A Dissertation Submitted to China University of Geosciences For the Bachelor Degree of Computer Science and Technology

Dynamic Reinforcement Learning-based Task Scheduling Optimization Method for Load Balancing in Cloud Environments

Student Number:	20211004609
Student Name:	Xinxiang Chen
Major:	Computer Science and Technology
Tutor:	Prof. Xiaoyu Chen
	Prof. Mustafa Bilgic
	Prof. Yousef Elmehdwi
Faculty:	School of Computer Science

May, 2025

Statement of originality of bachelor's thesis of China University of Geosciences

I declare that my bachelor's thesis "Dynamic Reinforcement Learning-based Task Scheduling Optimization Method for Load Balancing in Cloud Environments" is the result of my independent research work during my bachelor's degree study period at China University of Geosciences (Wuhan) under the guidance of my tutor. The paper does not contain the research results that have been published or written by others, except for the parts already indicated, and the relevant personnel who have provided assistance in the completion of the thesis have been mentioned and thanked in this thesis.

This bachelor's thesis does not violate academic ethics and academic norms, has no infringement, and I am willing to undertake the legal responsibility and legal consequences arising therefrom.

Dissertation author:_____

Date: May 27, 2025

摘　要

随着云计算技术的飞速发展，越来越多的应用场景依赖于云计算来实现。由于云计算具有弹性、高可用和按需计费等优势，许多公司都将其实际业务部署于云计算平台之上。然而，如何在任务请求规模变化过大、计算资源异构性强的环境下实现高效的任务调度与负载均衡，成为云计算系统设计中的一大关键难题。任务调度的核心目标是将用户提交的计算任务合理地分配至虚拟机（VM），以提升资源利用率、降低系统响应时间并保持负载的均衡性。然而，这一问题主要面临两个技术难点：（1）任务特征和资源状态的高度动态性与不确定性使得调度策略难以提前规划或做出静态设定；（2）虚拟化平台中各虚拟机异构性大，资源请求时常出现冲突，导致传统静态或启发式调度方法难以获得全局最优解。云计算任务调度问题在算法复杂度上属于 NP 难问题，尤其在大规模任务并发的场景中，更容易造成资源浪费、系统瓶颈及用户体验下降。因此，研究一种具有自适应能力、能够动态学习最优调度策略的智能任务调度方法，对推动云计算平台的智能化与高效化运行具有重要理论价值与实践意义。

当前云计算环境下的负载均衡研究主要集中在四类方法：传统静态方法、基于软件定义网络（SDN）、基于机器学习（ML）以及多目标优化算法。（1）传统的静态方法包括：轮询（Round Robin）、最小连接（Least Connections）、源地址哈希（Source IP Hash）等。这些方法实现简单，部署高效，但是对于负载动态变化较大的情况下表现不佳。（2）SDN 方法通过解耦控制平面与数据平面，实现网络资源的集中控制与灵活调度，已被广泛应用于负载监测与流量控制，但其部署复杂性较高，依赖高度灵活的网络基础设施。（3）机器学习方法则强调数据驱动的策略自适应能力，包括监督学习、深度学习与强化学习等技术，可用于任务预测、资源分配与调度优化，具有较强的动态性与泛化能力，但往往对数据质量和计算资源要求较高。（4）多目标优化方法则致力于在任务调度、资源利用率、能耗等多个目标之间寻求平衡，常见的技术包括粒子群优化、萤火虫算法、混合智能优化等，在提升系统性能方面表现突出，但也面临收敛速度慢、动态适应性弱等问题。综上，尽管现有方法各具优势，但在面对复杂、高动态云计算环境时仍存在泛化性差、实时性不足等挑战。

针对任务特征的高度动态性在复杂、高动态云计算环境下导致服务器负载均衡效率低下的问题，本文基于强化学习模型，采用优化任务调度策略的方法，开展了云环境中任务调度方法的研究，设计了任务调度问题的模型，提出了一种基于强化学习的动态任务调度优化方法——动态强化学习任务调度算法（Dynamic Reinforcement Learning-based Task Scheduling，简称 DRL-TS），通过优化任务调度分配策略实现了云环境中服务器的负载均衡。该算法基于深度 Q 网络（Deep Q-Network，DQN）构建，通过与环境的交互不断优化任务分配策略，从而在复杂的多目标云环境中实现任务调度与负载均衡的双重优化。本文所提出的方法通过强化学习模型建立状态-动作映射函数，使模型具备自我学习能力，能够根据不同的任务数量、虚拟机负载、资源利用率等信息，动态地感知各虚拟机状态并做出决

策，将任务分配到最优的虚拟机上，从而降低任务完成总时间，提升系统的运行效率以及资源利用率。该方法适用于任务执行时间变化大、资源请求频繁、计算压力高的动态环境场景，在理论与工程上均具有显著的创新性。

在总体设计上，本文首先基于马尔可夫决策过程（Markov Decision Process, MDP）对云平台中的任务调度问题进行了系统建模，其中环境定义为数据中心中各虚拟机（VM）的资源使用状态，智能体为本文提出的动态强化学习任务调度算法（DRL-TS），目标函数则涵盖多个优化维度，包括最小化总任务完成时间（makespan）、最小化调度总开销、最小化系统负载不均衡度以及最大化整体资源利用率。在状态建模方面，本文定义的状态空间为一个高维向量，结合了每台虚拟机当前的 CPU 负载率与待分配任务在各 VM 上预计执行时间，通过这种方式精确刻画了系统运行时的资源动态，增强了调度策略对系统变化的感知能力。在动作空间设计中，每一个动作对应于将当前待调度任务分配至某一台虚拟机，构建了一个 N 维离散动作空间（N 为虚拟机数量），使得策略决策具有针对性与可扩展性。

在奖励函数构造方面，本文综合考虑了多目标优化需求，设计了融合任务完成时间、负载均衡性与资源使用成本的复合型奖励机制。具体实现中，分别计算任务调度后的 makespan 变化率、负载不均衡率及调度引起的资源代价，并将其归一化生成即时奖励信号，以用于强化学习中的策略评估与更新。该设计兼顾了全局优化目标与调度反馈，显著提升了策略学习的收敛效率与最终性能。

同时，在训练机制方面，本文采用了深度强化学习中的 DQN（Deep Q-Network）架构来构建智能体，并引入经验回放机制（Replay Buffer）与双网络结构（即主网络与目标网络）以提升训练过程的稳定性与策略的收敛速度。具体而言，经验回放机制通过在每一步交互后将状态转移样本存储进回放缓冲区，并在训练中以小批量方式随机采样，打破数据间的相关性，避免因样本顺序引起策略震荡；而双网络结构则将策略网络与目标网络分开，主网络用于决策与训练更新，目标网络用于生成稳定的目标 Q 值，并以固定步长进行软更新，从而有效减少 Q 值估计的偏差，提升学习稳定性。在探索策略方面，本文设计了 ϵ 动态调整策略，并使用线性衰减机制对探索率 ϵ 进行动态调整：初始阶段 ϵ 设为 0.9 以增强探索，随后随训练轮次逐步衰减至 0.1，使智能体在前期能够充分试探潜在调度策略空间，在后期逐步趋于稳定策略利用。该机制确保了策略在训练早期具备较强的探索能力，同时在收敛阶段具有更高的执行效率与精度。

在系统实现方面，本文基于 CloudSim 7G 仿真平台构建了完整的实验环境，对所提出的 DRL-TS 算法进行了系统验证与多组对比试验。CloudSim 作为当前主流的云计算模拟工具，其 7G 版本在可扩展性、模块化支持和时间粒度控制方面均有显著提升，为复杂调度策略的可重复性测试与性能评估提供了良好支撑。为全面评估算法的泛化能力，本文采用了三类典型数据集：包括任务长度分布随机的数据集（Random）、符合真实服务请求模式的 GoCJ 数据集，以及模拟高负载场景的 Synthetic Workload 数据集。所设计的任务调度实验覆盖了不同任务长度、虚拟机异构配置与多样负载强度场景，确保方法在多变云环境下的广泛适应性与鲁棒性。

对比方法包括传统静态调度策略（如 Round Robin）与启发式算法（如 PSO），评估指标涵盖任务平均响应时间、总完成时间（makespan）、虚拟机利用率、资源调度成本与负载不均衡度多个性能维度。

在强化学习算法的实现方面，本文基于 PyTorch 框架构建了 DQN 网络结构与完整的智能体训练流程。PyTorch 作为当前主流的深度学习平台，具备动态图机制、模块化接口以及良好的生态支持，尤其适合于状态-动作高度耦合的强化学习任务建模。本文设计的 DQN 网络结构包含一个主网络与一个目标网络，利用经验回放机制（Replay Buffer）提升样本利用率与收敛速度，结合 ϵ-动态调整策略控制探索与利用的权衡，从而增强策略训练的稳定性与泛化能力。同时，PyTorch 提供的 GPU 加速与高效张量计算接口，有效提升了大规模调度训练过程中的计算效率与资源利用率，为强化学习在复杂云平台任务分配奠定基础。

在 CloudSim 与 PyTorch 的交互机制方面，本文提出了一种系统集成方式：通过基于 Socket 的跨语言通信接口，实现 Java 环境下的 CloudSim 仿真平台与 Python 环境下的智能体策略决策模块之间的实时数据传输。该方案具备以下优势：（1）解耦性强，避免对中间件依赖，便于强化学习模块向其他仿真平台或实际调度系统的迁移部署；（2）通信高效，基于轻量级 Socket 协议实现状态-动作对的实时交换；（3）支持动态交互，能够在 CloudSim 模拟运行过程中动态捕捉当前系统状态，并及时传输给 DQN 智能体进行动作决策，再将调度结果反馈回仿真系统，实现任务逐步分配与环境同步演化；（4）可扩展性高，为未来构建多智能体通信等复杂策略提供可复用的交互机制。

实验结果表明，所提出的 DRL-TS 算法在多个关键性能指标上均优于对比方法。在 makespan 方面，智能体通过学习 VM 状态与任务特征之间的关系，能够避免资源堆积与不均衡分配问题，显著提升了整体任务完成速度；在负载均衡性方面，调度策略有效降低了部分 VM 的高负载现象，实现更均匀的资源使用；在任务响应时间上，DRL-TS 能动态调度至低等待 VM，从而减少排队时间与响应延迟；此外，在系统吞吐率与资源利用效率方面，该方法同样表现出稳定的优势与良好的适应性。上述实验充分验证了 DRL-TS 在动态、复杂的云环境中具备优异的调度性能。

本文工作的创新性主要体现在以下两个方面：（1）基于马尔可夫决策过程（MDP）提出了一种基于负载感知的状态表示机制，结合预测的任务执行时间与 VM 当前负载信息，实现对云平台状态的建模，提升了调度策略对动态环境的适应能力；（2）引入探索系数的动态衰减策略，使模型在训练初期具备充分的探索能力，能够挖掘潜在调度策略空间，而在训练后期逐步收敛至稳定策略选择，从而提高策略的收敛速度与决策精度。

综上所述，本文系统性地提出并实现了一种面向动态云环境的基于强化学习的任务调度优化框架。通过将强化学习与任务调度场景紧密结合，本文在方法建模、状态表达、奖励函数构建与训练策略等方面进行了系统创新，并在大规模仿真实验中验证了其优越性与实用性。该方法不仅在学术研究中为“云调度 + 智能优化”融合提供了可行范式，在工程实践中也展现出较强的部署与扩展潜力。未

来工作可在以下方向进一步拓展：一，引入多智能体协同学习机制，实现多任务并发下的全局最优策略寻优；二，结合迁移学习等，提升模型在多中心、多场景部署下的泛化与迁移能力；三，将调度策略与能耗优化、服务等级保障（SLA）等目标结合，拓展至绿色计算与服务质量保障等新兴研究领域。

关键词： 云计算, 负载均衡, 任务调度, 强化学习, 虚拟机资源管理

Abstract

Cloud computing has become the backbone of modern applications due to its elasticity, high availability, and pay-per-use model. However, efficiently managing task scheduling and load balancing remains a critical challenge, especially under dynamic workloads and heterogeneous resources. The task scheduling problem is NP-hard and becomes increasingly complex with large-scale concurrent tasks, where traditional static and heuristic methods struggle to adapt and optimize system performance.

To address these challenges, this paper proposes a reinforcement learning-based task scheduling algorithm (DRL-TS) using Deep Q-Networks (DQN). DRL-TS enables the agent to learn optimal scheduling policies through continuous interaction with the environment. Unlike rule-based methods such as Round Robin or Least Connections, the proposed model dynamically adapts to varying task demands and resource states to minimize makespan, reduce load imbalance, and improve resource utilization.

The task scheduling problem is formulated as a Markov Decision Process (MDP), where states capture VM load and predicted task execution times, actions represent task-to-VM assignments, and rewards combine makespan, cost, and load balancing objectives. A normalized reward function guides the agent’s learning. To stabilize training and improve convergence, experience replay and a target network are used, along with a linearly decaying exploration strategy.

Experiments are conducted on the CloudSim 7G simulation platform using diverse workloads (Random, GoCJ, Synthetic). DRL-TS is compared against static and heuristic baselines, showing significant improvements in makespan, VM utilization, and response time. The DQN is implemented in PyTorch. To bridge the Java-based CloudSim environment with the Python-based agent, a socket-based communication mechanism is introduced. This design supports real-time bidirectional data exchange, ensures system decoupling, and enables dynamic task scheduling at runtime.

Key contributions in this paper include: (1) a load-aware state representation based on MDP for system modeling; (2) a dynamic decaying exploration strategy for balanced learning.

In summary, DRL-TS offers an adaptive and intelligent scheduling framework for cloud environments, demonstrating strong performance across diverse workloads. The method lays a foundation for future work in multi-agent learning, transferability across cloud scenarios, and energy-aware scheduling optimization.

Key Words: Cloud Computing, Load Balancing, Task Scheduling, Reinforcement Learning, Virtual Machines Resource Optimization, Dynamic Workload Management

Contents

Chapter1 Introduction 1
1.1 Research background and significance 1
1.2 Literature review 2
1.2.1 SDN-based load balancing approaches 2
1.2.2 Machine learning-based load balancing methods 3
1.2.3 Multi-objective optimization methods 4
1.3 Main contributions of this work 4
1.4 Thesis organization 5
Chapter2 Background and fundamental 7
2.1 Cloud computing 7
2.2 Load balancing and task scheduling 9
2.3 Deep learning-based method 12
2.4 Chapter summary 13
Chapter3 Swarm intelligence optimization method 14
3.1 PSO-based method 14
3.2 Hybrid GA-GWO method 15
3.3 Chapter summary 16
Chapter4 Dynamic Reinforcement Learning-based method 17
4.1 Problem formulation 17
4.2 The objective definition 18
4.3 Problem modeling based on MDP 19
4.4 DQN-based architecture 22
4.5 Training process 23
4.6 Chapter summary 24
Chapter5 Experimental results 26
5.1 CloudSim framework 26
5.1.1 Architecture 27
5.1.2 Technology implementation 29
5.1.3 Simulation workflow 30
5.2 Evaluation metrics 33
5.3 Simulation and result analysis 34
5.3.1 Simulation environment 34
5.3.2 Benchmark datasets 35
5.3.3 Simulation result of proposed method 37
5.3.4 Comparasion between DRL-TS and SOTA algorithms 39
5.4 Chapter summary 41
Chapter6 Summary and prospect 43

Chapter1 Introduction

1.1 Research background and significance

Recently, cloud computing has emerged as a more prominent technology[1]. Cloud computing is the on-demand availability of computing resources such as memory, CPU, network, etc. Resource and data are shared across the network. All physical computing components have been virtualized and enable the user to access from remote locations. Cloud computing is the on-demand availability of computing resources such as memory, CPU, network, etc. The resources and data are shared across the network. All physical computing components have been virtualized and enable the user to access from remote locations.[2]. This resource-sharing paradigm not only significantly reduces the cost of hardware procurement and maintenance but also enhances resource utilization efficiency. Meanwhile, against the backdrop of the rapid advancement of artificial intelligence, especially given the massive computational demands of large-scale models, such workloads are typically deployed on cloud servers to provide efficient services to users. According to forecasts, by 2025, 95% of workloads are expected to be supported by cloud services[3-4]. Therefore, how to efficiently and intelligently allocate and schedule cloud computing resources under dynamic workload variations and complex application requirements has become one of the core challenges in cloud computing environments[5].

