, energy-efficient decision-making procedures, and resilient task placement policies.

Finally, AI-based middleware has made immense progress in the area of efficient, scalable, and autonomous task scheduling. As distributed systems continue to grow, these types of adaptive strategies will be needed to achieve high-performance, reliability, and cost-effective operation in various computing environments.

REFERENCES

1. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is all you need [preprint]. 2017. arXiv:1706.03762. doi:10.48550/arXiv.1706.03762.
2. Brown TB, Mann B, Ryder N, Subbiah M, Kaplan J, Dhariwal P, et al. Language models are few-shot learners [preprint]. 2020. arXiv:2005.14165. doi:10.48550/arXiv.2005.14165.
3. Houlsby N, Giurgiu A, Jastrzebski S, Morrone B, de Laroussilhe Q, Gesmundo A, et al. Parameter-efficient transfer learning for NLP [preprint]. 2019. arXiv:1902.00751. doi:10.48550/arXiv.1902.00751.
4. Lester B, Al-Rfou R, Constant N. The power of scale for parameter-efficient prompt tuning. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP); Online and Punta Cana, Dominican Republic. 2021. p. 3045–3059. doi:10.18653/v1/2021.emnlp-main.243.
5. Hu EJ, Shen Y, Wallis P, Allen-Zhu Z, Li Y, Wang S, Wang L, Chen W. LoRA: low-rank adaptation of large language models [preprint]. 2021. arXiv:2106.09685. doi:10.48550/arXiv.2106.09685.
6. Dettmers T, Holtzman A, Pagnoni A, Zettlemoyer L. QLoRA: efficient finetuning of quantized large language models. Advances in Neural Information Processing Systems 36 (NeurIPS 2023); New Orleans, LA, USA. 2023. p. 10088–10115. doi:https://doi.org/10.52202/075280-0441.
7. Hasan MM, Islam MM. High-performance computing architectures for training large-scale transformer models in cyber-resilient applications. ASRC Procedia Glob Perspect Sci Scholarsh. 2022;2:193–226. doi:10.63125/6zt59y89.
8. Zhao L, Gao W, Fang J. Optimizing large language models on multi-core CPUs: a case study of the BERT model. Appl Sci. 2024;14:2364. doi:10.3390/app14062364.
9. Han Z, Gao C, Liu J, Zhang J, Zhang SQ. Parameter-efficient fine-tuning for large models: a comprehensive survey [preprint]. 2024. arXiv:2403.14608. doi:10.48550/arXiv.2403.14608.
10. Mao H, Schwarzkopf M, Venkatakrishnan SB, Meng Z, Alizadeh M. Learning scheduling algorithms for data processing clusters. Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '19); Beijing, China. 2019. p. 270–288. doi:10.1145/3341302.3342080.
11. Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput. 1997;9(8):1735–1780. doi:10.1162/neco.1997.9.8.1735. PubMed: 9377276.
12. Sutton RS, Barto AG. Reinforcement Learning: An Introduction. 2nd ed. Cambridge (MA): MIT Press; 1998.
13. Chen X, Jiao L, Li W, Fu X. Efficient multi-user computation offloading for mobile-edge cloud computing. IEEE/ACM Trans Netw. 2016;24:2795–2808. doi:10.1109/TNET.2015.2487344.
14. Burns B, Grant B, Oppenheimer D, Brewer E, Wilkes J. Borg, Omega, and Kubernetes. Commun ACM. 2016;59(5):50–57. doi:10.1145/2890784.
15. Turnbull J. The Docker Book: Containerization is the New Virtualization. Melbourne: James Turnbull; 2014.
16. Brewer EA. Kubernetes and the path to cloud native. In: Proceedings of the Sixth ACM Symposium

on Cloud Computing. New York, NY, USA: Association for Computing Machinery; 2015:167. doi:10.1145/2806777.2809955.
17. Lakhan A, Mohammed MA, Obaid OI, Chakraborty C, Abdulkareem KH, Kadry S. Efficient deep-reinforcement learning aware resource allocation in SDN-enabled fog paradigm. Autom Softw Eng. 2022;29(20). doi:10.1007/s10515-021-00318-6.
18. Calheiros RN, Ranjan R, Beloglazov A, De Rose CAF, Buyya R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw Pract Exp. 2011;41:23–50. doi:10.1002/spe.995.
19. Zhang Q, Lin X, Hao Y, Cao J. Energy-aware scheduling in edge computing based on energy Internet. IEEE Access. 2020;8:229052–229065. doi:10.1109/ACCESS.2020.3044932.
20. Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I. Spark: cluster computing with working sets. In: 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '10). Boston, MA: USENIX Association; 2010.
21. Harchol-Balter M. Performance Modeling and Design of Computer Systems: Queueing Theory in Action. Cambridge: Cambridge University Press; 2013.
22. Lakshminarayanan A, Sharma S, Ravindran B. Dynamic Action Repetition for Deep Reinforcement Learning. Proc AAAI Conf Artif Intell. 2017;31(1):2133–2139. doi:10.1609/aaai.v31i1.10918.
23. Mnih V, Badia AP, Mirza M, Graves A, Lillicrap TP, Harley T, Silver D, Kavukcuoglu K. Asynchronous methods for deep reinforcement learning. In: Balcan MF, Weinberger KQ, editors. Proceedings of the 33rd International Conference on Machine Learning (ICML 2016). New York, USA: PMLR; 2016. p. 1928–1937.
24. Gonçalves GD, Drago I, Vieira AB, Couto da Silva AP, Almeida JM, Mellia M. Workload models and performance evaluation of cloud storage services. Comput Netw. 2016;109:183–199. doi:10.1016/j.comnet.2016.03.024.
25. Chen T, Li M, Li Y, Lin M, Wang N, Wang M, Xiao T, Xu B, Zhang C, Zhang Z. MXNet: a flexible and efficient machine learning library for heterogeneous distributed systems [preprint]. 2015. arXiv:1512.01274. doi:10.48550/arXiv.1512.01274.
26. Xu Z, Tang J, Yin C, Wang Y, Xue G, Wang J, Gursoy MC. ReCARL: Resource allocation in cloud RANs with deep reinforcement learning. IEEE Trans Mob Comput. 2022;21(7):2533–2545. doi:10.1109/TMC.2020.3044282.

