

Article

Balanced-DRL: A DQN-Based Job Allocation Algorithm in BaaS

Chaopeng Guo , Ming Xu, Shengqiang Hu and Jie Song * 

Software College, Northeastern University, Shenyang 110169, China; guochaopeng@swc.neu.edu.cn (C.G.); 2101253@stu.neu.edu.cn (M.X.); 1801237@stu.neu.edu.cn (S.H.)

* Correspondence: songjie@mail.neu.edu.cn

Abstract: Blockchain as a Service (BaaS) combines features of cloud computing and blockchain, making blockchain applications more convenient and promising. Although current BaaS platforms have been widely adopted by both industry and academia, concerns arise regarding their performance, especially in job allocation. Existing BaaS job allocation strategies are simple and do not guarantee load balancing due to the dynamic nature and complexity of BaaS job execution. In this paper, we propose a deep reinforcement learning-based algorithm, Balanced-DRL, to learn an optimized allocation strategy in BaaS based on analyzing the execution process of BaaS jobs and a set of job scale characteristics. Following extensive experiments with generated job request workloads, the results show that Balanced-DRL significantly improves BaaS performance, achieving a 5% to 8% increase in job throughput and a 5% to 20% decrease in job latency.

Keywords: Blockchain as a Service; job allocation; load balancing; deep reinforcement learning

MSC: 68M14; 68M20; 94A16



Citation: Guo, C.; Xu, M.; Hu, S.; Song, J. Balanced-DRL: A DQN-Based Job Allocation Algorithm in BaaS. *Mathematics* **2023**, *11*, 2638. <https://doi.org/10.3390/math11122638>

Academic Editor: Jan Lansky

Received: 10 May 2023

Revised: 6 June 2023

Accepted: 8 June 2023

Published: 9 June 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Blockchain as a Service (BaaS) has recently emerged as a platform that provides blockchain technology as cloud services [1]. BaaS not only reduces the development and management costs associated with blockchain, but it also benefits from cloud computing characteristics such as high flexibility, scalability, and reliability [2]. BaaS is now widely adopted by both industries [3] and academic institutions, including Microsoft Azure (<https://azure.microsoft.com/en-gb/solutions/blockchain/>, accessed on 10 May 2023), IBM Blockchain Platform (<https://www.ibm.com/blockchain/platform>, accessed on 10 May 2023), and Amazon (<https://aws.amazon.com/cn/blockchain/>, accessed on 10 May 2023). Additionally, distinctive BaaS platforms are currently under development [4,5]. Due to the broad applications of BaaS, concerns have arisen regarding its performance. To measure the performance of BaaS, indicators such as average job throughput and average job delay are typically used [6].

Job allocation is an effective approach in cloud computing to improve platform performance. It involves allocating cloud computing jobs to proper execution nodes using a specific allocation strategy. In a data center, an execution node can be a physical server or a virtual machine. Typical job allocation algorithms include packing algorithms, heuristic algorithms [7], and machine learning-based algorithms [8]. The main objective of these algorithms is to achieve load balancing, which improves average job throughput and reduces average job delay.

Although BaaS utilizes concepts and techniques from cloud computing, the existing job allocation approaches used in cloud computing are insufficient for BaaS. Firstly, most job allocation approaches in cloud computing are based on allocating resources to virtual machines with consistent resources. This allows these allocation problems to be solved using boxing or heuristic algorithms [9,10]. However, in BaaS, jobs are allocated to containers with varying and dynamic resources during runtime. Secondly, BaaS job

characteristics differ from those in cloud computing. The resources required by BaaS jobs are more complex and dynamic. For instance, when generating a new block in Fabric [11], the required resources are influenced by endorsement nodes and endorsement strategies. All in all, the existing job allocation approaches used in cloud computing cannot be directly applied to BaaS.

The current job allocation approaches BaaS uses mostly consist of two simple allocation strategies: round-robin polling and random allocation. However, these strategies cannot guarantee load balancing among BaaS containers. As a result, these strategies might decrease average job throughput and increase average job delay.

Therefore, our research motivation comes from the following statements: (1) BaaS urgently needs continuous development to boost its performance. (2) The existing allocation methods for cloud computing platforms are difficult to apply to the BaaS platform. (3) The job size of BaaS is difficult to measure, which limits job allocation in BaaS. (4) The dynamic nature of BaaS containers makes it challenging for current optimization algorithms to generate optimized allocation strategies in a reasonable amount of time.

To overcome the job allocation problem in BaaS, we analyzed the execution process of BaaS jobs and proposed a set of job characteristics to measure job scale. Next, we formulated the job allocation problem as an optimization problem and analyzed the corresponding constraints and objectives. Finally, we propose a novel deep reinforcement learning-based (DRL) algorithm, named Balanced-DRL, to learn the optimal allocation strategy. We designed corresponding state and action spaces for the job allocation problem and replaced the network structure used in the original algorithm. Additionally, our algorithm adopts a Boltzmann action exploration strategy to accelerate the training process. Our main contributions include the following:

- We analyzed the execution and allocation processes of BaaS jobs in detail and proposed a set of job scale characteristics that can benefit our job allocation approach and other BaaS-related optimization approaches in the future.
- We propose a novel deep reinforcement learning (DRL)-based algorithm, Balanced-DRL, to produce an optimized job allocation strategy in BaaS. We carefully designed the action space, state space, and reward function required for DRL.
- We conducted experiments to evaluate the performance of Balanced-DRL compared with other commonly-used allocation strategies, including random allocation, round-robin allocation, and minimum resource allocation. The results show that Balanced-DRL can maximize job throughput and minimize job delay, especially under various load scenarios.

The rest of the paper is organized as follows: In Section 2, we discuss the differences and improvements of our work compared with related research. Section 3 presents an overview of BaaS jobs, including their definitions and allocation process, as well as our proposed job scale characteristics. We also discuss the constraints and objectives of the job allocation problem. In Section 4, we first discuss the different allocation strategies required in different job-workload scenarios; then, we propose a novel algorithm, Balanced-DRL, and demonstrate its process for allocating BaaS jobs. Section 5 proposes a deep Q-network-based algorithm, HDQN, for optimizing job allocation in the high-load scenario. We explain the design of the action space, the state space, and the reward function, as well as the training procedure. Section 6 describes a series of experiments we conducted to validate the performance of Balanced-DRL, including comparisons with other traditional allocation strategies. In Section 7, we first discuss the experimental results of our study, followed by a comparative analysis between the algorithm proposed in this paper and some representative works in the related field. Finally, in Section 8, we conclude the paper and propose future research directions.

2. Related Works

In the Related Works section, we first introduce the relevant algorithms for traditional data center job allocation problems, analyze their applicable scenarios, and identify their

limitations when applied to the BaaS job allocation problem. We then discuss optimization solutions for blockchain, particularly Hyperledger Fabric, which is the most widely used blockchain for enterprise applications. In the end, we review research on BaaS platforms.

2.1. Job Allocation Algorithms for Traditional Data Centers

In traditional data centers, job allocation to suitable nodes is also important to improve resource utilization, increase throughput, reduce energy costs, and ensure load balancing. In this section, we will introduce job allocation algorithms in traditional data centers from three perspectives: traditional bin-packing algorithms, heuristic algorithms, and machine learning algorithms.

Regarding bin-packing algorithms, Li et al. [9] addressed the problem of reliable resource allocation in the context of an unreliable bin-packing problem. They used the First-Fit algorithm combined with the mirror algorithm to minimize the number of bins used while satisfying reliability requirements. Liu et al. [12] abstracted the job allocation problem in cloud computing environments as the MinUsageTime dynamic bin-packing problem. To address the issue that traditional dynamic bin-packing algorithms do not provide any performance guarantees in harsh environments, they proposed a robust and consistent algorithm that guarantees a lower bound on performance. Gupta et al. [10] proposed two new dynamic bin-packing algorithms based on the Best-Fit-Decreasing heuristic, achieving optimization in terms of energy consumption, resource utilization, and completion time. However, bin-packing algorithms are only suitable for static job allocation with known resources. They do not apply to the dynamic job allocation problem in BaaS, where the resources are unknown.