The high workload on virtual machines is one of the challenges of cloud computing in the allocation of virtual machines[6]. The task, requested by a client, has to wait to be allocated to the work and the resources needed.[7] Resource allocation technique is an important process to allocate resources based on the user' s application demands to achieve an optimal number of servers in use. This process is done dynamically for load balancing of non-preemptive tasks. Load balancing is an NP-hard optimization problem in cloud computing[8]. Traditional server load balancing methods, such as Round Robin, Least Connections, Source IP Hashing, etc., usually rely on static rules or server performance metrics to allocate loads. Although these approaches perform well under stable load conditions, they often struggle to accommodate rapidly changing demands, particularly in scenarios where the length of tasks fluctuates dramatically. So, in cloud computing platforms, the uncertainty of resource demands further exacerbates the limitations of traditional load balancing techniques, potentially leading to resource waste, server overload, or performance bottlenecks, impacting system responsiveness and user experience. Therefore, achieving flexible resource scheduling through intelligent methods in dynamic, complex, and uncertain environments has become crucial for improving cloud

platform performance, reducing operational costs, and optimizing resource utilization.

With the advancement of artificial intelligence and machine learning, an increasing number of load balancing solutions are incorporating machine learning and optimization algorithms for resource scheduling. For example, Seyedmajid Mousavi et al. proposed a hybrid algorithm based on Teaching-Learning-Based Optimization (TLBO) and Grey Wolf Optimization (GWO)[7], while G. Sulthana Begam et al. introduced an approach based on Multi-Regression Based Search (MRBS)[9]. These methods leverage large volumes of historical and real-time data to predict server load and make more intelligent scheduling decisions.

Reinforcement Learning (RL)[10], a class of machine learning approaches, enables agents to adapt to unknown environments by learning from feedback through environmental interaction, and has recently been applied to load balancing algorithms. Through reinforcement learning, an agent continuously refines its decision-making strategy by interacting with the environment, making it possible to adaptively learn how to distribute workloads among servers in highly dynamic cloud environments. Compared with traditional static scheduling methods, RL-based approaches can dynamically adjust resource allocation strategies based on real-time data and feedback, thereby optimizing load balancing on cloud platforms. This reduces dependency on manually designed rules and models, offering greater flexibility and scalability. By making exploratory adjustments to resource allocation and continuously evaluating the performance of current scheduling strategies, the agent optimizes future decisions using reward mechanisms.

Therefore, integrating reinforcement learning into server load balancing algorithms enables the use of adaptive learning capabilities and efficient policy optimization to address the limitations of traditional methods in handling dynamic workloads, complex environments, and multi-objective optimization. This research not only enhances resource utilization and system performance in cloud computing environments but also provides novel perspectives for intelligent cloud management and automated scheduling, contributing significantly to the advancement and adoption of cloud computing technologies.

1.2 Literature review

1.2.1 SDN-based load balancing approaches

Software-Defined Networking (SDN) is an emerging network architecture that decouples the control logic from underlying routers and switches, breaking the vertical integration between the control plane and the data plane that exists in traditional networks. The core idea of SDN is to separate the control functions from physical hardware, enabling logically centralized control and programmable network management[11-12]. Al-Mashhadi et al.[13]

explored the advantages of employing SDN for load balancing in cloud computing environments. They proposed both static and dynamic load balancing schemes and utilized OpenFlow and OpenStack to obtain service type, server load, and link health information, thereby improving decision-making and configuration for more efficient service distribution. Kang et al.[14] introduced an SDN-enhanced Inter-Cloud Manager (S-ICM) that performs load balancing by monitoring network latency and making corresponding decisions. Compared to Round Robin (RR) and the Honeybee Foraging Algorithm (HFA), their approach more effectively prevents system saturation under high-load conditions. Abdelltif et al.[15] proposed a method that maximizes resource utilization and minimizes user response time through service classification, dynamic load balancing, and a monitoring module. Liu et al.[16] presented a novel Service-Oriented Data Aggregation framework (SODA), which orchestrates data as services and aggregates data packets to reduce redundancy and service latency. Compared with traditional methods, SODA achieves better performance in terms of data redundancy reduction and response delay. Praveen et al.[17] proposed a method can automatically transfer the load from the heavy load controller to the light load controller depending on the load condition of each controller. Govindarajan et al.[18] proposed a Particle Swarm Optimization (PSO)-based load balancing technique. Simulation experiments using real-world application traces demonstrated that their PSO-based mechanism minimizes average response time while maximizing throughput, cloud consumer satisfaction, and resource utilization. Despite the efficiency of SDN-based load balancing approaches in resource allocation, they often entail considerable implementation complexity and require highly flexible network infrastructure.

1.2.2 Machine learning-based load balancing methods

The application of machine learning in load balancing enables dynamic adjustment of resource allocation strategies in a data-driven manner, allowing systems to adapt to complex and evolving environments. Many researchers have applied machine learning techniques to cloud computing for load balancing and resource scheduling. For example, Xu et al.[19] proposed a deep reinforcement learning (DRL)-based mobile load balancing (MLB) algorithm and a two-tier architecture to address large-scale load balancing issues in ultra-dense networks (UDN). Dittakavi et al.[20] utilized deep learning models to predict future CPU and memory consumption in cloud environments, enabling intelligent resource allocation and cost optimization while reducing the need for manual monitoring. Amanpreet Kaur et al.[21] introduced a deep learning-based framework (DLD-PLB) that optimizes virtual machine scheduling and load balancing to reduce workflow execution cost and latency in cloud computing, thereby improving system reliability and resource utilization. Shafiq et al.[22] leveraged both supervised and unsupervised learning to address challenges in load balancing, task scheduling, and resource optimization. Adil et al.[23] proposed a novel AI-

assisted hybrid load balancing scheduler (CA-MLBS) that combines machine learning and metaheuristic algorithms. They employed Support Vector Machines (SVM) for file-type classification of tasks and used Particle Swarm Optimization (PSO) to optimize task allocation in the cloud, leading to improved load balancing efficiency, reduced execution and response times, and enhanced throughput. Other methods also include some heuristic algorithms[24-27] and optimization algorithms[28-30]. Despite their performance benefits, machine learning-based methods often require large volumes of high-quality data, incur significant computational overhead, and suffer from limited interpretability.

1.2.3 Multi-objective optimization methods

Multi-objective optimization algorithms have been widely used in the context of load balancing in cloud computing. These methods aim to enhance overall system performance by simultaneously considering various optimization objectives such as task scheduling, resource utilization, and energy efficiency. Kruekaew et al.[31] proposed a multi-objective task scheduling optimization approach (MOABCQ) that integrates Artificial Bee Colony (ABC) and Q-learning algorithms to maximize virtual machine (VM) throughput and achieve load balancing in cloud environments, thereby improving execution efficiency under concurrent conditions. Hayyolalam et al.[32] developed a load balancing method combining chaos theory with the CBWO algorithm, achieving improvements of 67.28% and 29.03% in task completion time and energy consumption, respectively, compared to existing methods. Devaraj et al.[33] introduced a hybrid algorithm combining the Firefly Algorithm and an Improved Multi-Objective Particle Swarm Optimization (FIMPSO), which reduces the search space and optimizes response time to achieve more efficient resource utilization and faster task execution. Haris et al.[34] proposed a dynamic load balancing algorithm based on a hybrid optimization strategy (MMHHO), integrating Harris Hawks Optimization (HHO) with the Marine Predators Algorithm (MRFO). By considering multiple factors such as cost, response time, and resource utilization, their method improves VM throughput and overall load balancing performance. Although multi-objective optimization algorithms can significantly enhance load balancing effectiveness, they are often computationally expensive and may suffer from slow convergence and instability in dynamic environments, limiting their effectiveness in real-time resource allocation.

1.3 Main contributions of this work

This study proposes and implements a reinforcement learning-based load balancing approach for cloud computing servers. Traditional load balancing strategies often exhibit limitations when dealing with dynamic workloads and complex application demands, es-

pecially under the highly uncertain and fluctuating resource requirements typical in cloud computing environments. To address these challenges, this paper leverages the adaptive policy optimization capability of reinforcement learning to design a dynamic load balancing algorithm. Through continuous interaction with the environment, the system learns and optimizes resource allocation strategies automatically, achieving more flexible and efficient load balancing.

The paper begins by outlining the background and challenges of the load balancing problem and reviewing the characteristics of cloud platforms and current research developments in this area. Subsequently, a reinforcement learning-based load balancing scheme is designed, including a detailed explanation of the algorithm's principles, workflow, and training process. On this basis, an experimental environment is constructed, and the performance of different load balancing strategies is compared and analyzed.

1.4 Thesis organization

The structure of this thesis is organized into the following chapters, Figure 1.1 shows the technical route of this paper.

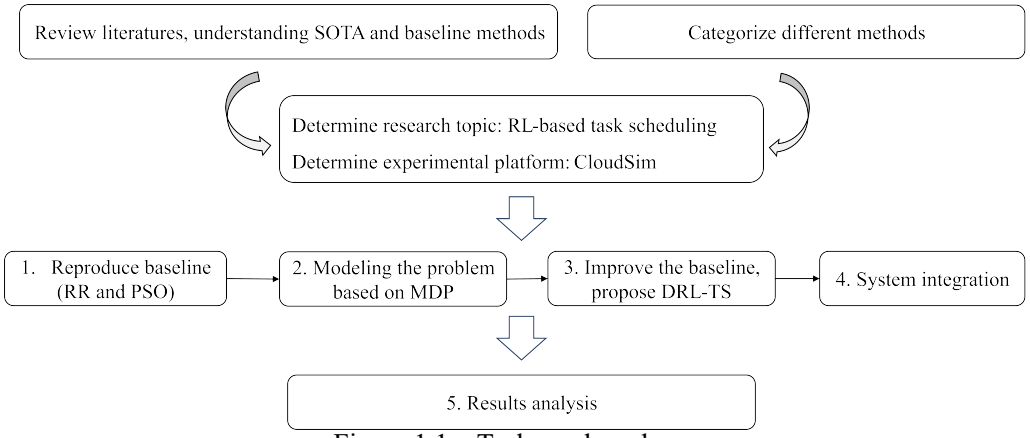


Figure 1.1 Technocal roadmap

Chapter 1: Introduction. This chapter introduces the significance and challenges of load balancing in the context of cloud computing. It analyzes the limitations of traditional load balancing approaches and highlights the potential and advantages of reinforcement learning in addressing dynamic load balancing problems. Furthermore, it briefly reviews recent research developments in related fields, providing the theoretical foundation for the proposed approach.

Chapter 2: Background and fundamental. This chapter introduces foundational concepts essential to understanding proposed method. It begins with an overview of cloud computing and its three core service models: SaaS, PaaS, and IaaS, illustrated through visual hierarchies and comparative tables. The chapter then emphasizes the importance of load balancing in cloud infrastructures and outlines key performance metrics. It distin-

guishes between static and dynamic load balancing strategies, highlighting the limitations of static methods in dynamic environments. Finally, the chapter introduces task scheduling as a critical component of load balancing and discusses reinforcement learning as a promising approach for adaptive, real-time decision-making.

Chapter 3: Swarm intelligence optimization method. This chapter reviews representative task scheduling strategies in cloud computing, focusing on three state-of-the-art approaches. First, a PSO-based scheduling method improves execution time and cost through a fitness-guided optimization model. Then, a Hybrid GA-GWO algorithm combines Genetic Algorithms and Grey Wolf Optimization to address energy, cost, and makespan objectives, using advanced energy modeling and constraint-based optimization. Experimental comparisons show each method's effectiveness under different scenarios, laying a strong foundation for designing learning-based schedulers.

Chapter 4: Dynamic Reinforcement Learning-based method. This chapter presents a Deep Q-Network-based task scheduling method tailored for dynamic cloud environments, called DRL-TS algorithm. It begins by formulating the problem and defining key objectives such as minimizing makespan and cost. The state, action, and reward structures are designed based on MDP to reflect real-time system conditions, with states encoding VM load and runtime estimates. A custom reward function is introduced to guide scheduling decisions through normalized makespan differences. The framework incorporates experience replay and a target network to ensure stable learning, along with a linearly decaying exploration rate ϵ to enhance policy optimization over time.

Chapter 5: Experimental result This section presents the experimental evaluation of the proposed reinforcement learning-based task scheduling approach for cloud environments. The experiments were conducted using CloudSim 7G, a widely adopted simulation framework for modeling and testing cloud computing systems. The experiments aimed to assess the effectiveness of the proposed Dynamic Reinforcement Learning-based task scheduling(DRL-TS) algorithm in comparison to baseline and state-of-the-art methods across various task volumes and VM configurations. The evaluation focused on key performance metrics, including makespan, total cost, utilization, and imbalance.

Chapter 6: Summary and prospect. This chapter summarizes the research contributions and key findings of the study, and discusses potential directions for future research.

Chapter2 Background and fundamental

This section provides an overview of cloud computing and its service models, including SaaS, PaaS, and IaaS, highlighting their differences through figures and tables. The section also discusses the importance of load balancing in optimizing cloud resources, comparing static and dynamic load balancing techniques and introducing performance metrics. Additionally, task scheduling is briefly covered, emphasizing its role in allocating tasks efficiently in dynamic cloud environments. Finally, this section introduces a DL-based method, which provides ideas for subsequent experiments.

2.1 Cloud computing

The basic definition of cloud computing is: A computing Cloud is a set of network enabled services, providing scalable, QoS guaranteed, normally personalized, inexpensive computing infrastructures on demand, which could be accessed in a simple and pervasive way[35]. So, users can acquire these services in a pay-per-use manner rather than investing in physical infrastructure. It significantly reduces the costs of deployment and maintenance. The underlying hardware is typically housed in data centers that employ advanced virtualization technologies, enabling high degrees of scalability, flexibility, and availability[36].

As a result, the development of enterprise systems on cloud computing platforms has become increasingly popular. The service models of cloud computing defined by the National Institute of Science and Technology (NIST) are[36]:

- **Software as a Service (SaaS)** refers to a cloud computing delivery model in which software applications are hosted by a service provider and made available to users over the internet. Users interact with the application through a web interface, while the underlying infrastructure, software maintenance, and updates are managed entirely by the provider.
- **Platform as a Service (PaaS)** provides a cloud-based environment equipped with tools and services that facilitate the development, testing, deployment, and scaling of applications. It abstracts away infrastructure management, allowing developers to focus on application logic and code without concern for the underlying hardware or system configuration.
- **Infrastructure as a Service (IaaS)** delivers fundamental computing resources—such as virtualized servers, storage, and networking—on a pay-as-you-go basis. Users retain control over operating systems and deployed applications, while the service provider manages the physical infrastructure and virtualization layer.

This paper use visualization of stacks in Figure 2.1 to describe the differences between infrastructure, platform, and software "-as-a-service" models. In the meanwhile, Table 2.1 shows the differences between the three types of services intuitively.

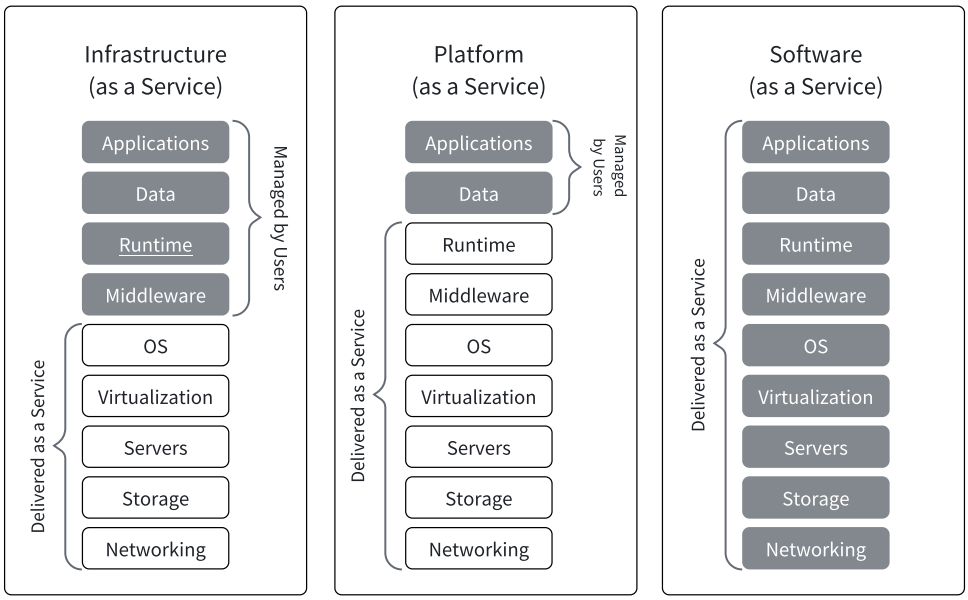


Figure 2.1 The difference between three service models

Table 2.1 Comparison of SaaS, PaaS, and IaaS

Aspect	SaaS	PaaS	IaaS
Primary Use	End-user software delivery	Application CI/CD	Virtualized computing infrastructure
User Responsibility	Application usage only	Application code and data	Application, OS, runtime, middleware
Provider Responsibility	Full stack	Infrastructure and development platform	Underlying infrastructure
User Access	Web browser or client	APIs or web-based environments	Command-line or APIs
Customization	Minimal	Moderate	Full control
Examples	Google Workspace, Zoom	Google App Engine, AWS Elastic Beanstalk	Amazon EC2, Microsoft Azure VMs
Target Audience	End-users	Developers	System administrators and IT professionals

These three types of cloud services exhibit a hierarchical structure, showing in Fig 2.2. Infrastructure as a Service (IaaS) lies at the bottom layer, sitting directly above the physical infrastructure of data centers. It virtualizes physical hardware resources into logical resource pools through virtualization technologies. Platform as a Service (PaaS) occupies the middle layer and is often referred to as the "cloud operating system." It provides an application runtime environment based on the underlying IaaS resource pool. Software as a Service (SaaS) is positioned at the top layer and can be developed and operated on top of the PaaS layer, or directly implemented using IaaS.

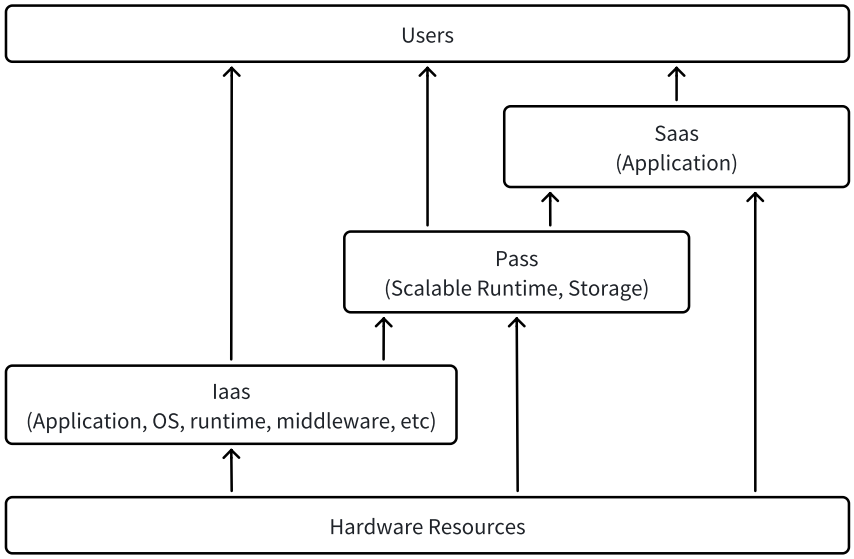


Figure 2.2 Cloud Computing Hierarchical Structure[37]

2.2 Load balancing and task scheduling

Load balancing is essential for optimizing the utilization of cloud computing resources, including processors, storage, and memory. Virtual machines (VMs). Without load balancing, workloads on each VMs may become uneven, leading to over-used and under-used of the physical resources. In the meanwhile, user demands are incredibly dynamic in cloud computing, and achieving multi-tenancy requires separating different users in the cloud infrastructure[38].

To solve this problem, load balancing techniques are required. Load balancing refers to the process of distributing workloads across multiple computing resources to ensure no single resource is overwhelmed, thereby maximizing throughput, minimizing response time, and achieving optimal resource utilization. In large-scale systems, improper load distribution can lead to bottlenecks, degraded performance, and even system failures. Load balancing techniques can be broadly categorized into static and dynamic approaches, depending on whether workload information is known a priori or changes dynamically during runtime. Algorithms such as Round Robin, Least Connections have been widely adopted to achieve efficient distribution. Load balancing is formulated as a multidimensional bin-packing problem which is considered as a NP-hard combinatorial optimization problem[39]. The main objective of load balancing methods is to speed up the execution of applications on resources whose workload varies at run time in unpredictable way[40].

The Figure 2.3 shows the basic model of load balancing architecture. The data center controller receives requests from users, using specific load balancing algorithms, and distributes the requests among virtual machines. User requests refer to tasks in this context,

the data center controller receives tasks and assigns them to the best virtual machine based on an algorithm. This process is called load balancing, enabling each virtual machine in this environment to achieve the best status, which is evaluated by the metrics as follow[41]:

- ***Throughtput***: This metric is used to calculate the number of processes completed per unit time.
- ***Response time***: It measures the total time that the system takes to serve a submitted task.
- ***Makespan***: This metric is used to calculate the maximum completion time or the time when the resources are allocated to a user.
- ***Scalability***: It is the ability of an algorithm to perform uniform load balancing in the system according to the requirements upon increasing the number of nodes. The preferred algorithm is highly scalable.
- ***Migration time***: The amount of time required to transfer a task from an overloaded node to an under-loaded one.
- ***Degree of imbalance***: This metric measures the imbalance among virtual machines.
- ***Performance***: It measures the system efficiency after performing a load-balancing algorithm.
- ***Energy consumption***: It calculates the amount of energy consumed by all nodes. Load balancing helps to avoid overheating and therefore reducing energy usage by balancing the load across all the nodes.
- ***Carbon emission***: It calculates the amount of carbon produced by all resources. Load balancing has a key role in minimizing this metric by moving loads from underloaded nodes and shutting them down.

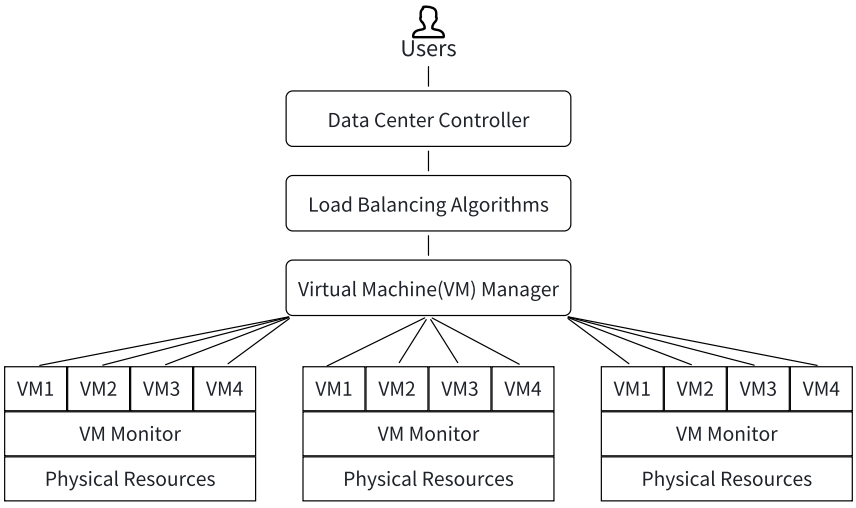


Figure 2.3 The Model of Load Balancing Architecture in Cloud Computing[41]

Virtualization is a crucial technique in cloud computing which is shown at the bottom of the model in Figure 2.3. The main objective of virtualization is sharing expensive

hardware among VMs[41]. In the meanwhile, cloud computing depends highly on Virtual Machine Monitor (VMM) or hypervisors. Hypervisors help operate multiple VMs in the same hardware layer[31]. VMs handle user requests, which are generated randomly by users located across the globe. These requests must be effectively assigned to VMs for processing, making task assignment a critical challenge in cloud computing. When some VMs are overloaded while others remain idle or underutilized, the quality of service (QoS) degrades[42]. As QoS diminishes, user satisfaction declines, potentially leading to user attrition and reduced system retention.

There are basically two categories of load balancing: **Static(i)** and **Dynamic(ii)**[40]:

Static load balancing methods rely on predetermined strategies to distribute the workload across resources, without considering real-time changes in the system's state. These approaches typically assume that the system is homogeneous and that workloads can be distributed evenly based on available resource specifications. Static methods are advantageous in systems where the workload is predictable or follows a repetitive pattern, as they simplify implementation and reduce computational overhead. Examples of static load balancing techniques include Round Robin, Least Connections. However, these methods face significant challenges when dealing with fluctuating workloads or varying task sizes, since they are difficult to adapt to real-time conditions, potentially leading to inefficiencies or even bottlenecks in the system. The MOABCQ algorithm[31] proposed by Kruekaew et al., CBWO algorithm[32] proposed by Hayyolalam et al. and MMHHO algorithm[34] proposed by Haris et al. belong to static load balancing methods.

In contrast, dynamic load balancing methods make real-time decisions based on the current state of the system, adapting to fluctuations in workload, resource availability, and task execution times. For example, the RL-based method[43] proposed by Khan et al. belong to dynamic load balancing technique. Dynamic techniques are particularly well-suited for environments where the workload is unpredictable, such as in cloud computing or distributed systems, where tasks and resources can vary significantly over time. These methods monitor the system's state continuously, adjusting the workload distribution accordingly. Popular dynamic load balancing algorithms include Least Load, which assigns tasks to the resource with the least load, and Feedback-based methods, which dynamically redistribute tasks based on real-time performance metrics, such as CPU utilization or network throughput. While dynamic approaches are more complex and resource-intensive compared to static methods, they offer greater flexibility and efficiency in handling unpredictable or highly variable workloads, significantly improving system performance and scalability in diverse environments.

In the same manner, task scheduling focuses on determining the order and allocation of computational tasks to available resources over time, aiming to optimize specific objectives such as minimizing the makespan, meeting task deadlines, or balancing resource usage. In cloud computing, task scheduling is especially critical due to the heterogeneous

and scalable nature of resources, where the system must dynamically adapt to changing workload patterns and resource availability. Various heuristic, metaheuristic, and reinforcement learning-based strategies have been explored for effective task scheduling in dynamic environments.

2.3 Deep learning-based method

Kaur et al. (2020) proposed a novel Deep Learning-based Deadline-constrained, Dynamic VM Provisioning, and Load Balancing (DLD-PLB) framework[21]. The framework utilizes Convolutional Neural Networks (CNNs) for task scheduling, where the input tasks are represented by their computation time and cost, and the output is an optimal schedule that minimizes both makespan and cost. The CNN model consists of multiple layers, including convolution, pooling, and ReLU activation functions, to extract and process task features.

The DLD-PLB framework optimizes resource allocation by employing gradient descent optimization to tune task weights (time and cost) and predict an optimal schedule. This deep learning approach outperforms traditional hybrid-heuristic methods by significantly reducing makespan and resource wastage. The experimental results show that the DLD-PLB framework provides better performance than previous models based on hybrid heuristic and metaheuristic approaches, such as the Hybrid PEFT-ACO[21].

The objective function in the proposed DLD-PLB framework is to minimize two key parameters: makespan and cost. The system aims to balance the load across available VMs while optimizing these parameters. The cost function is defined as:

$$f(q_j) = a_1 \cdot k_{iq_j} \cdot EET^h_{iq_j} + a_2 \cdot k_{iq_j} \cdot EEC^h_{iq_j} \tag{2.1}$$

where a_1 and a_2 are weighting factors for execution time (EET) and execution cost (EEC), respectively. The parameter k_{iq_j} represents the allocated resources for task q_j, and $EET^h_{iq_j}$ and $EEC^h_{iq_j}$ are the execution time and cost for task q_j on VM h. The deep learning model iteratively updates these parameters using gradient descent optimization.

The performance of the DLD-PLB framework was evaluated through simulations conducted on a Cloud Workflow Simulator (CWS) with varying numbers of VMs (2, 4, 8, 16, and 32). The results were compared to those obtained from the Hybrid PEFT-ACO model. The makespan and cost comparisons between the two models are summarized in Tables 2.2.

Table 2.2 Comparison between DLD-PLB and Hybrid PEFT-ACO on Makespan and Cost

Metrics	Number of VMs	DLD-PLB	Hybrid PEFT-ACO
Makespan (ms)	2	311	6426.21
	4	165	19246.99
	8	102	33886.77
	16	81	61839.59
	32	79	259757
Cost (Rs.)	2	32.655	31.72
	4	34.65	71.66
	8	42.84	373.42
	16	68.04	706.34
	32	132.72	1266

2.4 Chapter summary

This chapter presents a comprehensive overview of cloud computing, focusing on its service models and resource management strategies. It begins by introducing the three primary service models—Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS)—highlighting their hierarchical relationships, functional responsibilities, and typical use cases through illustrative figures and tabulated comparisons. The discussion then transitions to the critical role of load balancing in cloud environments, where the dynamic and heterogeneous nature of virtualized resources necessitates efficient workload distribution to ensure optimal system performance and resource utilization. Both static and dynamic load balancing approaches are analyzed, with emphasis on their underlying principles, typical algorithms, and respective strengths and limitations. Performance evaluation metrics such as throughput, makespan, response time, energy consumption, and carbon emission are introduced to assess the effectiveness of different balancing strategies. Then, this chapter also touches on task scheduling as a complementary mechanism to load balancing, underlining its significance in mapping computational tasks to appropriate virtual machines in a time- and resource-efficient manner. This chapter, in the last, introduces a existing DL-based method, which is a novel load balancing method. This method provides ideas for subsequent experiments. Collectively, these foundational concepts establish the technical background necessary for developing intelligent and adaptive cloud resource management frameworks in subsequent sections.

Chapter3 Swarm intelligence optimization method

This section reviews some SOTA task scheduling methods in cloud computing. Kumar and Sharma[44] proposed a PSO-based scheduling technique and introduced the PSO-BOOST algorithm to improve task allocation by optimizing processing time, cost, and throughput. Behera and Sobhanayak (2024)[45] introduced a Hybrid GA-GWO method that integrates Genetic Algorithm and Grey Wolf Optimization to optimize makespan, energy consumption, and cost, showing superior performance over other metaheuristic algorithms.

3.1 PSO-based method

Kumar and Sharma proposed a novel Particle Swarm Optimization (PSO)-based resource scheduling technique[44,46] designed to improve key Quality of Service (QoS) parameters in cloud computing environments, including processing time, execution cost, and throughput. The main objective of their work was to optimize resource allocation by reducing processing time and financial costs while fulfilling users' service-level agreements (SLA). The authors modeled the problem as a multi-objective optimization challenge, where the task was to balance the trade-off between execution time and cost. To solve this, they introduced a fitness function that combines both objectives with user-defined weights, allowing the cloud service provider to dynamically adjust resource allocation based on user priorities .

The fitness function used in the paper is defined as follows:

$$f(q_j) = a_1 \cdot k_{iq_j} \cdot EET^h_{iq_j} + a_2 \cdot k_{iq_j} \cdot EEC^h_{iq_j} \tag{3.1}$$

where a_1 and a_2 represent the weight factors for execution time and execution cost, respectively. These weights are chosen according to user preferences, with $a_1 + a_2 = 1$. The decision variable k_{iq_j} is a binary value indicating whether a task is assigned to a resource or not.

They also introduce the PSO-BOOST algorithm, a modification of standard PSO, aimed at improving the exploration and exploitation abilities of the search process. The algorithm iteratively updates particle velocities and positions based on local and global best values, which helps in mapping tasks to the most suitable virtual machines (VMs) while minimizing execution time and cost. The algorithm's performance is evaluated through a series of simulation experiments conducted using CloudSim, which is used to model cloud computing environments with varying workloads and VM configurations.

The experimental setup involved the simulation of various QoS parameters under different conditions, with results demonstrating that PSO-BOOST outperforms existing algorithms such as PSO, APSO, artificial bee colony (ABC), and BAT in processing time, makespan time, and throughput. Specifically, the PSO-BOOST algorithm was able to process more tasks within the user-specified deadlines, leading to higher task acceptance ratios and lower SLA violations. Additionally, the algorithm showed improvements of up to 14% in processing time and reduced execution costs compared to the baseline methods.

3.2 Hybrid GA-GWO method

The work by Behera and Sobhanayak (2024)[45] introduces a novel hybrid method combining Grey Wolf Optimization (GWO) and Genetic Algorithm (GA) for multi-objective task scheduling in heterogeneous cloud environments. The proposed Hybrid GA-GWO algorithm addresses the limitations of standard GWO, such as premature convergence, by integrating the crossover and mutation operations from GA. These genetic operators enhance the algorithm's ability to explore and exploit the search space, ensuring a balance between exploration and exploitation.

The hybrid algorithm optimizes multiple objectives simultaneously, including makespan, energy consumption, and computational cost. The task scheduling problem in this context is formulated as follows:

$$\text{Minimize } F(x) = (\chi(x),\, \zeta(x),\, \epsilon(x)) \tag{3.2}$$

where $\chi(x)$ represents makespan, $\zeta(x)$ denotes computational cost, and $\epsilon(x)$ represents energy consumption. The goal is to allocate tasks efficiently to physical machines while minimizing these objectives, subject to constraints on available resources such as CPU, memory, and bandwidth. The problem is constrained as follows:

$$h_i(x) \leq \hat{h}(x,i), \quad g_i(x) \leq \hat{g}(x,i), \quad q_i(x) \leq \hat{q}(x,i) \tag{3.3}$$

The objective function used in the hybrid GA-GWO algorithm aims to minimize three key factors: makespan, cost, and energy consumption. The fitness function is defined as:

$$\text{Fitness} = \gamma_1 \cdot \epsilon + \gamma_2 \cdot \chi + \gamma_3 \cdot \zeta \tag{3.4}$$

where $\gamma_1, \gamma_2, \gamma_3$ are the weight parameters for energy, makespan, and cost, respectively. The energy consumption is calculated using a power model considering both static and dynamic power components:

$$P_t = P_s + P_d \tag{3.5}$$

$$P_d = \alpha_i \cdot (f_d)^3 \tag{3.6}$$

where P_s is the static power, P_d is the dynamic power, and f_d is the CPU frequency. The energy consumed by a physical machine (PM) is calculated as:

$$E_t = \int_{st}^{ft} \left(\beta_i \cdot P_{\max} \cdot S_t + (1 - \beta_i) \cdot P_{\max} \cdot (f_{\max})^3 \cdot (f_d)^3\right) \, dt \tag{3.7}$$

The proposed hybrid GA-GWO algorithm was evaluated using the CloudSim toolkit, which simulates cloud data centers with virtual machines and tasks. Experiments were conducted on synthetic datasets and real-world datasets from NASA Ames and HPC2N. The evaluation metrics included Normalized Schedule Length (NSL), total energy consumption, and speedup.