Regarding heuristic algorithms, Kumar et al. [13] studied energy-aware resource allocation in cloud data centers using a two-level ant colony optimization algorithm. Li et al. [14] proposed a new multi-objective particle swarm optimization algorithm for dynamic resource allocation that assigns a sub-problem to each particle using problem decomposition and accelerates convergence to the optimal solution by introducing a new particle velocity update strategy. Shiekh et al. [15] proposed a hybrid heuristic algorithm for parallel task allocation and load balancing in cloud computing environments. By maximizing resource utilization using three stages—parallelization, task allocation, and task redistribution—the algorithm aims to provide better services. Infantia et al. [16] proposed a novel meta-heuristic algorithm for virtual machine (VM) placement and migration in cloud environments, aiming to achieve optimal VM placement and migration, reduce the number of active servers, and minimize completion time and energy consumption. Ghobaei-Arani et al. [17] proposed an efficient IoT service deployment solution based on the whale optimization algorithm to address the deployment problem of IoT applications in fog computing. The proposed solution selects suitable fog nodes based on throughput and energy consumption, while satisfying the quality of service (QoS) requirements of IoT services. Simulation results showed that this solution reduces both latency and energy consumption.

Although heuristic algorithms such as ant colony optimization and particle swarm optimization have been widely used in resource allocation problems, they cannot guarantee the optimal solution and cannot predict the deviation between feasible and optimal solutions. Moreover, their stability and generality may be limited. Therefore, in this paper, we do not adopt heuristic algorithms for BaaS job allocation optimization.

Regarding machine learning algorithms, Zhang et al. [18] proposed two resource allocation prediction algorithms, Linear-ALLOC and Logistic-ALLOC, based on linear and logistic regression, respectively. These algorithms generalize and learn the optimal multi-dimensional cloud resource allocation scheme by learning the optimal allocation schemes on small training sets. Bal et al. [19] proposed a combined approach for secure and efficient task scheduling and resource allocation in cloud computing using a hybrid machine learning technique called RATS-HM. Experimental results showed improved resource utilization, energy consumption, and response time. Talwani et al. [20] proposed

an energy-efficient VM allocation and migration algorithm using a machine learning-based artificial bee colony algorithm to rank VMs based on load and select the most efficient VMs for job allocation and migration. Simulation experiments showed that this research was able to reduce energy consumption.

Yi et al. [21] used deep reinforcement learning to allocate long-running compute-intensive jobs. They used the deep Q-network to train the job allocation offline to maximize cumulative rewards. They used a computational model based on the LSTM network to capture the dynamic changes in server energy consumption and heat. To reduce the energy consumption of cloud data centers, Yan et al. [22] proposed a method based on deep reinforcement learning to allocate incoming jobs to appropriate virtual machines, to achieve high-quality service. Jayanetti et al. [23] proposed a workflow scheduling framework based on deep reinforcement learning to address the task scheduling problem in cloud-edge collaborative environments. Compared with other benchmark algorithms, their framework showed significant improvements in multiple metrics, such as energy consumption, execution time, and the percentage of jobs completed before their deadlines.

Seid et al. [24] proposed a dynamic resource allocation algorithm based on blockchain and multi-agent deep reinforcement learning for allocating resources of multiple drones to mobile user devices. The blockchain in the algorithm ensures the security of virtual resource transactions among mobile user devices, infrastructure providers, and virtual network operators. The resource allocation problem is abstracted as a multi-layered Stackelberg game problem, and reinforcement learning is used to learn allocation strategies. The experiments demonstrated that the algorithm achieved state-of-the-art results in utility optimization and service quality.

Wang et al. [25] proposed a new blockchain-based, hierarchical, digital twin Internet of Things (IoT) framework that aims to minimize system latency and energy consumption. To address the resource allocation problem, a multi-agent reinforcement learning algorithm based on proximity policy optimization was employed. The experimental results demonstrated that this approach can improve system efficiency and balance system latency and energy consumption.

Reinforcement learning can learn efficient job resource allocation methods without requiring specific models or prior knowledge, by interacting with the data center environment. Deep reinforcement learning, particularly deep Q-learning, combines the strong generalization ability of deep neural networks and can handle high-dimensional state space problems that traditional reinforcement learning cannot solve. Therefore, in this paper, we implement efficient BaaS job allocation based on deep reinforcement learning.

2.2. Optimization Solutions for Blockchain

Currently, the main optimization goals for blockchain are to improve the throughput of job requests and reduce latency.

FastFabric [26] achieves higher throughput by identifying performance bottlenecks and optimizing consensus, validation modules, and the world state. However, its test data and environment are too ideal and may not apply to the complex situation of the Hyperledger Fabric network in BaaS.

Javaid et al. [27] analyzed various fine-grained delays occurring during the block verification phase and achieved 1.3 to 2 times greater verification latency optimization by using chaincode caching and concurrent read-write of the world state during the verification phase.

Kwon et al. [28] proposed two optimization methods in the endorsement and ordering phases. For the endorsement phase, they optimized the throughput of read requests. For the ordering phase, they optimized the ordering service by creating a new consensus protocol, reducing latency.

Hang et al. [29] proposed a new method for building blockchain networks to address scalability and performance issues. They analyzed the configurable network components

in Hyperledger Fabric and proposed an optimized network configuration, which improved throughput and latency.

Nakaike et al. [30] tested the performance of Hyperledger Fabric using GoLevelDB as the database and found that the database was one of the bottlenecks affecting throughput. To improve the performance of Hyperledger Fabric, they disabled database compression.

Zhang et al. [31] identified performance bottleneck analysis rules with extensive performance testing. For the network running in real time, they monitored performance data, node resources, and network parameters in real time. They identified the reasons for performance limitations based on the performance bottleneck analysis rules. Finally, they dynamically adjusted network parameters to improve performance.

The paper [32,33] focused on reducing the latency caused by block propagation through the network. They proposed a score-based algorithm for neighbor selection, using propagation latency as the scoring criterion. By using this algorithm, they reduced latency in the blockchain network.

Li et al. [34] proposed a method to reduce the average latency of blockchain networks by using a probabilistic verification approach. By introducing a metric to determine whether verification is necessary, transmission can occur immediately if verification is not required. However, this method may have certain security risks.

To improve the throughput of blockchain, Locher et al. [35] proposed an optimization algorithm at the consensus layer, a synchronous, leader-based Byzantine agreement protocol. This protocol is easy to implement and malleable, and can accelerate the consensus speed and improve throughput.

Geng et al. [36] employed deep reinforcement learning algorithms to optimize blockchain consensus algorithms, which is denoted as the Deep Reinforcement Consensus algorithm, and applied the algorithm to the business model of intelligent manufacturing. The experimental results demonstrated that the algorithm with blockchain could enhance system decentralization, save storage space, and improve the efficiency of intelligent manufacturing.

Cao et al. [37] addressed the issue of severe performance degradation of the Raft algorithm in private chains when network latency is high and proposed using the Multi-Raft algorithm with multiple leaders instead of the Raft algorithm. Experiments showed that the Multi-Raft algorithm could reduce network traffic in larger-scale networks and improve system scalability and performance.

Jalalzai et al. [38] proposed an algorithm that randomly selects a group of nodes for consensus to improve the complexity of the Byzantine Fault-Tolerant protocol in blockchain and address the performance issues of a single primary node. This algorithm can save bandwidth and enhance the scalability of the blockchain.

To address the issue of low throughput in public blockchains, Zhang et al. [39] improved the compact-block relaying protocol in blockchain, which allows blocks with a large number of transactions to propagate without introducing additional propagation latency.

The above methods have achieved certain optimizations for blockchain performance, especially throughput and latency. However, these studies mostly focus on modifying Hyperledger Fabric itself or related network parameters, which may have poor adaptability to environmental changes. This article proposes a performance optimization method for BaaS without changing the underlying blockchain technology.

2.3. Research Related to BaaS Platform

Lu et al. [5] proposed a unified BaaS platform solution called uBaaS, which suggests deploying as a service to avoid the impact of different cloud computing used by different BaaS platforms on development. Additionally, the proposed solution introduces a design model as a service to address the issues of data management and contract design in a unified manner, thereby avoiding the impact of different blockchain services provided by different BaaS platforms.

Zheng et al. [40] designed a BaaS platform named NutBaaS, which provides functions such as network deployment, system monitoring, contract analysis, and testing, allowing developers to focus more on business logic development.

Ma et al. [41] implemented a trusted BaaS platform called TrustedBaaS based on Hyperledger Fabric. The platform provides functions such as identity authentication, sensitive data protection, and trusted data storage, effectively protecting the privacy and security of data without sacrificing performance.

Weerasinghe et al. [42] proposed a blockchain-based platform to address issues such as subscriber management, roaming customers, spectrum, security, and infrastructure that arise from deploying 5G networks. This platform includes functions such as service rating, bidding, and selection, which can ensure advantages such as availability, decentralization, secure transfer payments, and more.

Onik et al. [43] summarized the current major BaaS platforms and compared the blockchain services supported by these platforms.

Jiang et al. [44] proposed a general-purpose BaaS platform called Polychain to address the scalability limitations and customization difficulties of existing BaaS platforms. Polychain is designed with high modularity, flexibility, scalability, reliability, and security, utilizing multiple design principles from software engineering to make it more flexible and convenient.

Li et al. [45] studied recent BaaS models and classified them into three categories: data-layer models, network layer models, and others. These models have optimized security, privacy, and trust, but most do not address issues such as energy and cost.

Rajendra et al. [46] addressed the issue that most existing blockchain platforms run and deploy in virtual environments without considering the performance and power consumption of IoT devices. They designed and developed BlockPaaS, which allows users to run various blockchain protocols in a real IoT environment, considering device performance, power consumption, deployment, and other issues.

Zheng et al. [47] found that existing BaaS platforms do not provide Service Level Agreements to ensure the quality of service. Therefore, they proposed a consensus algorithm based on the KNN algorithm. This algorithm classifies transactions by priority and guarantees the quality of service by prioritizing the execution of some users' requests.

Cai et al. [48] believe that current BaaS platforms built in the environment of cloud service providers compromise the trustlessness and availability of blockchain. Therefore, they designed a new cloud-edge collaborative BaaS platform. By selecting redundant blockchain nodes, leader election, and self-recovering edge networks, they built a highly available BaaS platform.

The research mentioned above mainly focuses on constructing BaaS platforms, making it more convenient for users to use and deploy BaaS platforms. In contrast, this paper proposes optimization schemes for BaaS platforms' performance.

2.4. Brief Summary

In Section 2, we introduce our related work from three aspects: traditional job scheduling algorithms in data centers, optimization methods for blockchain, and research on BaaS platforms. By summarizing the traditional job scheduling algorithms, we analyze their shortcomings and gain inspiration from reinforcement learning algorithms. Through the summary of optimization methods for blockchain, we identify the main optimization objectives, including throughput and latency. We also recognize that current methods mainly focus on optimizing the blockchain itself and lack optimization for BaaS. Finally, by summarizing research on BaaS platforms, we find that the focus of current research is on the construction of the platform, lacking research on performance improvement. From the perspective of job scheduling on BaaS platforms, we focus on improving the performance of BaaS platforms, especially in terms of throughput and latency.

3. Allocation Model

This section analyzes the characteristics of BaaS jobs and presents a detailed overview of the job allocation process. We then formulate the allocation process as an optimization problem, proposing corresponding constraints and objectives that must be satisfied for optimal job allocation.

3.1. BaaS Job Analysis

Two main types of BaaS jobs can be identified: query and transaction. Query jobs involve clients sending queries to the ledger to retrieve transaction records, which are then returned. Transaction jobs, on the other hand, involve the ledger generating transaction records based on trading operations, which are then recorded in the account book.

Figure 1 illustrates a typical process for BaaS query jobs, which involves the following three steps:

1. A client submits a request using the BaaS service. The service application constructs a query proposal and sends it to a peer node in the corresponding channel.
2. Upon receiving the proposal, a local chaincode is called by the peer node to query local account data in the ledger. This step includes proposers' identity verification, proposal validation, query simulation, and generation of a reading set. Eventually, the query result is returned to the application.
3. Upon receiving the query result, the service application returns it to the client.

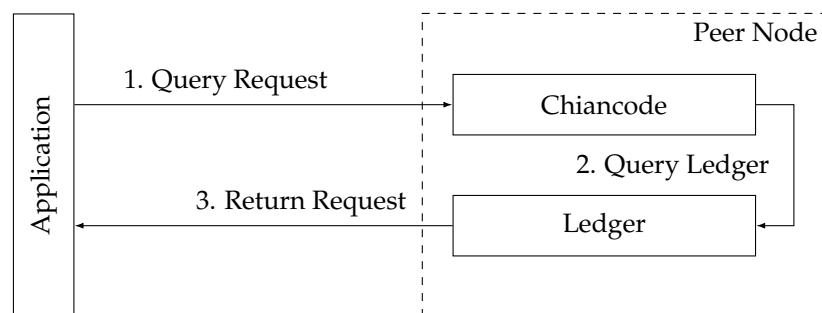


Figure 1. Query Job Execution Flow in Hyperledger Fabric.

In a query job flow, the execution of the chaincode constitutes a key and complex process, which is the primary source of latency.

Figure 2 shows the execution process of a BaaS transaction job, which includes the following five steps:

- A client initiates a transaction job by sending a transaction proposal to an endorsement node *P* using the BaaS service. The endorsement strategy in the organization of the channel determines the endorsement node.
- Upon receiving the proposal, endorsement node *P* verifies the proposal's legitimacy and determines the chaincode to be called. It then simulates the execution of the chaincode function, which includes reading and writing to the ledger, generating simulated transaction results, and puts the results into a read–write set. *P* signs the results to indicate the endorsement by the organization and sends them back to the application.
- After collecting the results that meet the endorsement strategy, the application packages the endorsement results together with the proposal into endorsement transactions and sends them to a sorting node *S*.
- Sorting node *S* collects endorsement transactions from each client in the channel. It sorts and packages multiple transactions into blocks and distributes them to all submission nodes under each organization in the channel.

- Submission nodes P_1 and P_2 verify the block and each endorsement transaction in the block, and they update the local ledger data. At the same time, the application notifies the client on completing the job.