The algorithm was compared against other metaheuristic algorithms, including GWO, GA, and PSO. The experimental results showed that the Hybrid GA-GWO algorithm outperformed the other algorithms in terms of NSL, energy consumption, and speedup. Specifically, the proposed method achieved reductions of 19%, 21%, and 15% in makespan, cost, and energy consumption, respectively, compared to the GWO, GA, and PSO approaches.

Table 3.1 Performance Comparison of the Proposed Algorithm with GWO, GA, and PSO

Algorithm	Makespan Reduction	Cost Reduction	Energy Reduction
GWO	19%	13%	17%
GA	21%	17%	19%
PSO	15%	22%	23%

3.3 Chapter summary

This chapter reviews SOTA task scheduling methods in cloud computing, focusing on PSO-based and hybrid metaheuristic methods. The PSO-BOOST algorithm, proposed by Kumar and Sharma, improves upon standard PSO by integrating enhanced exploration and exploitation mechanisms to optimize execution time and cost. It demonstrates superior performance in minimizing makespan and improving throughput compared to traditional algorithms. Then, Behera and Sobhanayak presented a hybrid GA-GWO algorithm to avoid premature convergence and enhance global search capability. This method targets multi-objective optimization, minimizing makespan, energy consumption, and cost, and shows significant improvements in all metrics over standard GWO, GA, and PSO. Collectively, these studies demonstrate the efficacy of integrating heuristic and learning-based strategies for dynamic and efficient task scheduling in heterogeneous cloud environments, paving the way for more adaptive and intelligent resource management frameworks.

Chapter4 Dynamic Reinforcement Learning-based method

This chapter presents a Dynamic Reinforcement Learning-based task scheduling (DRL-TS) method to solve the task scheduling problem in cloud computing environments. This chapter describe how to represent the system state using VM load and task runtime estimates, define the action space, and design a reward function that guides the learning process based on makespan optimization.

To support learning in dynamic and uncertain environments, this paper also incorporate experience replay and a target network into the DQN training pipeline. This paper also propose a linear decay strategy for the exploration parameter ϵ, enabling the agent to gradually shift from exploration to exploitation.

4.1 Problem formulation

Task scheduling process plays an important role in cloud computing environment. A good task scheduling algorithm can reduce makespan, latency, response time significantly. The task scheduling problem in this paper is defined as follows[31]:

Input:

- Let $V = \{v_1, v_2, v_3, \ldots, v_m\}$, where V is a collection of Virtual Machines (VMs) and m is the total number of VMs in the cloud network. Each VM has its own resources (e.g., CPU, RAM, and bandwidth), and the cost of usage and computing power are defined differently. v_i represents the i-th VM.
- Let $T = \{t_1, t_2, t_3, \ldots, t_n\}$, where T is the set of assigned tasks, and n is the number of independent tasks performed on the VMs. t_j represents the j-th task. Each task submission specifies the number of instructions, required memory, required CPU, etc[47].

Output: Optimize task scheduling by mapping n tasks to m VMs (resources) to achieve the following objectives[48-49]:

- Minimize makespan,
- Minimize execution cost,
- Maximize resources utilization(Minimize machine's idle time),
- Increase load balancing across VMs.

4.2 The objective definition

The first objective is defined in the condition of makespan or task execution time, which is the time that the system completes its last task[31]. Each virtual machine (VM) exhibits a distinct execution time for completing assigned tasks, as reflected in the makespan. A high maximum execution time corresponds to an elevated makespan, indicating an inefficient distribution of tasks across VMs. Conversely, a low maximum execution time results in a reduced makespan, suggesting that the system has effectively balanced the task allocation among available resources.

For a given task $t_j \in T$, where T denotes the set of all tasks, and a given virtual machine $v_i \in V$, where V denotes the set of available virtual machines, the execution time (ET) of task t_j on machine v_i is defined as:

$$\text{ET}(t_j, v_i) = \frac{\text{length}(t_j)}{\text{MIPS}(v_i)} \tag{4.1}$$

where $\text{ET}(t_j, v_i)$ represents the expected time required to complete task t_j when assigned to virtual machine v_i. Here, $\text{length}(t_j)$ denotes the number of instructions contained in task t_j, measured in units of million instructions. Similarly, $\text{MIPS}(v_j)$ represents the processing capability of virtual machine v_j, defined as the number of million instructions that can be executed per second.

The execution time matrix for all task-VM pairs can thus be expressed as:

$$\mathbf{ET} = [\text{ET}(t_j, v_i)]_{j=1,\ldots,|T|}^{i=1,\ldots,|V|} \tag{4.2}$$

where $|T|$ and $|V|$ denote the cardinalities of the task set and VM set, respectively.

The second objective is defined in terms of cost which is the cost of requesting to be processed by the task and can be calculated from CPU cost, memory usage cost, and bandwidth usage cost[31]. For a given task $t_j \in T$ and a virtual machine $v_i \in V$, the execution cost (Cost) associated with executing task t_j on machine v_i is defined as:

$$\text{Cost}(t_j, v_i) = (c_1(v_i) + c_2(v_i) + c_3(v_i)) \times \text{ET}(t_j, v_i) \tag{4.3}$$

where:

- $c_1(v_i)$ denotes the unit CPU cost of virtual machine v_i,
- $c_2(v_i)$ denotes the unit memory cost of virtual machine v_i,
- $c_3(v_i)$ denotes the unit bandwidth cost of virtual machine v_i,
- $\text{ET}(t_j, v_i)$ denotes the execution time of task t_j on machine v_i, as previously defined.

Thus, the total cost is proportional to both the resource cost characteristics of v_i and the task execution time on that machine.

4.3 Problem modeling based on MDP

In the reinforcement learning framework, the task scheduling problem can be formulated as a Markov Decision Process (MDP)[50], which consists of a state space, an action space, a state transition function, and a reward function. At each time step, the agent observes the current system state and selects an action, corresponding to assigning the incoming task to a specific virtual machine. The environment then transitions to a new state and provides an immediate reward that reflects the quality of the scheduling decision. This formulation is well-suited for capturing the dynamic and uncertain nature of resource allocation in cloud computing environments.

In the meanwhile, to effectively address the dynamic and complex nature of cloud computing environments, this paper adapt the Deep Q-Network (DQN) framework proposed by Mnih *et al.*[10] to the task scheduling problem. DQN combines reinforcement learning with deep neural networks to learn control policies directly from high-dimensional inputs, demonstrating strong generalization and stability properties even under sparse and delayed reward signals. Reinforcement learning agent-environment interaction in MDP can be represented in Figure 4.1.

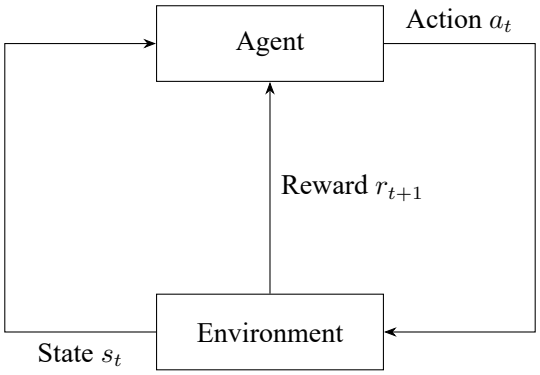


Figure 4.1 Reinforcement learning agent-environment interaction in MDP

In the task scheduling framework, this paper define the state, action, and reward as follows:

State representation At each decision step, the system state $s \in \mathbb{R}^{2N}$ is composed of two parts:

- $\mathbf{l} = (l_1, l_2, \ldots, l_N)$: the current load on each virtual machine (VM), where l_i denotes the remaining execution time on VM i.
- $\mathbf{r} = (r_1, r_2, \ldots, r_N)$: the estimated runtime of the incoming task if assigned to each VM, where r_i denotes the estimated runtime on VM i.

Thus, the state vector in time t is:

$$s_t = [l_1, l_2, \ldots, l_N,\ r_1, r_2, \ldots, r_N] \tag{4.4}$$

Since the load of each virtual machine is proportional to the time required to complete all tasks in its queue, this paper further refines the definition of the load l_i for each $v_i \in V$.

Specifically, for virtual machine v_i denoted as:

- $v_i^{\text{in\_process}} = \{t_1, t_2, \ldots, t_x\}$: the set of tasks currently executing on v_i;
- $v_i^{\text{awaiting}} = \{t_1, t_2, \ldots, t_y\}$: the set of tasks waiting to be executed on v_i;

where $x, y \in \mathbb{N}$ and $x, y < |T|$, with T representing the total set of tasks.

Then, the load l_i of virtual machine v_i can be formally defined as:

$$l_i = \sum_{a=1}^{x} \text{ET}(t_a, v_i) + \sum_{b=1}^{y} \text{ET}(t_b, v_i) \tag{4.5}$$

where $\text{ET}(t, v_i)$ denotes the execution time of task t on v_i.

According to Equation 4.1, it can substitute into the load definition yields:

$$l_i = \frac{1}{\text{MIPS}(v_i)} \left(\sum_{a=1}^{x} \text{length}(t_a) + \sum_{b=1}^{y} \text{length}(t_b) \right) \tag{4.6}$$

Thus, the load of each virtual machine is determined by the sum of the lengths of the tasks currently being executed and waiting to be executed, normalized by the computational capability of the machine.

For each $r_i \in \mathbf{r}$, where $\mathbf{r}$ denotes the set of predicted execution times for the incoming task on all available virtual machines, this paper define, according to Equation 4.1, the predicted execution time of task t_j on virtual machine v_i as:

$$r_i = \text{ET}(t_j, v_i) \tag{4.7}$$

The process of building a state s_t at time t is described in Figure 4.2.

Action space An action $a_t \in \{1, 2, \ldots, N\}$ corresponds to assigning the current task to the selected VM a.

Reward design After taking action a, the load of the selected virtual machine v_a is updated as follows:

$$l'_a = l_a + r_a \tag{4.8}$$

The updated load vector becomes $\mathbf{l}' = [l_1, l_2, \ldots, l'_a, \ldots, l_N]$. Consequently, the next state after taking action a is given by:

$$s_{t+1} = [\mathbf{l}', \mathbf{r}] = [l_1, l_2, \ldots, l'_a, \ldots, l_N,\ r_1, r_2, \ldots, r_N] \tag{4.9}$$

The predicted makespan after taking action a is computed as:

$$\text{makespan}_a = \max(\mathbf{l}') \tag{4.10}$$

where makespan_a denotes the completion time of the latest finishing task after the update.

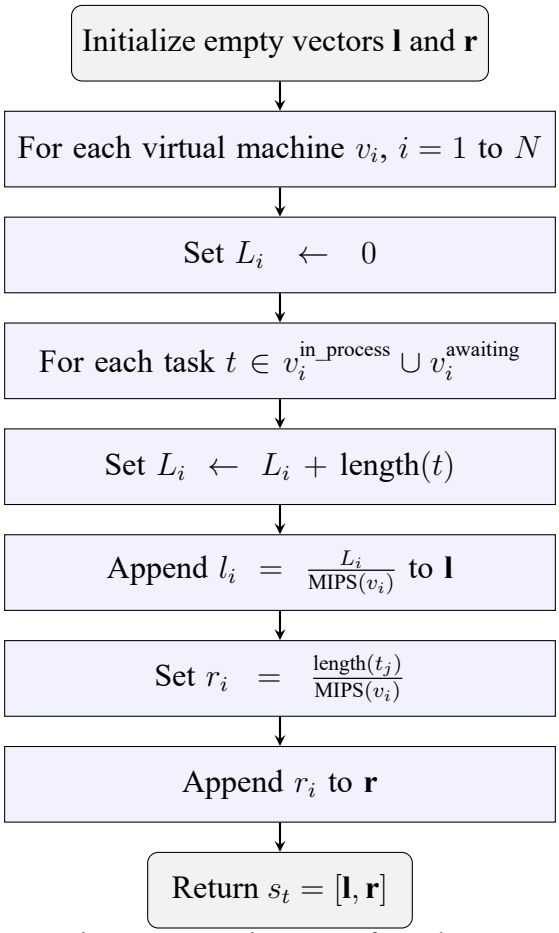


Figure 4.2 State construction process at time step t for task t_j across N virtual machines.

To encourage makespan optimization, this paper evaluate the impact of assigning the task to each VM i by simulating all possible actions and calculating the resulting makespan:

$$\textbf{all\_situations} = [\text{makespan}_1, \text{makespan}_2, \ldots, \text{makespan}_N] \tag{4.11}$$

where makespan_i is the makespan if task t_j were assigned to VM v_i.

Assuming the agent selects action a, this paper computes the difference between the makespan for each possible action and that of action a:

$$\boldsymbol{\Delta} = \textbf{all\_situations} - \text{makespan}_a = [\text{makespan}_1 - \text{makespan}_a, \ldots, \text{makespan}_N - \text{makespan}_a] \tag{4.12}$$

A negative value Δ_i in $\boldsymbol{\Delta}$ indicates that action i is better than action a, while a positive value implies it is worse. This paper defines the bounds for normalization as:

$$\text{best\_makespan} = \min(\textbf{all\_situations}) \tag{4.13}$$

$$\text{worst\_makespan} = \max(\textbf{all\_situations}) \tag{4.14}$$

To quantify the quality of the selected action a, this paper normalizes the makespan

differences using min-max scaling to map the value into the range $[-1, 1]$:

$$\mathbf{\Delta}^{\text{norm}} = 2 \times \frac{\Delta_i - \text{best\_makespan}}{\text{worst\_makespan} - \text{best\_makespan}} - 1 \tag{4.15}$$

The closer Δ_a^{norm} is to -1, the closer action a is to the best action. Conversely, values closer to 1 indicate a worse decision.

Finally, the reward for taking action a is defined as:

$$r_t = -\alpha \cdot \Delta_a^{\text{norm}} \tag{4.16}$$

where $\alpha > 0$ is a weighting factor that controls the magnitude of the reward signal.

This reward formulation incentivizes the agent to minimize the predicted makespan increase and promotes a balanced distribution of tasks across the available VMs. The pseudo code of reward calculation is shown in Algorithm 1.

Algorithm 1 Reward Calculation

Require: Current state $s_t = [\mathbf{l}, \mathbf{r}]$, selected action a, number of VMs N, weighting factor α

Ensure: Reward value r_t

$\mathbf{l}' \leftarrow \mathbf{l}$ ▷ Copy current VM loads
$l'_a \leftarrow l_a + r_a$ ▷ Update load of selected VM
$s_{t+1} \leftarrow [\mathbf{l}', \mathbf{r}]$ ▷ Form next state
makespan$_a \leftarrow \max(\mathbf{l}')$
Initialize **all_situations** $\leftarrow []$
for $i = 1$ to N **do**
    $\mathbf{l}^{(\mathbf{i})} \leftarrow \mathbf{l}$ ▷ Copy original loads
    $l_i^{(i)} \leftarrow l_i + r_i$ ▷ Simulate task assignment to VM i
    Append $\max(\mathbf{l}^{(\mathbf{i})})$ to **all_situations**
end for
$\mathbf{\Delta} \leftarrow$ **all_situations** $-$ makespan$_a$
best_makespan $\leftarrow \min($**all_situations**$)$
worst_makespan $\leftarrow \max($**all_situations**$)$
$\mathbf{\Delta}^{\text{norm}} \leftarrow 2 \cdot \frac{\mathbf{\Delta} - \text{best\_makespan}}{\text{worst\_makespan} - \text{best\_makespan}} - 1$
$r_t \leftarrow -\alpha \cdot \Delta_a^{\text{norm}}$
return r_t

4.4 DQN-based architecture

DQN approximates the optimal action-value function $Q^*(s, a)$, which represents the maximum expected return achievable by taking action a in state s and following the optimal

policy thereafter:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}[R_t \mid s_t = s, a_t = a, \pi] \tag{4.17}$$

The fundamental Bellman equation characterizing $Q^*(s,a)$ is:

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a \right] \tag{4.18}$$

To approximate $Q^*(s,a)$, a neural network parameterized by θ is trained to minimize the temporal difference loss at iteration i:

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[(y_i - Q(s,a;\theta_i))^2 \right] \tag{4.19}$$

where the target y_i is defined as:

$$y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) \mid s,a \right] \tag{4.20}$$

The gradient of the loss function with respect to network parameters is computed as:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot), s' \sim \mathcal{E}} \left[\left(r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i) \right) \nabla_{\theta_i} Q(s,a;\theta_i) \right] \tag{4.21}$$

To address the issues of correlated data and non-stationary distributions inherent in reinforcement learning, DQN employs *experience replay*[10]. The agent stores past transitions $e_t = (s_t, a_t, r_t, s'_{t+1})$ into a replay buffer $\mathcal{D}$ and samples random minibatches during training. This technique stabilizes learning by smoothing over changes in the data distribution and reducing variance.

4.5 Training process

To balance exploration and exploitation, this paper linearly decay the exploration rate ϵ from an initial value ϵ_{start} to a minimum value ϵ_{end} over the course of training:

$$\epsilon_t = \max \left(\epsilon_{\text{end}}, \epsilon_{\text{start}} - (\epsilon_{\text{start}} - \epsilon_{\text{end}}) \times \frac{\text{train\_count}}{\text{max\_iterations}} \right) \tag{4.22}$$

This ensures sufficient exploration in the early training phase and more deterministic decision-making as training progresses. The full algorithm is presented in Algorithm 2[10].

Algorithm 2 Deep Q-Network with Experience Replay for Cloud Task Scheduling[10]

Initialize Q-network $Q(s, a; \theta)$ and target network $Q'(s, a; \theta^-)$ with random weights
Initialize replay buffer $\mathcal{D} \leftarrow \emptyset$
for each training step $t = 1$ to E **do**
    Receive current state s_t from environment
    Form full state $s_t = [\mathbf{l}, \mathbf{r}]$
    Compute exploration rate ϵ_t by linear decay based on Equation 4.22
    Select a random number $random \in [0, 1]$
    if $random < \epsilon$ **then**
        Select random action a_t
    else
        Select greedy action $a_t = \arg\max_a Q(s_t, a; \theta)$
    end if
    Execute a_t and observe reward r_t and next state s_{t+1}
    Store transition $(s_t, a_t, r_t, s_{t+1}, \text{done} = \text{False})$ into $\mathcal{D}$
    if $|\mathcal{D}| \geq$ batch size **then**
        Sample minibatch from $\mathcal{D}$
        for each sample $(s, a, r, s', \text{done})$ **do**
            if done **then**
                $y \leftarrow r$
            else
                $y \leftarrow r + \gamma \cdot \max_{a'} Q'(s', a'; \theta^-)$
            end if
            Compute loss $L(\theta) = \frac{1}{N} \sum (y - Q(s, a; \theta))^2$
            Perform gradient descent on $L(\theta)$
        end for
    end if
end for

4.6 Chapter summary

This chapter presents a Deep Q-network-based task scheduling framework designed for dynamic and heterogeneous cloud environments. This paper formulated task scheduling as a Markov Decision Process, where an agent learns to make scheduling decisions based on real-time system states. The system state incorporates both the current VM load and the estimated runtime of the incoming task on each VM.

Each scheduling decision is treated as an action, and the agent receives feedback through a reward function centered on minimizing makespan. By simulating the outcomes

of all possible task-VM assignments, this paper compute normalized reward values that reflect the relative quality of each decision. A linear decay strategy is applied to the exploration parameter ϵ, enabling the model to gradually shift from exploration to exploitation over time.

To stabilize training, this paper uses a target network and experience replay. The Q-network is updated through temporal-difference learning, and the learning process is summarized in pseudocode to provide a clear understanding of the implementation flow.

The main contributions are: (1) introducing a dynamic, feedback-driven scheduling mechanism, and (2) designing a flexible exploration strategy via adaptive ϵ decay to improve learning efficiency in complex cloud environments.

Chapter5 Experimental results

This section introduces *CloudSim*, the simulation platform employed to model and evaluate the task scheduling process in a controlled cloud computing environment. This paper details the experimental setup and analyze the performance of the proposed method under varying configurations, including different numbers of virtual machines (VMs), task volumes, and datasets. Additionally, this paper defines a set of evaluation metrics to compare the proposed approach with baseline and state-of-the-art methods.

5.1 CloudSim framework

CloudSim is a widely adopted simulation toolkit specifically designed for modeling and simulating cloud computing infrastructures and services, which has been developed by the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne. The framework provides a comprehensive and extensible platform for evaluating the performance of cloud-based environments under various configurations, enabling researchers and developers to test and validate novel resource allocation and scheduling strategies in a controlled and repeatable manner.

This paper used CloudSim 7G[51] as experimental framework, which is the newest version of CloudSim released by CLOUDS. CloudSim 7G represents a major advancement in the modeling and simulation of cloud computing systems. It introduces a re-engineered and extensible architecture designed to overcome limitations of previous CloudSim versions and to support the complexities of next-generation cloud environments. The significance and role of CloudSim 7G can be summarized as follows:

- **Cost-Efficient Experimentation:** By simulating complex cloud infrastructures and scheduling policies, CloudSim 7G eliminates the need for expensive real-world deployment, significantly reducing experimentation costs.
- **Support for Realistic Scenarios:** It enables modeling of diverse scheduling environments, including nested virtualization, power-aware scheduling, and network-aware workflows, providing high-fidelity simulations.
- **Flexible and Extensible Architecture:** CloudSim 7G introduces standardized Java interfaces such as `CloudletScheduler`, `HostEntity`, `GuestEntity`, which can be easily implemented and tested custom scheduling strategies.
- **Improved Performance and Scalability:** Through extensive refactoring and optimization, CloudSim 7G achieves reduced memory consumption and faster simulation times, supporting large-scale scheduling experiments with improved efficiency.
- **Reproducible and Controlled Testing:** It provides a repeatable simulation envi-

ronment to evaluate the performance of task scheduling algorithms under varying workloads and system configurations.

- **Integrated Multi-Module Support:** CloudSim 7G enables the coexistence of multiple modules within a single scenario, allowing comparative studies of task scheduling across paradigms.

5.1.1 Architecture

Figure 5.1 shows the up-to-date architectural components of CloudSim 7G[51]. The overall structure is the same: a user-facing section for Cloud providers, application developers, and researchers, called the "User Code", and a "backend" section dedicated to the development of the simulator and its modules.

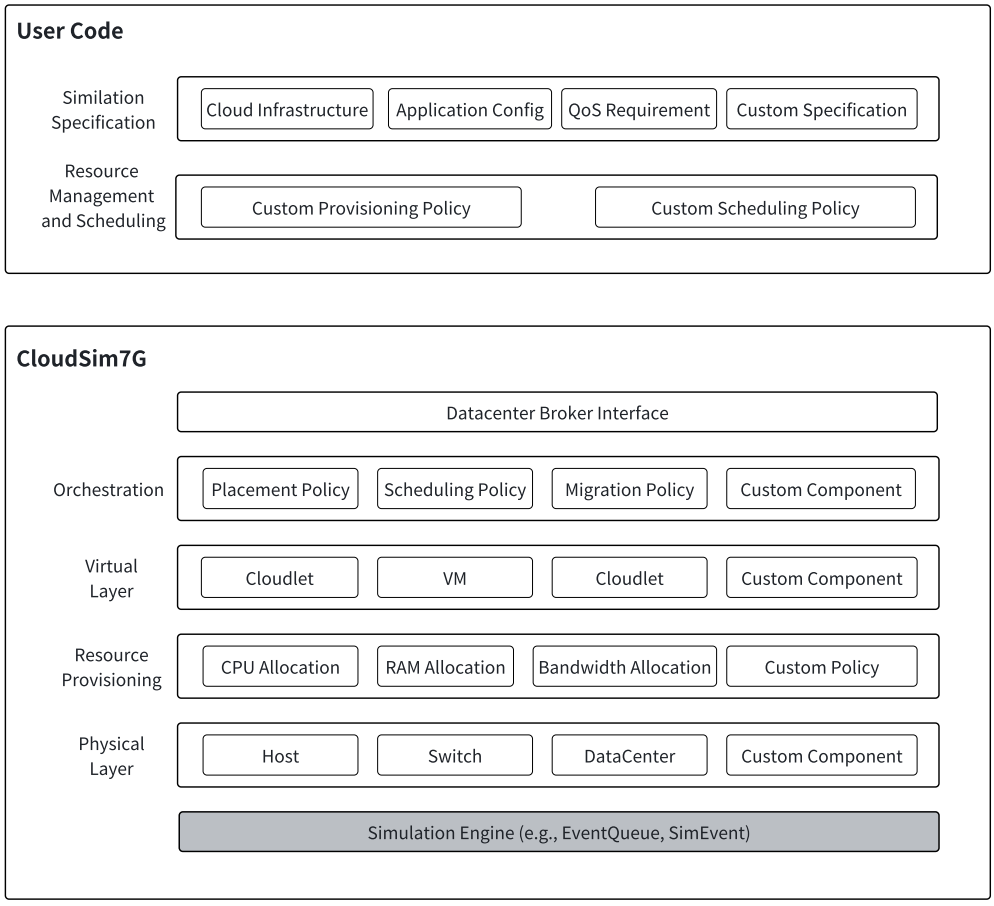


Figure 5.1 The architecture of CloudSim 7G. Blue boxes correspond to the CloudSim components of the base layer. Orange boxes correspond to user-defined policies and configurations, as well as newly created or extended CloudSim components.[51]

CloudSim 7G adopts a layered architecture, with the bottom-most layer serving as the simulation engine that governs core functionalities such as initiating, pausing, and terminating simulated entities, as well as managing the dispatch of discrete events during

runtime. Building upon this foundation, the framework models key components of cloud infrastructure, from physical hosts and network switches to virtualized entities and orchestration mechanisms such as VM placement and cloudlet scheduling policies. The "User Code" layer provides the flexibility to configure and extend these layers by specifying infrastructure characteristics, application workloads, resource requirements, and quality-of-service constraints. Through build automation tools like Maven or Gradle, users can integrate additional modules to support advanced capabilities such as workflow execution, power-aware resource allocation, or serverless computing. The function of each layer can be described as follows:

(1) Simulation Engine

- `CloudSim`: Main class. Responsible for controlling the simulation event queue and managing the sequential execution of the event queue.
- `DeferredQueue`: Used to implement the deferred event queue used by CloudSim.
- `FutureQueue`: Used to implement the future event queue used by CloudSim.
- `CloudInformationService (CIS)`: Serves as a central entity responsible for resource registration, indexing, and discovery within the cloud simulation environment.
- `SimEntity`: Represents a simulated entity capable of both transmitting messages to other entities and receiving and processing incoming messages.
- `CloudSimTags`: This class defines static command and time tags that are used by CloudSim entities to identify the type of operation associated with sending or receiving events.
- `SimEvent`: This entity facilitates the transmission of simulation events among multiple entities within the simulation environment.
- `CloudSimShutdown`: This entity is used to terminate all end-user and proxy entities and send a termination signal to CIS.

(2) CloudSim7G Layer

The CloudSim layer supports the modeling and simulation of cloud data centers, and is mainly responsible for monitoring system status, scheduling from host to virtual machine, managing and executing applications, and other basic issues. The CloudSim layer can be divided into 5 layers:

- Physical Layer: Host, Switch and DataCenter models the basic physical infrastructure. Custom components allows integration of enhanced physical models.
- Resource Provisioning: Manages fine-grained provisioning of physical resources to virtual entities.
- Virtual Layer: Represents virtual tasks and machines that abstract application execution.Custom components enables support for complex or specialized virtual entities (e.g., Network Cloudlet).

- Orchestration: Coordinates how VMs and tasks are placed, executed, and potentially migrated during simulation.
- Datacenter Broker Interface: Interfaces between user-defined logic and the simulator's core components, facilitating interaction and control over the orchestration process.

(3) User Code

The User Code layer in CloudSim 7G serves as the interface for users—such as cloud providers, application developers, and researchers—to configure and control simulation scenarios. It is composed of two main functional blocks:

Simulation Specification

- Cloud Infrastructure Scenario: Defines the physical layout of the cloud, including hosts, switches, and data centers.
- Application Configuration: Specifies the structure and behavior of the cloud applications to be simulated.
- QoS Requirement: Sets the Quality-of-Service constraints for application execution.
- Custom Specification: Enables user-defined parameters or extensions not covered by standard configurations.

Resource Management and Scheduling

- Custom Provisioning Policy: Allows the user to define how physical or virtual resources (CPU, memory, bandwidth) are allocated.
- Custom Scheduling Policy: Lets the user implement specialized scheduling logic for tasks.

CloudSim 7G further allows programmatic customization of scheduling and migration policies, thereby enabling sophisticated evaluation of resource management strategies. This modular and extensible architecture ensures that CloudSim 7G remains a robust simulation platform for contemporary cloud computing research.

5.1.2 Technology implementation

Figure 5.2 illustrates the class design diagram of the CloudSim simulation platform, which serves as the foundational structure for constructing cloud simulation environments. The functionalities of the principal classes are outlined as follows.

- **CloudSim**: Core simulation engine; manages events, entities, and simulation lifecycle.
- **Datacenter**: Represents a physical cloud infrastructure, including hosts and switches.
- **Host**: Models a physical server with configurable resources (CPU, RAM, bandwidth, storage).

- **VM (VirtualMachine)**: Represents a virtual machine deployed on a host; executes cloudlets.
- **Cloudlet**: Denotes a user task or application; executed on VMs with defined resource demands.
- **DatacenterBroker**: Mediates between users and infrastructure; manages VM and cloudlet submission, placement, and scheduling.
- **VmAllocationPolicy**: Determines how VMs are allocated to hosts.
- **CloudletScheduler**: Defines the policy for scheduling cloudlets on VMs (e.g., time-shared, space-shared).
- **UtilizationModel**: Models the dynamic use of resources (e.g., CPU utilization over time) for a cloudlet.
- **NetworkTopology**: Enables network modeling; simulates latency and bandwidth between components.
- **PowerModel**: Represents power consumption characteristics of a host for energy-aware simulations.

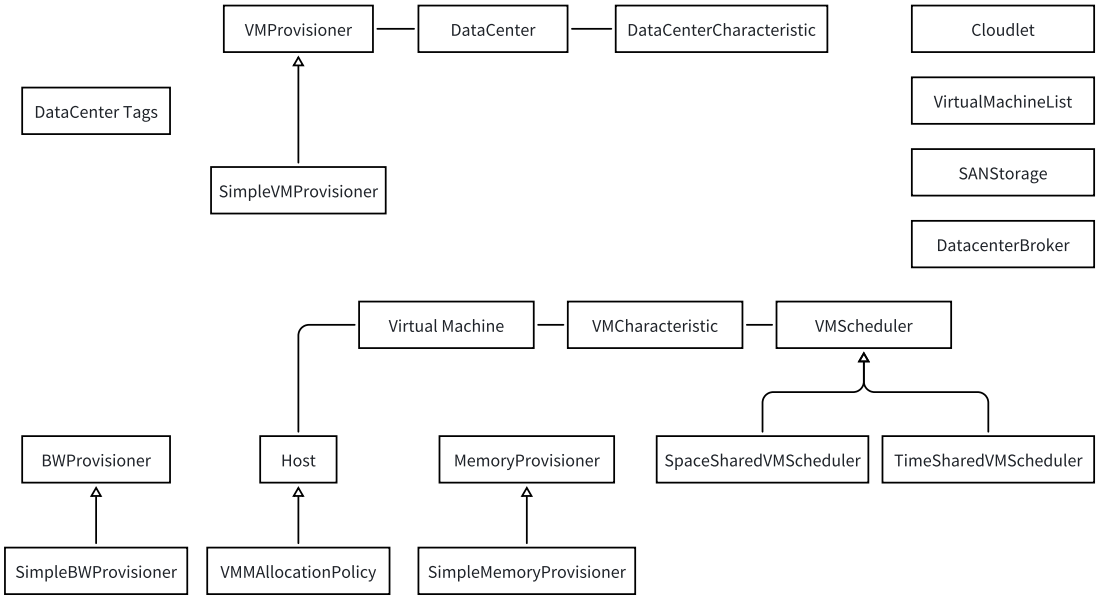


Figure 5.2 Class design diagram of the CloudSim simulation platform

5.1.3 Simulation workflow

The CloudSim framework provides a modular simulation environment for modeling and evaluating cloud computing infrastructures. A typical simulation begins with the initialization of the environment through `CloudSim.init()`, which sets up core simulation variables such as the event queue, simulation clock, and system entities including the Cloud Information Service (CIS). Following initialization, one or more `Datacenter` in-

stances are created to represent cloud infrastructure providers. Each datacenter comprises a set of `Host` objects, each configured with computational cores (`Pe`), memory, bandwidth, and storage, which are managed using provisioners such as `RamProvisionerSimple`, `BwProvisionerSimple`, and `PeProvisionerSimple`. The static attributes of the datacenter, including architecture, operating system, virtual machine monitor (VMM), and pricing models, are encapsulated in a `DatacenterCharacteristics` object.