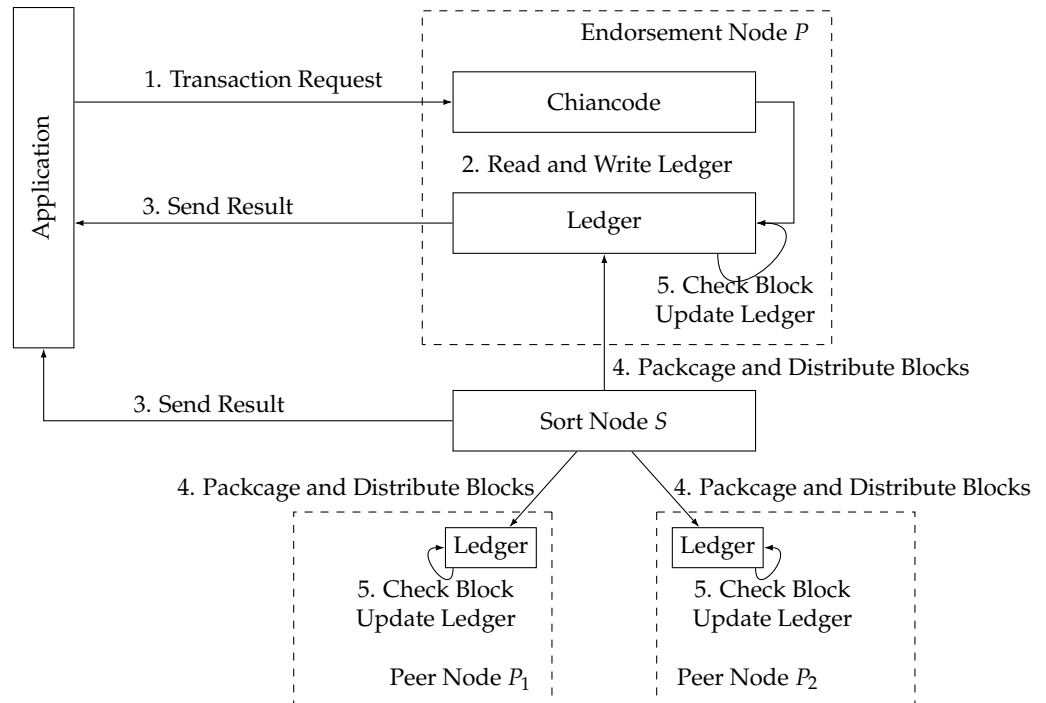


Figure 2. Transaction Job Execution Flow in Hyperledger Fabric.

In the transaction process, step 2 needs to complete jobs such as verification, simulation execution, saving and reading data, transaction results, and endorsement, which is the main reason for latency.

Whether a job is a query job or a transaction job, the steps of executing the chaincode are always the most complex and time-consuming. Although BaaS jobs need to consume resources in each step of the execution process, we only consider the resources consumed in chaincode execution, namely, step 2, in query jobs and transaction jobs.

To measure the scale of BaaS jobs, two types of characteristics must be considered: the number of reading and writing operations, and the size of datasets. Based on process analysis, we selected the features listed in Table 1 to measure the job scale. These characteristics can effectively describe the scale evaluation for both query and transaction jobs. The only difference is that transaction jobs generate a written set. To maintain simplicity and consistency in job scale measurement, we established the following convention: if both Put State and Del State are 0, the corresponding job is a transaction job. Otherwise, it is a query job.

Table 1. BaaS Job Scale Characteristics.

Item	Unit	Description	Target Dataset
Chain Size	Block	Size/height of the currently queried blockchain	-
Query Block	Block	Number of times to query historical blocks	Blockchain and block index
Query Transaction	Transaction	Query historical transaction times	Blockchain and block index
Query State	Number of key	Number of query key history writes	Blockchain and history index
World Size	Megabytes	The size of the current world state	-
Get State	Key value pair	Number of times to query world state	World State and Read Set
Put State	Key value pair	Number of times to update world state	World State and Write Set
Del State	Number of Key	Number of times to delete world state	World State and Write Set

BaaS jobs in Fabric call chaincode APIs to perform designed functions, and jobs are written in GO language. Therefore, the above job scale characteristics can be obtained with the language static analysis method.

3.2. BaaS Job Allocation Model

A chaincode is always instantiated on a peer node, and it runs in a container. Compared with virtual machines, containers are lighter and more flexible [49]. Resources consumed by a container only depend on the overhead of chaincode operations. When a BaaS job is assigned to a peer node, the chaincode receives the job and starts execution. When a peer node has no job, the chaincode container is idle. For the convenience of modeling, we have the following assumptions:

- When a chaincode is idle, it does not consume any resources.
- When a chaincode is running, the total resources required from the beginning to the end remain unchanged.
- The BaaS service dynamically creates more chaincode containers for a higher job workload since a chaincode can only process one job per time.
- The total number of chaincode containers created on one peer node is limited to avoid affecting the performance and isolation of containers.
- The resource types considered are CPU, memory, and disk I/O. We assume that the disk storage and network bandwidth are unlimited.

(1) BaaS Job Request

Let $\mathbf{J} = \{j_1, j_2, \dots\}$ represent a set of BaaS jobs. A job $j = \langle \text{ChannelID}, \text{ChaincodeID}, \text{Args} \rangle$ indicates a requirement for calling the chaincode in the corresponding channel and passing the related parameters. In addition, let t_i represent the specific time when job j_i is submitted to the BaaS.

(2) BaaS Platform

The BaaS platform can be considered a general cloud computing platform, including servers, virtual machines, peer nodes, and containers. The server set of BaaS is denoted as $\mathbf{N} = \{n_1, n_2, \dots\}$. $\forall i, n_i$ contains several virtual machines. In general, we assume that a virtual machine is a peer node. Therefore, the peer nodes set in n_i is denoted as $\mathbf{P}^i = \{p_1^i, p_2^i, \dots\}$, and the set of all peer nodes is denoted as \mathbf{P} . Furthermore, we denote jobs allocated to \mathbf{P}^i as \mathbf{J}^i . p_j^i hosts several containers, denoted as $\mathbf{C}^{ij} = \{c_1^{ij}, c_2^{ij}, \dots\}$. Each container only hosts one chaincode.

To simplify our description, we have the following simplified denotations:

- A peer node p_j^i of node n_i is simplified as p_j to indicate an arbitrary peer node.
- A container c_z^{ij} of peer node p_j^i is simplified as c_z to indicate an arbitrary container.

(3) Resources and Utilization

A function R indicates the total amount of resources of a server, a peer node. For example, $R(p_j, \mathcal{C})$, $R(p_j, \mathcal{M})$, and $R(p_j, \mathcal{D})$ represent p_j 's CPU capacity, memory capacity, and disk I/O, respectively. The resource allocation constraint is shown in Equation (1), indicating that the total resources allocated to peer nodes equal the resources available on the server.

$$\sum_{p_j \in \mathbf{P}^i} R(p_j, \mathcal{T}) = R(n_i, \mathcal{T}), \mathcal{T} \in \{\mathcal{C}, \mathcal{M}, \mathcal{D}\} \tag{1}$$

A function U indicates the total amount of resources a container uses at a specific time. For example, $U(c_z, \mathcal{C}, t)$ gives the total amount of CPU used by c_z at t . Combined with the resource allocation constraint of p_j , we have the resource usage constraint for c_z as shown in Equation (2). Equation (2) indicates that the usage of a certain resource \mathcal{T} by all

containers c_z on peer node p_j should be less than the amount of resource \mathcal{T} owned by the peer node.

$$\sum_{c_z \in \mathbf{C}^{ij}} U(c_z, \mathcal{T}, t) \leq R(p_j, \mathcal{T}) \tag{2}$$

For resource \mathcal{T} of a peer node p_j , the usage rate of this resource at time t is equal to the sum of the resource usage of all containers divided by the total amount of resources available on the peer node. The resource utilization rate of p_j at time t is denoted as $U^t(p_j, \mathcal{T})$, presented by Equation (3). The resource utilization rate of nodes is a critical metric. Improving the resource utilization rate of nodes can enhance the ability of BaaS to process job requests, namely, increasing throughput and reducing latency.

$$U^t(p_j, \mathcal{T}) = \frac{\sum_{c_z \in \mathbf{C}^{ij}} U(c_z, \mathcal{T}, t)}{R(p_j, \mathcal{T})} \tag{3}$$

To compare the relative resource utilization of different peer nodes, the comprehensive resource utilization for p_j is denoted as $U^t(p_j)$, which is shown in Equation (4), where ω_C , ω_M , and ω_D represent the weight factors for CPU utilization, memory utilization, and disk I/O utilization. Additionally, we have $\omega_C + \omega_M + \omega_D = 1$. Traditional cloud computing applications can be divided into compute-intensive and I/O-intensive. For the former, CPU resources are more precious and should be given greater weight, while for the latter, memory and disk resources are more important and should be given greater weight. However, for BaaS applications, there are no obvious features of CPU, memory, or disk resource usage. In this case, we refer to some literature [19] and treat all resources equally.

$$U^t(p_j) = \omega_C \times U^t(p_j, \mathcal{C}) + \omega_M \times U^t(p_j, \mathcal{M}) + \omega_D \times U^t(p_j, \mathcal{D}) \tag{4}$$

The resources required by a container are determined by the job it runs, which is presented as a resource vector, shown in Equation (5). $U(j, \mathcal{C})$, $U(j, \mathcal{M})$, and $U(j, \mathcal{D})$ represent the amount of CPU resources, memory resources, and hard disk I/O required to complete job j , respectively. Δt represents the time required to complete the job. We assume that the total resources required by a chaincode container to complete the job are the same from the job start to the job end. Therefore, we omit Δt in Equation (5).

$$\langle U(j, \mathcal{C}), U(j, \mathcal{M}), U(j, \mathcal{D}) \rangle \tag{5}$$

(4) Allocation Constraints

If a BaaS job is about to be assigned to a peer node, the node must meet the following constraints:

1. The peer node is equipped with a chaincode.
2. The peer node currently has at least one available chaincode.
3. If it is a transaction job, the node complies with the endorsement policy corresponding to the chaincode.

For constraint 1, we introduce a binary variable ϕ , denoted as $\phi(\text{ChannelID}, \text{ChaincodeID}, p_j)$, to indicate whether the corresponding chaincode is installed on a peer node. Specifically, if $\phi = 1$, it indicates that the chaincode is installed; otherwise, if $\phi = 0$, it is not installed.

For constraint 2, we consider a scenario in which the containers on a node are running and the number of containers has already reached the maximum limit. In this case, the job cannot be assigned to the node immediately. Instead, it has to wait for an available chaincode before execution. Consequently, this waiting time introduces a certain latency in the job allocation.

For constraint 3, in Hyperledger, an endorsement policy for a transaction chaincode can be specified using logical operators such as AND, OR, and NoutOf, as well as role assignments to specific organizations. For instance, the policy $\text{OR}(o_1.\text{Admin}, \text{AND}(o_2.\text{Member},$

o_3 .Member)) requires endorsement from either the administrator of organization o_1 or both members of organizations o_2 and o_3 . To simplify our model, we only consider a single layer of AND/OR policies composed of organizations.

3.3. Optimization Goal

The main goal of this work is to improve the performance of BaaS and thereby enhance the quality of service. To quantify BaaS performance, we use four indicators: resource utilization, load balancing, job throughput, and job latency.

(1) Resource Utilization

Equation (6) defines the total resource utilization of BaaS U^t at time t . Its value is equal to the weighted sum of the total resource utilization of all nodes $U^t(\mathbf{N}, \mathcal{C})$, $U^t(\mathbf{N}, \mathcal{M})$, and $U^t(\mathbf{N}, \mathcal{D})$ under the weights $\omega_{\mathcal{C}}$, $\omega_{\mathcal{M}}$, and $\omega_{\mathcal{D}}$. The range of $U^t(\mathbf{N})$ is $[0, 1]$.

$$U^t(\mathbf{N}, \mathcal{T}) = \frac{\sum_{ij} U(p_j^i, \mathcal{T}, t)}{\sum_{ij} R(p_j^i, \mathcal{T})} \tag{6}$$

$$U^t(\mathbf{N}) = \omega_{\mathcal{C}} \times U^t(\mathbf{N}, \mathcal{C}) + \omega_{\mathcal{M}} \times U^t(\mathbf{N}, \mathcal{M}) + \omega_{\mathcal{D}} \times U^t(\mathbf{N}, \mathcal{D})$$

(2) Load Balancing

Load balancing can be described as resource balancing and job balancing. In particular, we measure the load balancing factor between peer nodes at time t using the quantity $RB^t(\mathcal{T})$ (resource balancing) and JB^t (job balancing). Both values approach zero as the load across peer nodes becomes more balanced. To calculate the load balancing factor, we use Equation (7) to obtain the absolute deviation of each node’s relative average utilization and job number and divide them by the number of nodes. Here, the average utilization and the average of jobs are represented by $\overline{U^t(\mathbf{P}, \mathcal{T})}$ and \bar{J} .

$$\begin{aligned} \overline{U^t(\mathbf{P}, \mathcal{T})} &= \frac{\sum_i \sum_j U^t(p_j^i, \mathcal{T})}{\sum_{\mathbf{P}^i \in \mathbf{P}} |\mathbf{P}^i|} \\ RB^t(\mathcal{T}) &= \frac{\sum_{\mathbf{P}^i \in \mathbf{P}} |U^t(\mathbf{P}^i, \mathcal{T}) - \overline{U^t(\mathbf{P}, \mathcal{T})}|}{\sum_{\mathbf{P}^i \in \mathbf{P}} |\mathbf{P}^i|} \\ \bar{J} &= \frac{|\mathbf{J}|}{\sum_{\mathbf{P}^i \in \mathbf{P}} |\mathbf{P}^i|} \\ JB^t &= \frac{\sum_{\mathbf{P}^i \in \mathbf{P}} ||\mathbf{J}^i| - \bar{J}|}{\sum_{\mathbf{P}^i \in \mathbf{P}} |\mathbf{P}^i|} \end{aligned} \tag{7}$$

(3) Job Throughput

Equation (8) defines job throughput, JPS . \mathbf{J} defines a job set composing a series of job requests $\{j_1, j_2, \dots\}$ reaching BaaS service. ΔT is the completion interval that is equal to the difference between the arrival time of the first job and the completion time of the last job. JPS is equal to the ratio of the number of jobs $|\mathbf{J}|$ and interval time ΔT . It is worth noting that when the workload is too low, JPS is linear with the workload. Therefore, JPS is more important in the time-varying scenario and a high-workload scenario.

$$JPS = \frac{|\mathbf{J}|}{\Delta T} \tag{8}$$

(4) Job Latency

Equation (9) defines three indicators to measure the job latency: Latency Average (LA), Latency Size (LS), and Latency Ratio (LR), where LA represents the average total time of job completion, LS represents the average of job latency, and LR represents the average proportion of job latency. The actual running time of job j is defined as $T(j, \mathcal{R})$, and the

ideal running time under the condition of sufficient resources is $T(j, \mathcal{N})$. In addition, we count the latency of all jobs without distinguishing between transactions or queries.

$$\begin{aligned}
 LA &= \frac{\sum_{j \in \mathbf{J}} T(j, \mathcal{R})}{|\mathbf{J}|} \\
 LS &= \frac{\sum_{j \in \mathbf{J}} (T(j, \mathcal{R}) - T(j, \mathcal{N}))}{|\mathbf{J}|} \\
 LR &= \frac{\sum_{j \in \mathbf{J}} \frac{T(j, \mathcal{R}) - T(j, \mathcal{N})}{T(j, \mathcal{N})}}{|\mathbf{J}|}
 \end{aligned} \tag{9}$$

Based on the above indicators, the objective functions of the BaaS job allocation problem are shown in Equation (10). The optimized strategy should maximize resource utilization, minimize the load balancing factor, maximize job throughput, and minimize job latency. It is worth noting that even though we have four indicators, all of them reflect the performance of BaaS.

$$\begin{aligned}
 &\max U^t(\mathbf{N}) \\
 &\min RB^t(\mathcal{T}), JB^t \\
 &\max JPS \\
 &\min LA, LS, LR
 \end{aligned} \tag{10}$$

3.4. Brief Summary

In Section 3, we analyze the execution process of two types of jobs in BaaS and identify the key factors that primarily affect performance, specifically, the execution of the chaincode. In this analysis, we identify eight features used to measure the job scale in BaaS. Then, mathematical modeling is applied to the job allocation in the BaaS platform, and the constraints on job allocation are discussed, including the number of chaincode containers and endorsement policies. The symbols used in the equations are summarized in Table 2. Finally, to optimize the performance of BaaS, we illustrate our objective equations.