To simulate user behavior, a `DatacenterBroker` is instantiated to mediate between cloud users and infrastructure providers. Users define virtual machines (`Vm`) with specified parameters such as MIPS, RAM, disk size, bandwidth, and a `CloudletScheduler` to determine task scheduling behavior. Computational tasks are represented by `Cloudlet` objects, which are characterized by instruction length, number of processing elements, file sizes, and utilization models (e.g., `UtilizationModelFull`). The broker submits the VM and cloudlet lists to the simulation environment. The simulation is executed using `CloudSim.startSimulation()` and terminated with `CloudSim.stopSimulation()`. Upon completion, results such as execution status, runtime, start and finish times, and VM assignments are retrieved using `broker.getCloudletReceivedList()`, enabling a comprehensive analysis of scheduling policies and resource utilization in cloud data centers. The entire workflow is described in Figure 5.3.

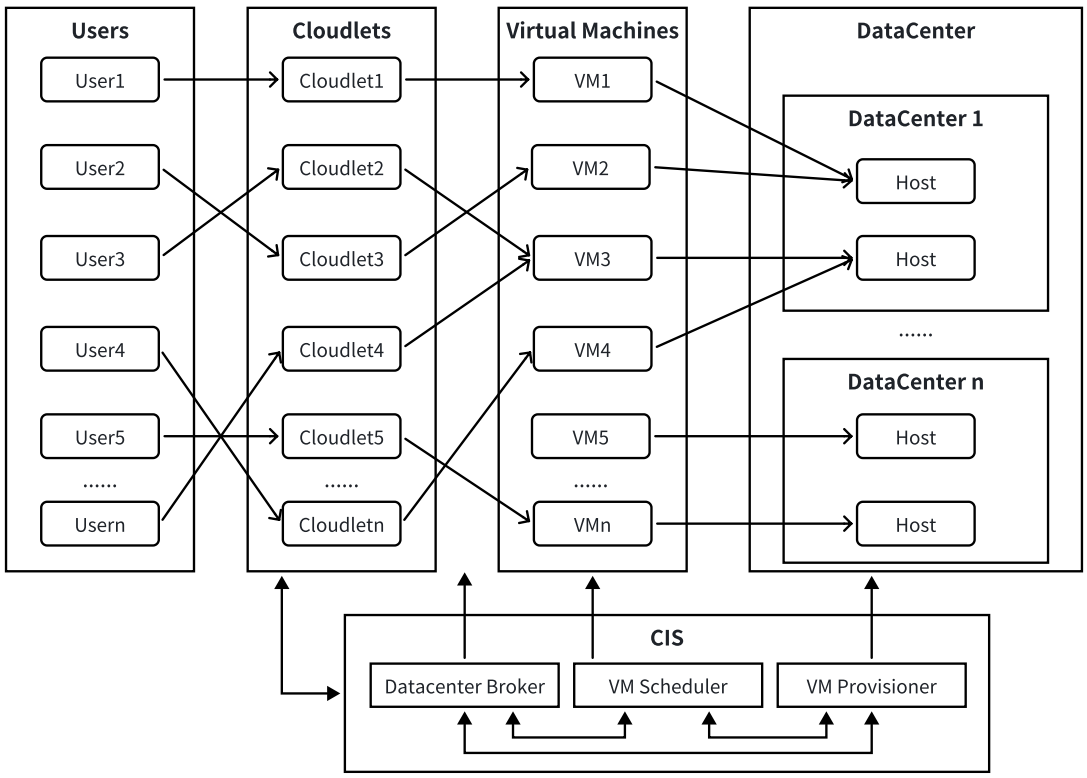


Figure 5.3 Simulation Workflow in CloudSim

To further clarify the components involved in the simulation process described above,

Table 5.1 summarizes the core classes within the CloudSim framework and their respective functionalities. Each class plays a distinct role in modeling different aspects of a cloud computing system, from physical infrastructure to user-level task scheduling. Specifically, `Datacenter`, `Host`, and `Pe` define the hierarchical structure of hardware resources, while `DatacenterBroker`, `Vm`, and `Cloudlet` abstract user interactions and workloads. This modular architecture ensures that each layer of the cloud environment is independently configurable, allowing researchers to simulate and evaluate a wide range of scheduling and allocation strategies.

Table 5.1 Core Class Functions in the CloudSim Framework

Class	**Function Description**
`Datacenter`	Represents a physical cloud infrastructure composed of hosts and manages VM allocation policies. It serves as the primary resource provider in the simulation.
`Host`	Models a physical machine within a datacenter. It includes hardware specifications such as RAM, bandwidth, storage, and one or more processing elements (PEs).
`Pe`	Denotes a single processing element (e.g., CPU core) within a host. It is the basic unit of computation capacity in CloudSim.
`DatacenterBroker`	Acts on behalf of cloud users to manage the submission of virtual machines and cloudlets. It handles VM provisioning and task scheduling policies.
`Vm`	Defines a virtual machine instance with specific computational resources (e.g., MIPS, RAM) and a scheduling policy for cloudlet execution.
`Cloudlet`	Represents a user task or workload to be executed on a VM. It includes attributes such as instruction length, file size, and utilization models.

Figure 5.4 provides a visual overview of the simulation workflow described above, illustrating the sequential interactions between core components in the CloudSim environment. It captures the lifecycle from system initialization and infrastructure setup to the instantiation of brokers, virtual machines, and cloudlets, followed by task execution and result collection. This diagram not only clarifies the relationships among simulation entities but also emphasizes the modular and extensible nature of the framework. By visualizing each stage, the figure aids in understanding how CloudSim enables reproducible and configurable experimentation in cloud resource management and task scheduling research.

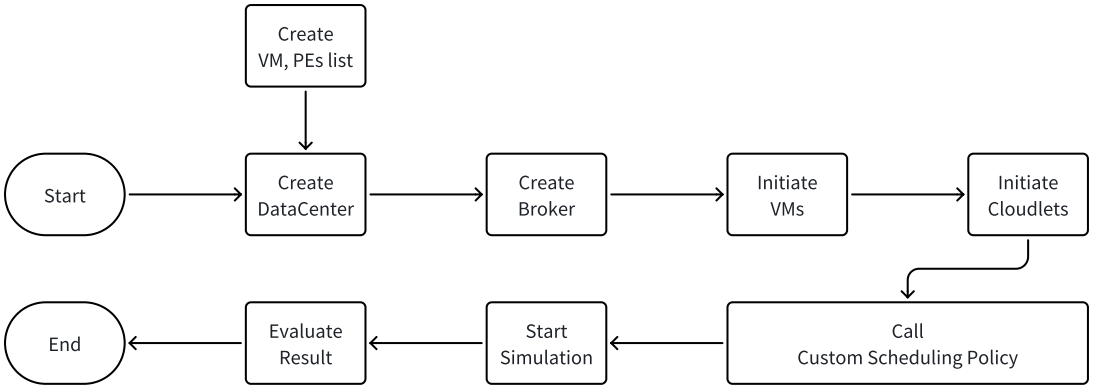


Figure 5.4 Visual Overview of the Simulation Workflow

5.2 Evaluation metrics

To assess the performance of the proposed scheduling strategy, this paper employ four evaluation metrics: makespan, total cost, resource utilization, and load imbalance. Let $\mathcal{C}$ denote the set of completed cloudlets and M denote the number of virtual machines (VMs). For each cloudlet $c \in \mathcal{C}$, let $\text{vm}(c)$ be the VM assigned to c, and $\text{start}(c)$ and $\text{finish}(c)$ denote its execution start and finish times, respectively.

Makespan. Makespan is defined as the maximum finish time among all cloudlets:

$$\text{Makespan} = \max_{c \in \mathcal{C}} \text{finish}(c). \tag{5.1}$$

Total cost. The cost of executing a cloudlet on VM i is estimated by summing three cost coefficients $C_1[i], C_2[i], C_3[i]$. The total cost is calculated as:

$$\text{TotalCost} = \sum_{c \in \mathcal{C}} (\text{finish}(c) - \text{start}(c)) \cdot (C_1[i] + C_2[i] + C_3[i]), \tag{5.2}$$

where $i = \text{vm}(c)$.

Utilization. Resource utilization measures the ratio of total execution time to the product of makespan and number of VMs:

$$\text{Utilization} = \frac{\sum_{c \in \mathcal{C}} (\text{finish}(c) - \text{start}(c))}{\text{Makespan} \cdot M}. \tag{5.3}$$

Imbalance. Load imbalance quantifies the deviation of per-VM workloads from the mean workload. Let T_i be the total execution time on VM i, and $\bar{T} = \frac{1}{M} \sum_{i=1}^{M} T_i$ be the

average. The normalized imbalance is defined as:

$$\text{Imbalance} = \frac{1}{M \cdot \bar{T}} \sum_{i=1}^{M} |T_i - \bar{T}|. \tag{5.4}$$

These metrics provide a comprehensive view of scheduling performance, balancing efficiency (makespan and utilization), economic cost, and load distribution across resources. At the same time, this paper will use these metrics to measure the superiority of each algorithm in the section 5.3: Simulation and result analysis.

5.3 Simulation and result analysis

5.3.1 Simulation environment

This section presents an experimental evaluation of the proposed RL-based method for task scheduling in cloud computing environment. The simulation was implemented using CloudSim 7G, which can be introduced in previous sections. CloudSim enables the emulation of virtual resources and supports comprehensive experimentation with cloud-based data center scenarios. The hardware specifications, software versions, and configuration settings used in this experiment are detailed as follows.

The simulation experiments were conducted on a machine equipped with an AMD Ryzen 7 5800H processor (3.20 GHz) and 16 GB of RAM. The CloudSim framework was executed using JDK 21. The reinforcement learning algorithm was implemented in Python 3.9.18 and executed with PyTorch 2.6.0+cu124, utilizing GPU acceleration provided by an NVIDIA GeForce RTX 3050 Ti GPU. The hardware and software versions used in the experiment are shown in Table 5.2. The configuration of the virtual environment and additional simulation parameters are summarized in Table 5.3.

Table 5.2 Hardware & Software Specification

Type	Component	Specification
Hardware	CPU	AMD Ryzen 7 5800H (3.20 GHz)
	GPU	NVIDIA GeForce RTX 3050 Ti
	RAM	16 GB
Software	Operating System	Windows 11
	CloudSim Version	7.0.0
	JDK Version	21
	Python Version	3.9.18
	CUDA Version	12.4

Table 5.3 Simulation Parameter Settings

Type	Parameter	Value
Host	Number of Hosts	5
	Number of PEs	10
	MIPS	17,700
	Bandwidth	10 GB/s
	Storage	2 TB
	RAM	8 GB
	VM Monitor	Xen
Data Center	Number of Data Centers	1
	VmScheduler	Time Shared
	Cost per Memory	0.05
	Cost per Storage	0.0001
	Operating System	Linux
	Architecture	x86
	VM Monitor	Xen
Cloudlet (Task)	Length of Tasks	100 – 900k
	Number of Tasks	100 – 1,000
Virtual Machine (VM)	Number of VMs	3 – 10
	Processor Speed	800 – 5,000 MIPS
	Memory	1 – 4 GB
	Bandwidth	1,000 – 10,000
	Cost per Memory	0.01 – 1.0
	Cost per Storage	0.01 – 1.0
	Cloudlet Scheduler	Space Shared
	Number of PEs	1
	VM Monitor	Xen

5.3.2 Benchmark datasets

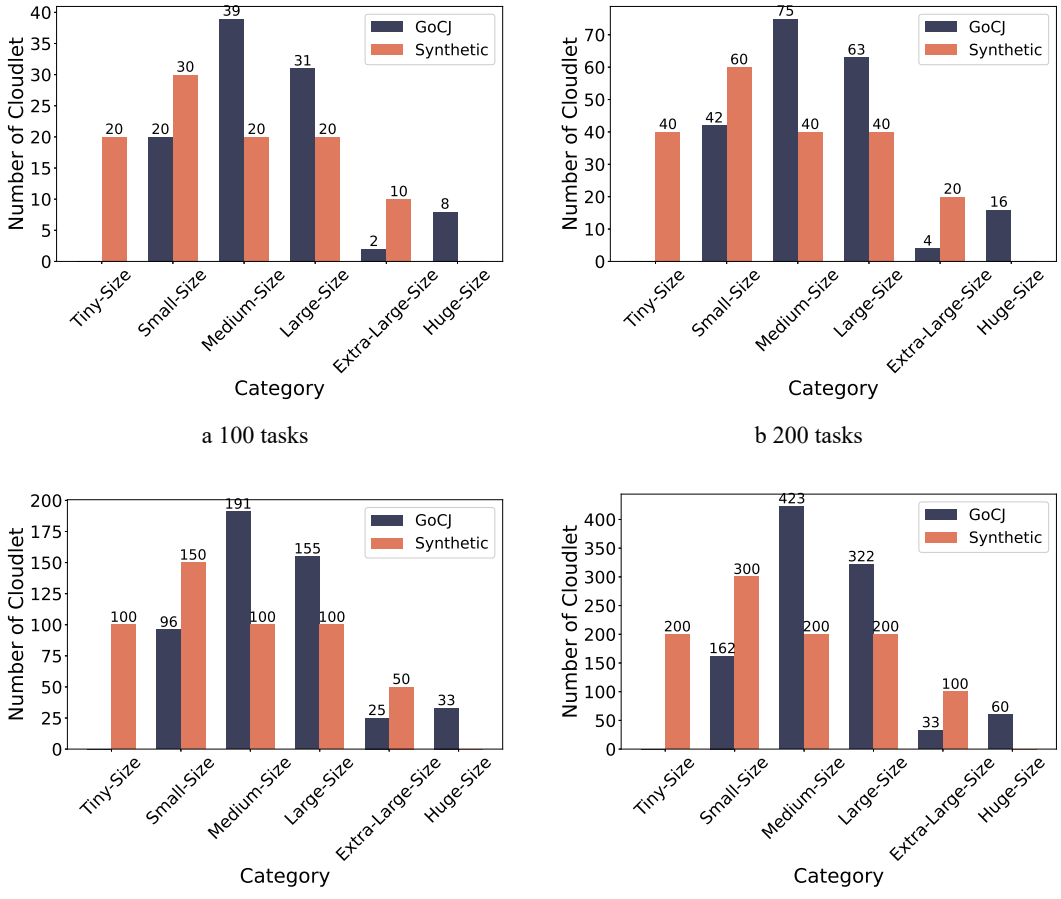
To assess the scheduling efficiency of the proposed method, three distinct datasets were employed in the evaluation:

1. **Random Dataset**: This paper generated a random task dataset containing a total of **100-1000 tasks** using the `RandomDatasetGenerator` program. Task sizes are centered around 50k MIs, with random variations 30k MIs, ensuring a more realistic workload distribution.
2. **GoCJ DataSet**[52]: The GoCJ dataset, regarded as a Google-like realistic dataset,

is synthesized from workload behaviors observed in Google cluster traces using the bootstrapped Monte Carlo method, a widely recognized simulation technique. The task sizes within the GoCJ dataset range from 15,000 to 900,000 million instructions (MIs) and are categorized as follows: small-size jobs (15,000–55,000 MIs), medium-size jobs (59,000–99,000 MIs), large-size jobs (101,000 – 135,000 MIs), extra-large-size jobs (150,000 – 337,500 MIs), and huge-size jobs (525,000–900,000 MIs).

3. **Synthetic Workload Dataset**[53]: The synthetic workload dataset is generated using a random-number generator mechanism based on the Monte Carlo simulation method. It comprises tasks of varying sizes, ranging from 1,000 to 45,000 million instructions (MIs), categorized as follows: tiny-size jobs (1 – 250 MIs), small-size jobs (800 – 1,200 MIs), medium-size jobs (1,800 – 2,500 MIs), large-size jobs (7,000 – 10,000 MIs), and extra-large-size jobs (30,000–45,000 MIs).

Figure 5.5 shows the overall distribution of GoCJ and Synthetic Workload datasets for different numbers of tasks in terms of Millions of Instructions(MIs).



a 100 tasks b 200 tasks

c 500 tasks d 1000 tasks

Figure 5.5 The overall cloudlets distribution of GoCJ and Synthetic Workload datasets: (a) 100 tasks, (b) 200 tasks, (c) 500 tasks, (d) 1000 tasks.