Table 2. Summary Table of Equation Symbols.

Symbol	Description
p_j	An arbitrary peer node
c_z	An arbitrary container
\mathbf{N}	The server set of BaaS
\mathcal{T}	A type of resource. It might be CPU, memory, or hard disk.
$R(p_j, \mathcal{T})$	Total resource amount of \mathcal{T} on p_j
$U(c_z, \mathcal{T}, t)$	Used resource of \mathcal{T} on c_z
$U^t(p_j, \mathcal{T})$	Resource utilization rate of \mathcal{T} on p_j at time t
$U^t(\mathbf{N}, \mathcal{T})$	Resource utilization rate of \mathcal{T} on \mathbf{N} at time t
$\overline{U^t(\mathbf{P}, \mathcal{T})}$	The average utilization of all peer nodes
$RB^t(\mathcal{T})$	Resource balancing factor of \mathcal{T} at time t
\bar{J}	The average jobs of all peer node
JB^t	Job balancing factor
JPS	Average number of completed jobs per second
LA	Average completion time of jobs
LS	Average job latency
LR	Average proportion of job latency

4. Balanced-DRL

In BaaS, the different workload types influence the specific allocation strategy of the allocation algorithm, as well as the specific optimization goals. Therefore, we first analyze

the requirements of the allocation algorithm in different workload scenarios. Then, the allocation process of Balanced-DRL is discussed in a complex workload scenario.

4.1. Workload Scenario Analysis

(1) Low-Workload Scenario

When the BaaS processes a low workload, job latency prioritizes job throughput. Job throughput with a low workload only depends on the workload size, with a larger workload achieving greater throughput. As for job latency, it only increases when the job demands on a node exceed the node's available resources. Therefore, the job allocation algorithm must maintain relatively balanced resource utilization of each node, known as resource workload balancing. To balance resource workload, jobs should be allocated to candidate nodes with the lowest utilization.

(2) High-Workload Scenario

During a high-workload scenario, the job allocation algorithm must improve overall throughput and reduce job latency. Two problems need to be addressed. Firstly, the resource utilization rate of all nodes is relatively high in the scenario, which may result in many small resource fragments. Therefore, the allocation algorithm needs to allocate jobs to nodes that are beneficial to further improve resource utilization. Secondly, each node undertakes an excessive number of jobs, leading to remarkable job latency. Therefore, the allocation algorithm also needs to consider allocating jobs to nodes with fewer jobs.

(3) Time-Varying-Workload Scenario

When the cluster processes a time-varying job workload, it processes a job workload whose workload size changes with time. Because the workload changes and the scale and resources required by different jobs are different, the following four node types often exist in BaaS at the same time:

1. Nodes with low utilization and a high number of jobs.
2. Nodes with low utilization and a low number of jobs.
3. Nodes with high utilization and a high number of jobs.
4. Nodes with high utilization and a low number of jobs.

Therefore, the allocation algorithm should distinguish the types of peer nodes and apply the corresponding strategies in the low-workload scenario and the high-workload scenario.

4.2. Allocation in Balanced-DRL

The allocation of jobs in varying-workload scenarios necessitates implementing different job allocation strategies. To this end, Balanced-DRL treats nodes with differing workloads in distinct manners. The allocation process of Balanced-DRL, as illustrated in Figure 3, begins by selecting potential peer nodes based on endorsement policies. Following this, the candidate nodes are split into low-workload and high-workload node sets based on a pre-defined resource utilization threshold. When the set of the low-workload node is non-empty, the job is assigned to the peer node with minimum resource utilization.

However, if the set of the low-workload node is empty, it implies that the current status of BaaS is under a high-workload scenario. In the scenario, Balanced-DRL has to consider not only improving overall resource balancing (*RB*), namely, reducing the resource fragments on peer nodes, but also reducing job latency, namely, improving job balancing (*JB*). In a high-workload scenario, the set of validated peer nodes is split into a low-job-number workload node set and a high-job-number workload set by a pre-defined job-number threshold. Balanced-DRL tries to allocate the job to a node in the low-job-number workload node set. If no nodes are within the set, the job is allocated to nodes within the high-job-number workload node set. It is noteworthy that because the allocation in a high-workload scenario needs to decrease the resource fragments of each peer node while considering the job's scale, this process is more intricate. To solve this problem, a deep reinforcement learning-based allocation algorithm, HDQN, is proposed in Section 5.

The interaction process of the aforementioned allocation procedure is shown in Figure 4. The user sends a job request to the BaaS platform using the application. During the job

allocation process, the Endorsement module first selects the available nodes for allocation based on the job’s endorsement policy. Then, a message is sent to the NodeSet module to query the status of the candidate nodes. If there are low-workload nodes among the candidate nodes, the low-workload node set is sent to the MinRes module, and the job is allocated to the node with the lowest resource load. If there are no low-workload nodes among the candidate nodes, but there are low-job nodes that exist, the job is allocated according to the HDQN module within the low-job node set. If both the low-workload node set and the low-job node set are empty, then the job is sent to the HDQN module for allocation within all nodes.

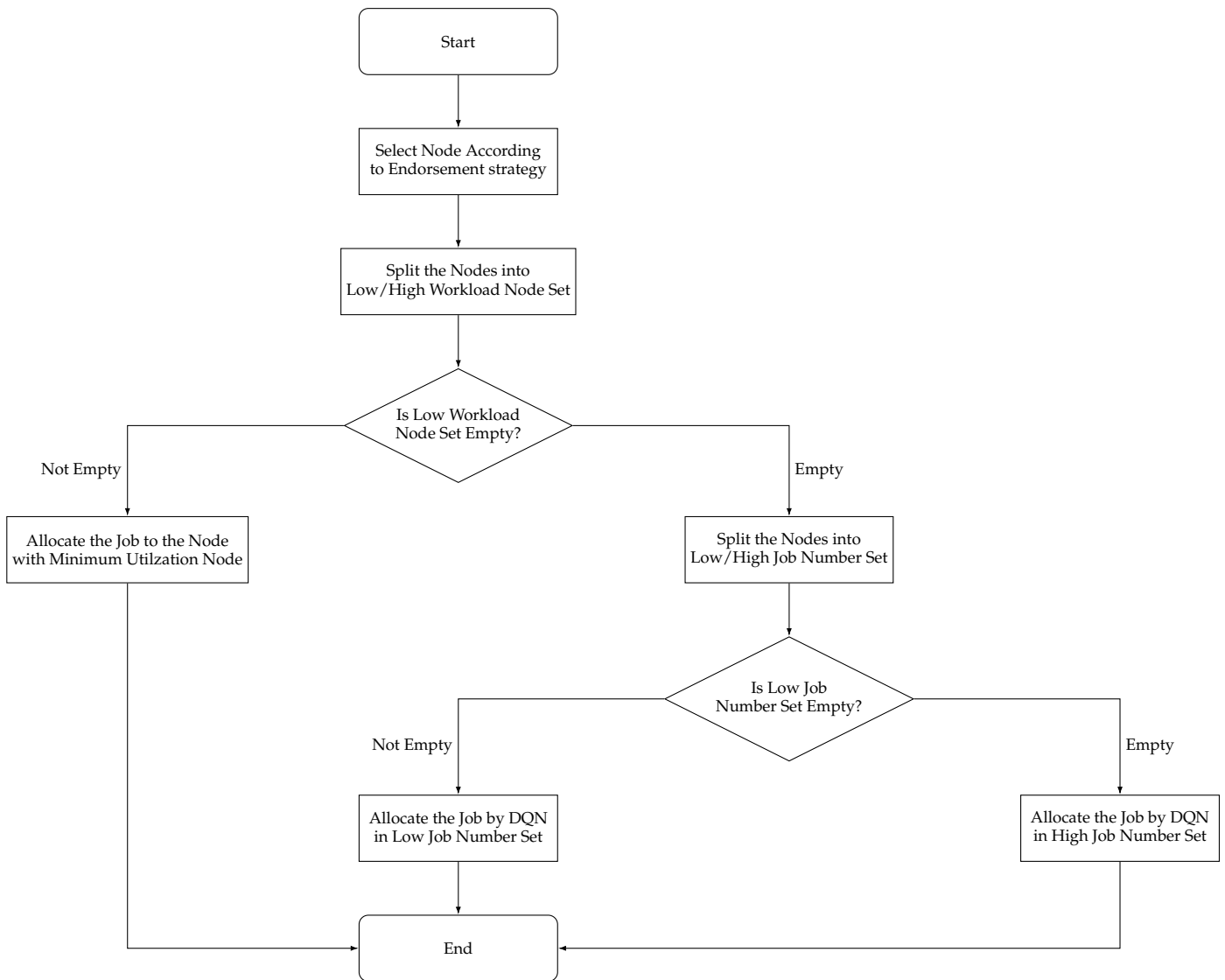


Figure 3. Allocation Process in Balanced-DRL.

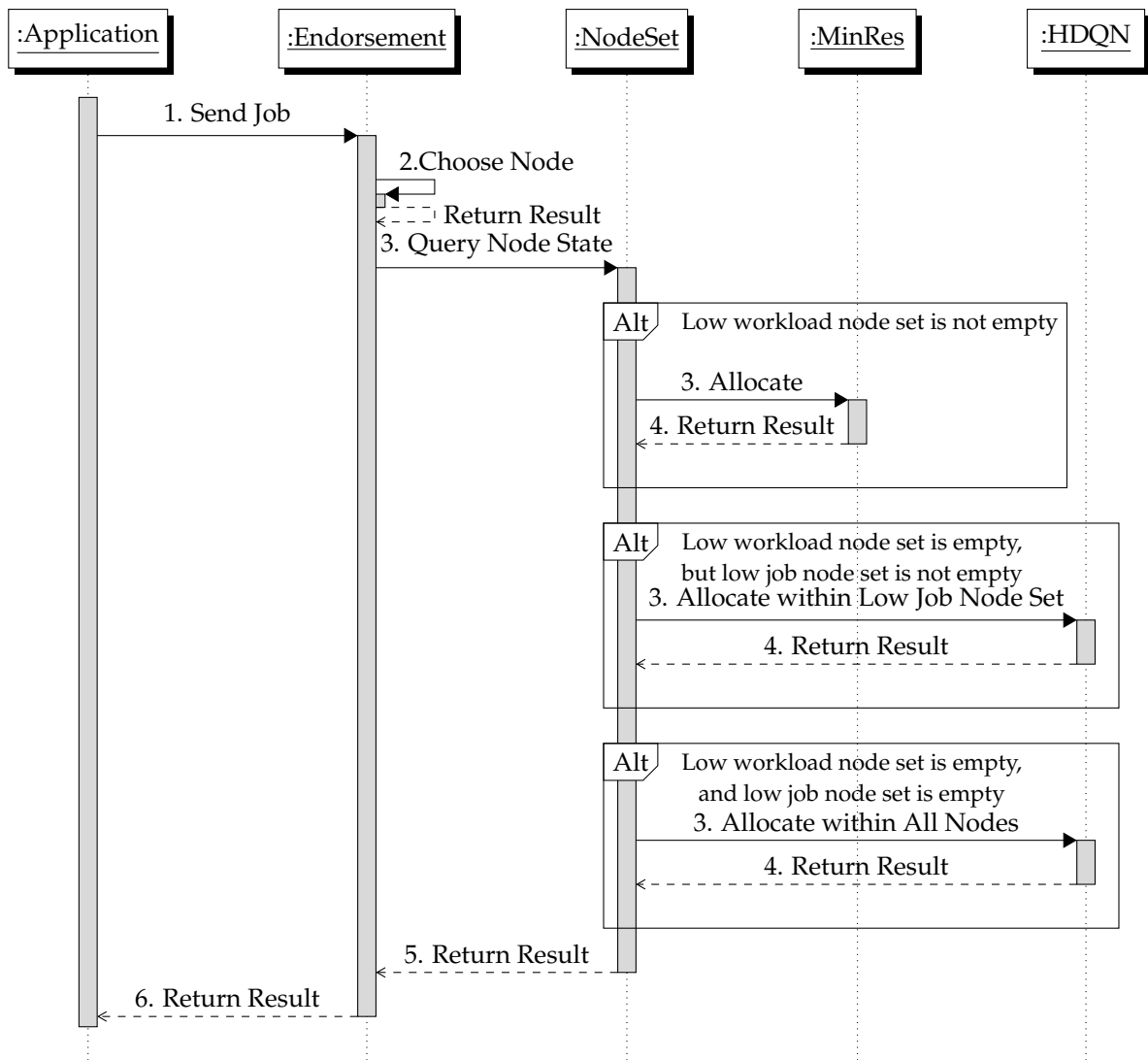


Figure 4. Sequence Diagram of Allocation Process.

For instance, let us consider a new job request, which can be allocated to any of the three nodes that satisfy its endorsement policy. Consider the following scenarios:

- If two of the nodes are low-load nodes, then the MinRes allocation strategy allocates the job to the node with the lowest resource utilization.
- If all three nodes are high-load nodes, but two of them have low job counts, then the HDQN allocation strategy allocates the task to the low job count nodes to achieve the highest overall resource utilization.
- If all three nodes have a high resource load and a high job count, then the HDQN allocation strategy is used to allocate among the three nodes to achieve the highest overall resource utilization.

4.3. Brief Summary

Section 4 first analyzes the impact of allocation strategies under different workload scenarios. In low-workload scenarios, a greedy allocation strategy is adopted directly. However, in both high-workload scenarios and time-varying-workload scenarios, the job allocation process is more complex. In high-workload scenarios, both resource utilization and job-load balancing should be considered. In time-varying-workload scenarios, different types of node partitions should be determined, and the current high- and low-workload scenarios should be identified to decide which strategy to use.

Furthermore, this section discusses the job allocation process of Balanced-DRL. If there are available low-workload nodes, the job is assigned to the node with the lowest resource utilization rate. Otherwise, we check whether there are nodes with low job numbers. If there are, the job is allocated to the node with the lowest number of jobs. If not, the DQN-based algorithm is used for job allocation.

5. Allocation Based on DQN

This section proposes a deep reinforcement learning-based allocation algorithm to solve the BaaS job allocation problem in the high-workload scenario, named HDQN. Firstly, the algorithm design, including the state space, the action space, and the reward function, is illustrated. Then, we focus on the training process of the deep Q-learning network (DQN), including network design and the Boltzmann action exploration strategy.