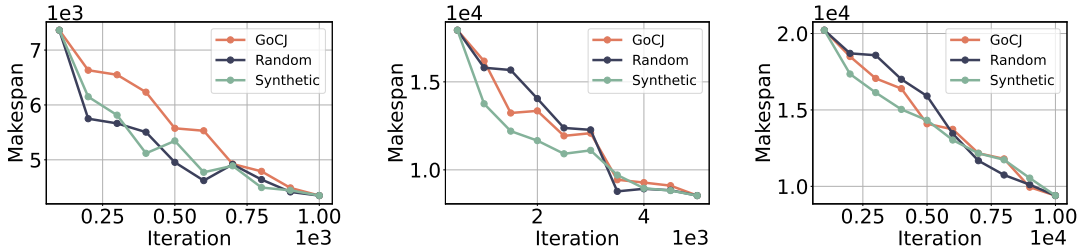
5.3.3 Simulation result of proposed method

To validate the effectiveness of proposed DRL-TS algorithm, this paper conducted experiments on three distinct datasets with varying task and VM sizes. The performance of the algorithm was evaluated using the metrics described in Section 5.2. The hyperparameters for the DRL-TS algorithm were configured as Table 5.4:

Table 5.4 Hyperparameter Configuration for the DRL-TS Algorithm

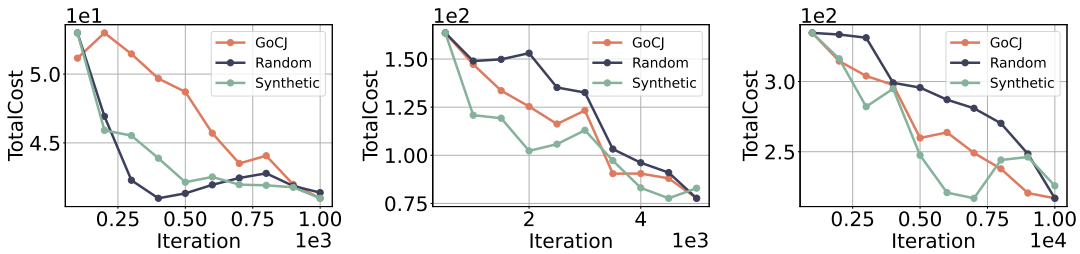
Parameter	Value	Notes
Ψ	$2 \times$ Number of VMs	Input dimension of the DQN network
Ω	Number of VMs	Output dimension of the DQN network
Φ	Adam	Optimization algorithm for network training
δ	0.001	Learning rate for the Adam optimizer
ϵ_{start}	0.9	Initial exploration rate for ϵ-greedy policy
ϵ_{end}	0	Final exploration rate for ϵ-greedy policy
γ	0.99	Discount factor for future rewards
ξ	32	Number of samples per training batch
α	1×10^4	Weighting factor for the reward function

Figure 5.7 illustrates the variation of four key performance metrics—Makespan, Total Cost, Utilization Rate, and Imbalance Rate—across multiple iterations using DRL-TS algorithm. The results are evaluated across three distinct workload settings: **100 tasks, 500 tasks, and 1000 tasks**, providing a comprehensive overview of the algorithm's scalability and efficiency under varying task loads.



100 tasks 500 tasks 1000 tasks

a Makespan



100 tasks 500 tasks 1000 tasks

b Total Cost

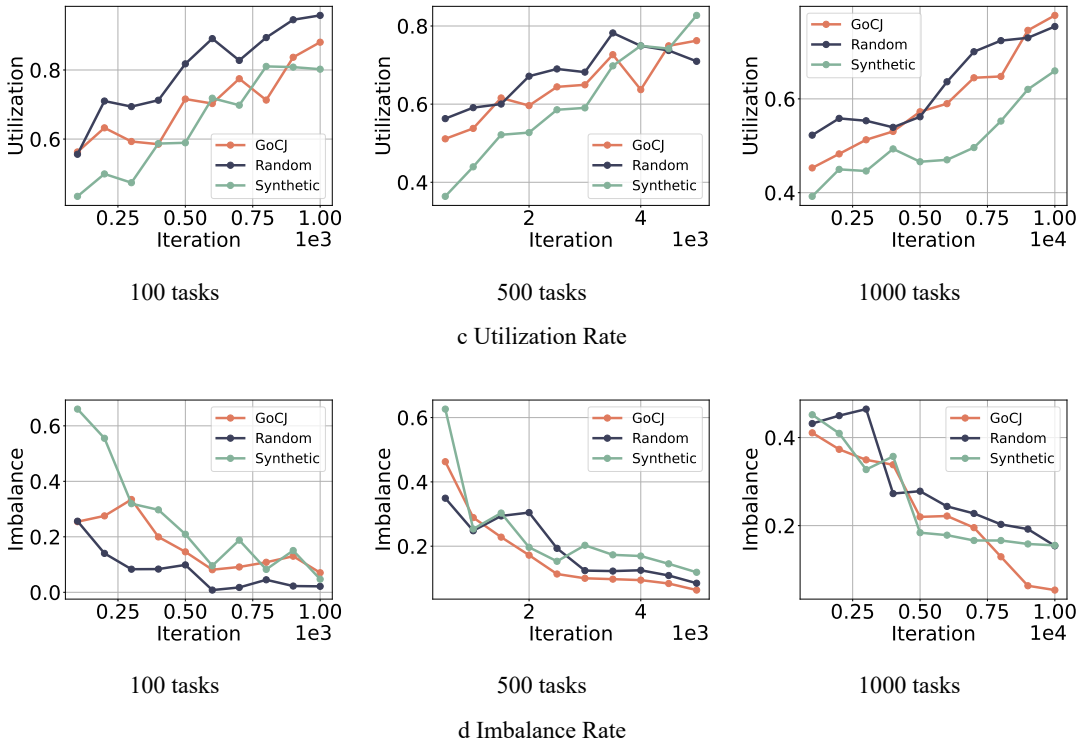


Figure 5.7 Performance comparison of DRL-TS algorithm on (a) Makespan, (b) Total Cost, (c) Utilization Rate, and (d) Imbalance Rate across different task scales (100, 500, 1000 tasks).

For Makespan, the algorithm achieves rapid convergence for smaller task sets (100 tasks), with significant reductions in the early iterations and stabilization after approximately 300 iterations. As the task size increases to 500 and 1000 tasks, the overall Makespan remains higher due to increased computational demands, but the general downward trend remains, indicating the algorithm's ability to reduce execution time even under high-load conditions.

Total Cost follows a similar pattern, with a sharp decline for 100 tasks, reflecting efficient early-stage resource allocation. Although the initial cost reduction is less pronounced for 500 and 1000 tasks due to greater scheduling complexity, the cost continues to decrease over time, highlighting the algorithm's scalability and adaptability.

For Utilization Rate, the algorithm quickly reaches near-optimal utilization for 100 tasks, while the growth is more gradual for larger task sets, reflecting the increased challenge of balancing workloads across multiple VMs. Nevertheless, the consistent upward trend indicates effective resource management.

Finally, the Imbalance Rate decreases rapidly for 100 tasks, suggesting efficient load distribution in smaller task sets. While the decline is slower for 500 and 1000 tasks, the overall downward trend underscores the algorithm's robustness in achieving balanced resource utilization across varying task sizes.

Overall, the performance results demonstrate that the DRL-TS algorithm effectively reduces Makespan, Total Cost, and Imbalance Rate while maximizing Utilization across various task sizes.

5.3.4 Comparasion between DRL-TS and SOTA algorithms

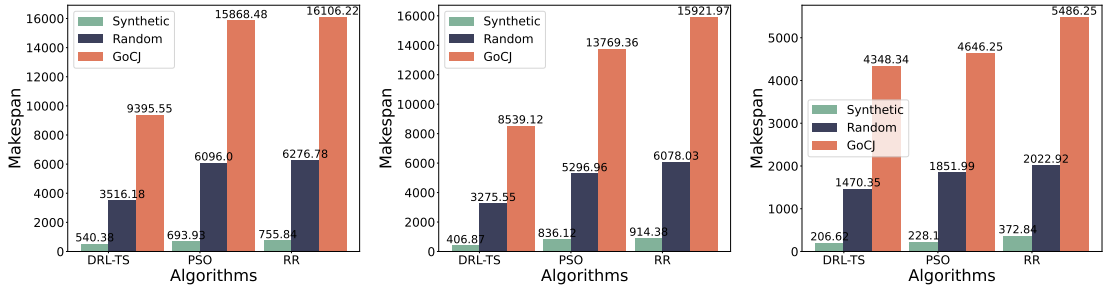
This paper compared the makespan, total cost, utilization rate and imbalance rate of the DRL-TS algorithm with other SOTA algorithms (e.g. RR and PSO) through comparative experiments. The experimental results of each algorithm are shown in Figure 5.8. The parameters are setting to `vm_num`=(10, 5, 3), `cloudlet_num`=(1000, 500, 100), respectively.

The experimental results in Figure 5.8 demonstrate that the proposed **DRL-TS** algorithm consistently outperforms traditional scheduling methods like Round-Robin (RR) and Particle Swarm Optimization (PSO) across four key metrics: Makespan, Total Cost, Utilization Rate, and Imbalance Rate.

For Makespan, DRL-TS achieves significantly shorter execution times, with the lowest value of 540.38 on the Synthetic dataset, compared to 3516.18 for Random and 9395.55 for GoCJ. This reflects DRL-TS's superior ability to efficiently schedule tasks, particularly in diverse cloud environments. In terms of Total Cost, DRL-TS also demonstrates a clear advantage, achieving a minimum of 10.72 on the Synthetic dataset, substantially lower than the 45.72 for Random and 216.83 for GoCJ. For Utilization Rate, DRL-TS consistently achieves the highest levels, with a peak of 0.78 on the Random dataset, indicating efficient resource usage. The Imbalance Rate analysis highlights the strong load balancing capability of DRL-TS, with the lowest imbalance of 0.05 on the Synthetic dataset, compared to 0.15 for Random and 0.16 for GoCJ. This reflects DRL-TS's ability to evenly distribute workloads, reducing performance bottlenecks and improving overall stability. Table 5.5 shows the performance across different datasets and approaches.

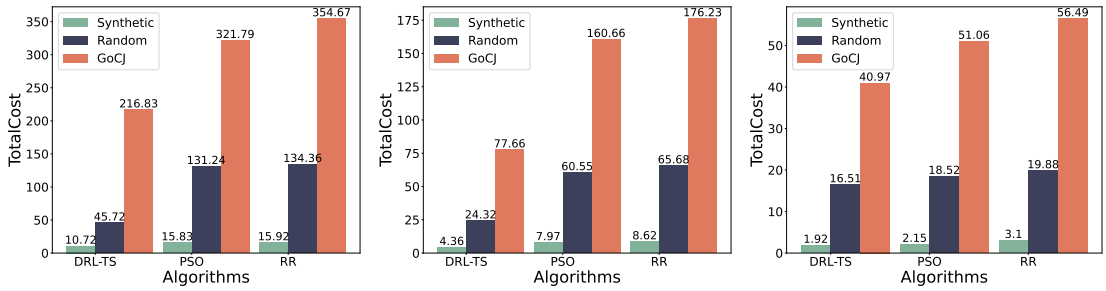
The experimental results presented in Table 5.5 demonstrate the comparative performance of three distinct task scheduling approaches, namely Round Robin (RR), Particle Swarm Optimization (PSO), and Deep Reinforcement Learning Task Scheduler (DRL-TS), across various metrics, datasets, and task scales. The performance metrics evaluated include Makespan, Total Cost, Utilization, and Imbalance, providing a comprehensive assessment of the scheduling efficiency.

The DRL-TS approach consistently outperforms both RR and PSO across most task sizes and datasets. This reflects the ability of DRL-TS to dynamically adapt to changing system states and optimize long-term performance through continuous learning. Additionally, the utilization rates of DRL-TS are generally higher, demonstrating its effectiveness in maintaining balanced resource usage. The imbalance metrics further confirm the robustness of DRL-TS, with consistently lower values, reflecting a more even workload distribution across available resources. Overall, results demonstrate that the DRL-TS method improves makespan, total cost, utilization rate, and imbalance rate by 26.88%, 33.40%, 13.73%, and 59.21%, respectively, compared to the PSO method.



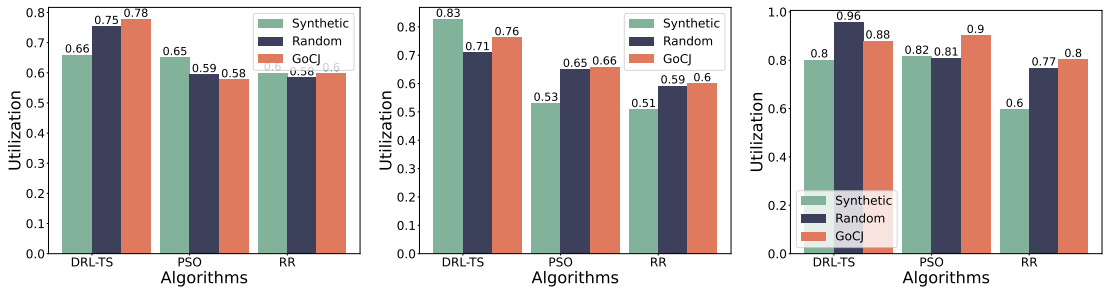
10 VMs, 1000 tasks　　5 VMs, 500 tasks　　3 VMs, 100 tasks

a Makespan comparison



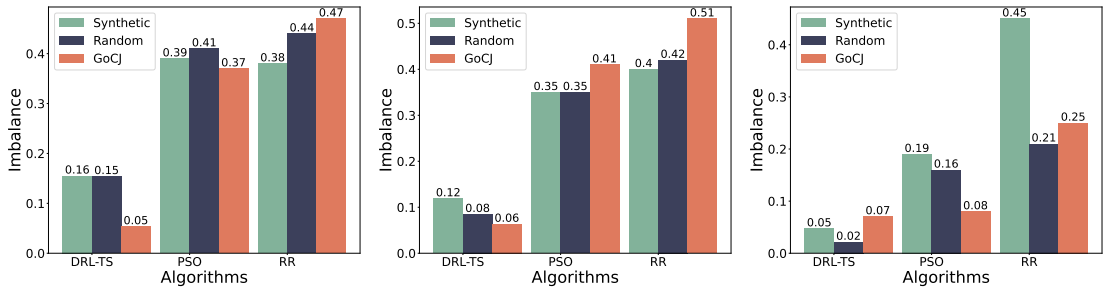
10 VMs, 1000 tasks　　5 VMs, 500 tasks　　3 VMs, 100 tasks

b Total Cost comparison



10 VMs, 1000 tasks　　5 VMs, 500 tasks　　3 VMs, 100 tasks

c Utilization Rate comparison



10 VMs, 1000 tasks　　5 VMs, 500 tasks　　3 VMs, 100 tasks

d Imbalance Rate comparison

Figure 5.8 Comparison between DRL-TS and SOTA algorithms under different settings: (a) Makespan, (b) Total Cost, (c) Utilization Rate, and (d) Imbalance Rate.

Table 5.5 Task Scheduling Performance Across Different Datasets and Approaches

a Makespan

Metrics	Dataset	Task	Task Scheduling Approach		
			RR	PSO	DRL-TS
Makespan	Random	100	2022.92	1851.99	**1470.35**
		200	3890.29	3347.93	**3019.29**
		500	6078.03	5296.96	**3275.55**
		800	4934.35	4570.71	**2820.30**
		1000	6276.78	6096.00	**3516.18**
	GoCJ	100	5486.25	4646.25	**4348.34**
		200	9513.12	9541.50	**8672.42**
		500	15921.97	13769.36	**8539.12**
		800	11432.47	12261.23	**7461.44**
		1000	16106.22	15868.48	**9395.55**
	Synthetic	100	372.84	228.10	**206.62**
		200	637.58	534.37	**414.05**
		500	914.38	836.12	**406.87**
		800	717.02	608.05	**515.84**
		1000	755.84	693.93	**540.38**

b Total Cost

Metrics	Dataset	Task	Task Scheduling Approach		
			RR	PSO	DRL-TS
Total Cost	Random	100	19.88	18.52	**16.51**
		200	39.11	36.30	**33.33**
		500	65.68	60.55	**24.32**
		800	106.99	99.77	**41.92**
		1000	134.36	131.24	**45.72**
	GoCJ	100	56.49	51.06	**40.97**
		200	104.19	105.07	**91.31**
		500	176.23	160.66	**77.66**
		800	247.79	265.91	**144.56**
		1000	354.67	321.79	**216.83**
	Synthetic	100	3.10	2.15	**1.92**
		200	5.57	5.18	**4.08**
		500	8.62	7.97	**4.36**
		800	13.54	12.66	**9.28**
		1000	15.92	15.83	**10.72**

c Utilization

Metrics	Dataset	Task	Task Scheduling Approach		
			RR	PSO	DRL-TS
Utilization	Random	100	0.768	0.809	**0.958**
		200	0.792	0.889	**0.942**
		500	0.592	0.651	**0.709**
		800	0.593	0.616	**0.676**
		1000	0.584	0.598	**0.754**
	GoCJ	100	0.804	**0.903**	0.880
		200	0.894	0.896	**0.903**
		500	0.600	0.659	**0.762**
		800	0.610	0.592	**0.715**
		1000	0.597	0.577	**0.778**
	Synthetic	100	0.597	**0.817**	0.802
		200	0.662	0.764	**0.872**
		500	0.509	0.531	**0.827**
		800	0.517	**0.589**	0.578
		1000	0.600	0.651	**0.659**

d Imbalance

Metrics	Dataset	Task	Task Scheduling Approach		
			RR	PSO	DRL-TS
Imbalance	Random	100	0.21	0.16	**0.02**
		200	0.19	0.08	**0.02**
		500	0.42	0.35	**0.08**
		800	0.43	**0.40**	0.41
		1000	0.44	0.41	**0.15**
	GoCJ	100	0.25	0.08	**0.07**
		200	0.10	0.14	**0.07**
		500	0.51	0.41	**0.06**
		800	0.41	0.46	**0.19**
		1000	0.47	0.37	**0.05**
	Synthetic	100	0.45	0.19	**0.05**
		200	0.34	0.21	**0.11**
		500	0.40	0.35	**0.11**
		800	0.41	0.35	**0.19**
		1000	0.38	0.39	**0.16**

5.4 Chapter summary

This chapter evaluates the proposed Deep Reinforcement Learning-based Task Scheduling (DRL-TS) algorithm using the CloudSim 7G simulation framework. CloudSim 7G's modular and extensible architecture enables realistic modeling of cloud infrastructures, allowing for controlled and reproducible experimentation. The experiments are conducted under varying configurations of virtual machines (VMs), task volumes, and datasets (Random, GoCJ, and Synthetic).

Four evaluation metrics—makespan, total cost, utilization, and load imbalance—are used to assess performance. The simulation results show that DRL-TS consistently outperforms traditional scheduling approaches such as Round Robin (RR) and Particle Swarm Optimization (PSO). Specifically, DRL-TS achieves lower makespan and execution cost while maintaining higher resource utilization and reduced load imbalance. These improvements are evident across all task sizes and datasets, demonstrating the algorithm's scalability and adaptability.

The DRL-TS algorithm's ability to learn optimal scheduling strategies dynamically

enables it to adapt to varying system states and workload distributions. The comparative analysis confirms that DRL-TS delivers more efficient and balanced resource allocation than baseline methods, validating its potential for real-world cloud computing environments. Overall, the chapter demonstrates that DRL-TS is an effective and intelligent task scheduling solution in dynamic and heterogeneous cloud systems.

Chapter6 Summary and prospect

This dissertation presents a dynamic reinforcement learning-based approach to task scheduling optimization for load balancing in cloud computing environments. The proposed method, DRL-TS, addresses the inherent challenges of dynamic task allocation, including unpredictable workloads, fluctuating resource demands, and high-dimensional decision spaces. By leveraging Deep Q-Network (DQN), the proposed framework can dynamically adjust resource allocation strategies based on real-time system states, thus significantly enhancing the efficiency of cloud resource utilization. Key contributions of this work include:

- The formulation of the task scheduling problem as a Markov Decision Process (MDP), capturing the real-time status of virtual machines (VMs) and task execution contexts to improve decision accuracy.
- The design of a custom reward function that balances multiple objectives, including makespan minimization, cost reduction, and load distribution, ensuring optimal task placement across VMs.
- The integration of experience replay and target networks to stabilize training, addressing the challenges of non-stationary data and high-dimensional state spaces typical in cloud computing.

Despite its promising results, the proposed approach also highlights several areas for future research. First, the current framework relies on a fixed neural network architecture, which may limit its adaptability to diverse cloud environments. Future work could explore adaptive architectures that dynamically adjust their depth and complexity based on real-time workload patterns. Additionally, the integration of multi-agent reinforcement learning (MARL) could further enhance scalability and efficiency by allowing multiple agents to collaboratively manage distributed cloud resources.

Moreover, while the current approach effectively addresses task scheduling within a single cloud environment, future studies should investigate cross-cloud or multi-cloud scenarios, where resource heterogeneity and network latency introduce additional layers of complexity. Incorporating edge computing and Internet of Things (IoT) considerations could also expand the applicability of this approach, addressing real-time decision-making challenges in latency-sensitive applications.

Finally, the security and reliability of reinforcement learning-based scheduling algorithms should not be overlooked. Robustness against adversarial attacks and fault tolerance under extreme workloads are critical for maintaining service quality in practical deployments. Research into secure RL strategies, including adversarial training and federated learning, could further enhance the reliability and trustworthiness of cloud-based task scheduling solutions.

Acknowledgement

This dissertation represents not only the culmination of my bachelor journey but also the collective support, guidance, and encouragement I have received from many people along the way. I would like to take this opportunity to express my deepest gratitude to all those who have contributed to the completion of this work.

First and foremost, I would like to express my sincere appreciation to my supervisor at China University of Geosciences (CUG), Professor Xiaoyu Chen, for his consistent and insightful guidance throughout every stage of this research. From the formulation of the research idea to the final draft of the dissertation, Professor Chen has provided invaluable academic advice, thoughtful feedback, and unwavering encouragement. His rigorous academic attitude, clarity of thought, and dedication to mentoring students have deeply influenced my approach to scientific research.

I am also deeply grateful to my co-supervisors from the Illinois Institute of Technology (IIT), Professor Mustafa Bilgic and Professor Yousef Elmehdwi. Their expertise in the field, constructive criticism, and ongoing support played a vital role in shaping the direction and quality of this work.

Beyond my advisors, I would like to thank my classmates, peers, and colleagues who have contributed to my growth—whether through academic discussion, project collaboration, or moral support. Their willingness to share knowledge, listen patiently, and assist during key phases of the research was truly appreciated. I am fortunate to have experienced a sense of academic community and friendship that enriched my time as a student.

Last but by no means least, I am deeply indebted to my family. Their love, support, and patience have been a constant source of strength throughout my studies. They have stood by me in times of doubt, offered encouragement in moments of difficulty, and celebrated every small achievement along the way.

As I reflect on this journey, I am filled with immense gratitude. This work is the result of not just individual effort, but the guidance, collaboration, and support of many.

References

[1] KATHOLE A B, VHATKAR K, LONARE S, et al. Optimization-based resource scheduling techniques in cloud computing environment: A review of scientific workflows and future directions[J]. Computers and Electrical Engineering, 2025, 123: 110080.

[2] PRAVEENCHANDAR J, TAMILARASI A. An enhanced load balancing approach for dynamic resource allocation in cloud environments[J]. Wireless Personal Communications, 2022, 122(4): 3757-3776.

[3] REHAN H. Revolutionizing America's cloud computing the pivotal role of AI in driving innovation and security[J]. Journal of Artificial Intelligence General science (JAIGS), 2024, 2(1): 239-240.

[4] VERGARA J, BOTERO J, FLETSCHER L. A comprehensive survey on resource allocation strategies in fog/cloud environments[J]. Sensors, 2023, 23(9): 4413.

[5] MUNISWAMY S, VIGNESH R. Joint optimization of load balancing and resource allocation in cloud environment using optimal container management strategy[J]. Concurrency and Computation: Practice and Experience, 2024, 36(12): e8035.

[6] ABID A, MANZOOR M F, FAROOQ M S, et al. Challenges and issues of resource allocation techniques in cloud computing[J]. KSII Transactions on Internet and Information Systems (TIIS), 2020, 14(7): 2815-2839.

[7] MOUSAVI S, MOSAVI A, VÁRKONYI-KÓCZY A R, et al. Dynamic resource allocation in cloud computing[J]. Acta Polytechnica Hungarica, 2017, 14(4): 83-104.

[8] MOUSAVI S, MOSAVI A, VARKONYI-KOCZY A R. A load balancing algorithm for resource allocation in cloud computing[J]. Recent Advances in Technology Research and Education, 2018: 289-296.

[9] BEGAM G S, SANGEETHA M, SHANKER N R. Load balancing in DCN servers through SDN machine learning algorithm[J]. Arabian Journal for Science and Engineering, 2022, 47(2): 1423-1434.

[10] MNIH V, KAVUKCUOGLU K, SILVER D, et al. Playing atari with deep reinforcement learning[J]. CoRR, 2013, 1312.5602.

[11] KREUTZ D, RAMOS F M V, VERÍSSIMO P, et al. Software-defined networking: A comprehensive survey[J]. CoRR, 2014, 1406.0440.

[12] FILALI A, MLIKA Z, CHERKAOUI S, et al. Preemptive SDN load balancing with machine learning for delay sensitive applications[J]. IEEE Transactions on Vehicular Technology, 2020, 69(12): 15947-15963.

[13] AL-MASHHADI S, ANBAR M, JALAL R A, et al. Design of cloud computing load balance system based on sdn technology[J]. Computational Science and Technology, 2020: 123-133.

[14] KANG B, CHOO H. An SDN-enhanced load-balancing technique in the cloud system[J]. The Journal of Supercomputing, 2018, 74(11): 5706-5729.

[15] ABDELLTIF A A, AHMED E, FONG A T, et al. SDN-based load balancing service for cloud servers[J]. IEEE Communications Magazine, 2018, 56(8): 106-111.

[16] LIU Y, ZENG Z, LIU X, et al. A novel load balancing and low response delay framework for edge-cloud network based on SDN[J]. IEEE Internet of Things Journal, 2020, 7(7): 5922-5933.

[17] PRAVEEN S P, SARALA P, KUMAR T N S K M, et al. An adaptive load balancing technique for multi SDN controllers[C]//2022 International Conference on Augmented Intelligence and Sustainable Systems (ICAISS). 2022: 1403-1409.

[18] GOVINDARAJAN K, KUMAR V S. An intelligent load balancer for software defined networking (SDN) based cloud infrastructure[C]//2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT). 2017: 1-6.

[19] XU Y, XU W, WANG Z, et al. Load balancing for ultradense networks: a deep reinforcement learning-based approach[J]. IEEE Internet of Things Journal, 2019, 6(6): 9399-9412.

[20] DITTAKAVI R S S. Deep learning-based prediction of CPU and memory consumption for cost-efficient cloud resource allocation[J]. Sage Science Review of Applied Machine Learning, 2021, 4(1): 45-58.

[21] KAUR A, KAUR B, SINGH P, et al. Load balancing optimization based on deep learning approach in cloud environment[J]. International Journal of Information Technology and Computer Science, 2020, 12: 8-18.

[22] SHAFIQ D A, JHANJHI N, ABDULLAH A. Machine learning approaches for load balancing in cloud computing services[C]//2021 National Computing Colleges Conference (NCCC). 2021: 1-8.

[23] ADIL M, NABI S, ALEEM M, et al. CA-MLBS: content-aware machine learning based load balancing scheduler in the cloud environment[J]. Expert Systems, 2023, 40(4): e13150.

[24] 杨震南, 柳燕, 王新奇. 云计算环境下的虚拟机负载均衡遗传算法应用[J]. 智能物联技术, 2025, 57(01): 21-25.

[25] 聂清彬. 基于蚁群模拟退火算法的云计算资源调度[J]. 计算机应用, 2024, 44(S2): 139-142.

[26] 程林钢, 王亚光. 基于分布特征学习灰狼优化算法的云计算资源调度方法[J]. 计算机应用与软件, 1-12.

[27] 龚瑞涛. 基于多目标遗传算法的云服务系统资源调度失效局部最优感知方法[J]. 长江信息通信, 2024, 37(08): 21-23.

[28] 毛锋晨. 云计算环境下资源动态分配算法的优化策略[J]. 信息记录材料, 2025, 26(02): 178-180.

[29] 李逸凡, 吴怡, 杨辉. 基于 OpenStack 私有云平台的资源动态调度方法[J]. 福建师范大学学报 (自然科学版), 2025, 41(02): 1-8.

[30] 郑龙海, 肖博怀, 姚泽玮, 等. 基于图强化学习的多边缘协同负载均衡方法[J]. 计算机科学, 2025, 52(03): 338-348.

[31] KRUEKAEW B, KIMPAN W. Multi-objective task scheduling optimization for load balancing in cloud computing environment using hybrid artificial bee colony algorithm with reinforcement learning[J]. IEEE Access, 2022, 10: 17803-17818.

[32] HAYYOLALAM V, ÖZKASAP Ö. CBWO: A novel multi-objective load balancing technique for cloud computing[J]. Future Generation Computer Systems, 2025, 164: 107561.

[33] DEVARAJ A F S, ELHOSENY M, DHANASEKARAN S, et al. Hybridization of firefly and improved multi-objective particle swarm optimization algorithm for energy efficient load balancing in cloud computing environments[J]. Journal of Parallel and Distributed Computing, 2020, 142: 36-45.

[34] HARIS M, ZUBAIR S. Mantaray modified multi-objective Harris hawk optimization algorithm expedites optimal load balancing in cloud computing[J]. Journal of King Saud University - Computer and Information Sciences, 2022, 34(10): 9696-9709.

[35] WANG L, von LASZEWSKI G, YOUNGE A, et al. Cloud Computing: a Perspective Study[J]. New Generation Computing, 2010, 28(2): 137-146.

[36] YOUSAFZAI A, GANI A, NOOR R M, et al. Cloud resource allocation schemes: review, taxonomy, and opportunities[J]. Knowledge and Information Systems, 2017, 50(2): 347-381.

[37] 王登科. 云计算任务调度算法的研究与实现[D]. 西北师范大学, 2013.

[38] THAKUR A, GORAYA M S. RAFL: A hybrid metaheuristic based resource allocation framework for load balancing in cloud computing environment[J]. Simulation Modelling Practice and Theory, 2022, 116: 102485.

[39] GABAY M, ZAOURAR S. Vector bin packing with heterogeneous bins: application to the machine reassignment problem[J]. Annals of Operations Research, 2016, 242(1): 161-194.

[40] L.D. D B, VENKATA KRISHNA P. Honey bee behavior inspired load balancing of tasks in cloud computing environments[J]. Applied Soft Computing, 2013, 13(5): 2292-2303.

[41] JAFARNEJAD GHOMI E, MASOUD RAHMANI A, NASIH QADER N. Load-balancing algorithms in cloud computing: A survey[J]. Journal of Network and Computer Applications, 2017, 88: 50-71.

[42] YANG P, ZHANG L, LIU H, et al. Reducing idleness in financial cloud services via multi-objective evolutionary reinforcement learning based load balancer[J]. Science China Information Sciences, 2024, 67(2): 120102.

[43] KHAN A R. Dynamic Load Balancing in Cloud Computing: Optimized RL-Based Clustering with Multi-Objective Optimized Task Scheduling[J]. Processes, 2024, 12(3): 519.

[44] KUMAR M, SHARMA S C. PSO-based novel resource scheduling technique to improve QoS parameters in cloud computing[J]. Neural Computing and Applications, 2020, 32(16): 12103-12126.

[45] BEHERA I, SOBHANAYAK S. Task scheduling optimization in heterogeneous cloud computing environments: A hybrid GA-GWO approach[J]. Journal of Parallel and Distributed Computing, 2024, 183: 104766.

[46] BISOY S, PRADHAN A. A novel load balancing technique for cloud computing platform based on PSO[J]. Journal of King Saud University - Computer and Information Sciences, 2020, 34.

[47] 王登科, 李忠. 基于粒子群优化与蚁群优化的云计算任务调度算法[J]. 计算机应用与软件, 2013, 30(01): 290-293.

[48] DEVI N, DALAL S, SOLANKI K, et al. A systematic literature review for load balancing and task scheduling techniques in cloud computing[J]. Artificial Intelligence Review, 2024, 57(10): 276.

[49] ZHOU J, LILHORE U K, M P, et al. Comparative analysis of metaheuristic load balancing algorithms for efficient load balancing in cloud computing[J]. Journal of Cloud Computing, 2023, 12(1): 85.

[50] HOANG D T, HUYNH N V, NGUYEN D N, et al. Markov Decision Process and Reinforcement Learning[G]//Applications and Implementation Deep Reinforcement Learning for Wireless Communications and Networking: Theory. IEEE, 2023: 25-36.

[51] ANDREOLI R, ZHAO J, CUCINOTTA T, et al. CloudSim 7G: An integrated toolkit for modeling and simulation of future generation cloud computing environments[J]. Software: Practice and Experience, n/a.

[52] HUSSAIN A, ALEEM M. GoCJ: Google Cloud jobs dataset for distributed and cloud computing infrastructures[J]. Data, 2018, 3(4): 38.

[53] HUSSAIN A, ALEEM M, KHAN A, et al. RALBA: a computation-aware load balancing scheduler for cloud computing[J]. Cluster Computing, 2018, 21(3): 1667-1680.