5.1. Algorithm Design

Given a BaaS job, j , and its validated peer nodes set, $\hat{\mathbf{P}}$, the state space of HDQN is denoted as \mathbb{S} , as shown in Equation (11). It is worth noting that we denote a valid peer node as $\hat{p} \in \hat{\mathbf{P}}$. The validated peer node set is obtained according to the endorsement strategy.

$$\mathbb{S} = \langle S(j), \langle \hat{R}(\hat{p}, \mathcal{T}), \langle R(\hat{p}, \mathcal{T}), \sum_{\hat{j} \in \hat{p}} S(\hat{j}) \rangle \rangle \quad \mathcal{T} \in \{\mathcal{C}, \mathcal{M}, \mathcal{D}\} \quad (11)$$

To fully describe job allocation steps, \mathbb{S} is presented by a connected vector including the following four components:

- $S(j)$ presents the scale of j obtained using code static analysis as discussed in Section 3.1. $S(j)$ includes the eight items shown in Table 1.
- $\hat{R}(\hat{p}, \mathcal{T})$ shows the remaining resources of peer node p . In our model, it is obtained as $(1 - U^t(\hat{p}, \mathcal{T})) \times R(\hat{p}, \mathcal{T}) \quad \mathcal{T} \in \{\mathcal{C}, \mathcal{M}, \mathcal{D}\}$. If a peer node is not a validated node according to the endorsement strategy, then we set the corresponding $\hat{R}(\hat{p}, \mathcal{T})$ to 0 to unify the length of the state.
- $R(\hat{p}, \mathcal{T})$ indicates the total resources of a valid peer node. To avoid heterogeneous clusters, the resources are indicated by their absolute amount rather than usage rate.
- $\sum_{\hat{j} \in \hat{p}} S(\hat{j})$ shows the total allocated job scale \hat{j} in peer node \hat{p} , which is calculated with the sum of all allocated jobs. However, $\sum_{\hat{j} \in \hat{p}} S(\hat{j})$ do not contain *ChainSize* and *WorldSize*, since they are included in $S(j)$.

The action space for assigning job j is denoted as \mathbb{A} . Since the allocation process can be treated as a selection of nodes, we have $\mathbb{A} = \hat{\mathbf{P}}$, where $\hat{p} \in \hat{\mathbf{P}}$ represents the action allocating job j to valid peer node \hat{p} . Obviously, the action space size equals the number of validated peer nodes.

The reward function of HDQN is defined in Equation (12). In the allocation process, if job j is allocated to peer node p^* , then we define the reward as the total resource utilization rate of p^* .

$$\mathcal{R} = U^t(p^*) \quad (12)$$

5.2. Training Process

DQN [50] is used to train an agent for gameplay, in which a convolution neural network is adopted to extract the features of input frames. The states are frame sequences, and the actions are game operations. HDQN deals with numerical features such as job scale and resource amount of peer nodes. Therefore, we adopt a simple fully connected network as our network architecture with the following changes:

- The training network of the original algorithm uses the convolutional neural network (CNN) to extract the input state, that is, the characteristics of a frame of picture. The training network of HDQN uses the fully connected depth neural network to process the state.

- The original algorithm uses multiple continuous picture frames and action sequences as the state, because a single frame cannot describe the continuous change. In contrast, the state used in HDQN can describe the job scale and all node resources without using sequences.
- The original algorithm divides training into multiple episodes for the fragment job in the training end state. However, HDQN involves the non-termination state and continuous job allocation, so the total step count of training termination is artificially specified.
- The original algorithm uses the ϵ -greedy strategy, while HDQN uses the Boltzmann strategy to improve the exploratory action.

5.3. Boltzmann Action Exploration Strategy

Traditional deep reinforcement learning algorithms such as DQN [51] generally use the ϵ -greedy strategy to select actions. In the training process, the ϵ -greedy strategy selects the action with the highest Q value with the probability of $1 - \epsilon$ and randomly selects the action with the probability of ϵ to maintain action exploration. ϵ often takes a smaller value.

$$\pi(a | s, \theta) = \frac{e^{Q(s,a,\theta)}}{\sum_{i=1}^n e^{Q(s,a_i,\theta)}} \tag{13}$$

$$a = \begin{cases} a_1, & \text{with } \pi(a_1 | s, \theta) & \text{prob} \\ a_2, & \text{with } \pi(a_2 | s, \theta) & \text{prob} \\ \dots & & \\ a_n, & \text{with } \pi(a_n | s, \theta) & \text{prob} \end{cases} \tag{14}$$

In our implementation, ϵ -greedy might cause two problems: 1. At the beginning of training, the cluster utilization rate might increase slowly. 2. The training process might have poor convergence. Therefore, we introduce the Boltzmann strategy. To train with the Boltzmann strategy, the probability of selecting each action is calculated using the exponential proportion of the Q value of each action to the sum of the Q values, as shown in Equation (13).

For problem 1, ϵ -greedy assigns jobs to the node corresponding to the highest Q value during most of the beginning of training, leading to a rapid increase in the utilization of the node and making ϵ -greedy still assign more jobs to the node next time to improve the utilization and obtain the maximum reward value, while other nodes are starving. It is difficult for these nodes to allocate jobs to improve overall utilization. Of course, with the continuous arrival of a high workload, most nodes will be in high utilization after a period of time, but the early utilization is slow, and the improvement still affects the training effect to a certain extent.

For problem 2, similarly, the more balanced the resources of each node are, the faster the training Q value converges. The convergence of ϵ -greedy is worse. On the contrary, the Boltzmann strategy allocates actions according to the probability corresponding to the proportion of Q value, and each action is executed with a certain probability, which increases the action selectivity and has relatively better convergence.

5.4. Complexity Analysis

In the HDQN, a fully connected network is used. Let Z_n denote the number of neurons in the n^{th} layer of the neural network. The computational complexity of the n^{th} layer is $O(Z_{n-1}Z_n + Z_nZ_{n+1})$. For a fully connected network with N layers, the computational complexity is $O(\sum_{n=2}^{N-1}(Z_{n-1}Z_n + Z_nZ_{n+1}))$. Therefore, the computational complexity of the proposed algorithm during training is $O(H \times T \times K(\sum_{n=2}^{N-1}(Z_{n-1}Z_n + Z_nZ_{n+1})))$. Here, H , T , and K represent the mini-batch size, the number of training steps, and the maximum episode, respectively. The computational complexity during inference is $O(T \times K(\sum_{n=2}^{N-1}(Z_{n-1}Z_n + Z_nZ_{n+1})))$.

5.5. Brief Summary

Section 5 first designs the state space, action space, and reward function of the HDQN algorithm. Then, in the training process of HDQN, improvements are made to the network structure, input data, training termination conditions, and exploration strategy based on the original DQN. Finally, by comparing and analyzing the greedy strategy and the Boltzmann strategy, the advantages of the Boltzmann strategy in this project are demonstrated.

6. Experiment

The optimization of this experiment lies in the job allocation algorithm of the BaaS platform. Conducting simulation experiments allows the experimental results to focus on the impact of the allocation algorithm, rather than being affected by other factors, such as physical equipment, blockchain, and network. Therefore, we adopted simulation experiments.

The first step of this experiment was to determine the relationship between job workload, and resource utilization, throughput, and latency, and to identify the boundary between high and low workload under a given configuration, to facilitate testing the algorithm's performance under different workloads and stages. Next, the training model was obtained. Finally, by comparing with different allocation algorithms, we demonstrated the optimization effect of Balanced-DRL on throughput and latency.

This section first introduces the simulation experiment configuration and then provides the design of experiments on simulated job workload, allocation model training, fixed low job workload, fixed high job workload, and time-varying job workload. In the end, we show our results and analysis.

6.1. Experimental Environment

The default configuration considered a single-channel homogeneous cluster to facilitate the experiment. Single channel refers to a cluster where Hyperledger Fabric contains only one channel, and homogeneous means that each physical machine and virtual machine in the cluster has the same amount of resources. The default simulation parameters for this section's BaaS platform are shown in Table 3.

Table 3. Parameters for homogeneous BaaS platform with single channel.

Class	Item	Value	Description
Platform configuration	Physical machine	50	Total number of physical machines in the cluster
	Resource of physical machine	< 10, 10, 10 >	Resource configuration of the physical machine
	Virtual machine	50	Total number of virtual machines in the cluster
	Resource of virtual machine	< 10, 10, 10 >	Resource configuration of the virtual machine
	Maximum containers	15	Maximum number of containers in a virtual machine
Hyperledger	Organizations	5	Total number of organizations
	Channels	1	Total number of channels
	Peers	50	Total number of peer nodes
	Chaincodes	20	Total number of chaincodes
Job workload	λ	0.5–4.0	Size of job workload under Poisson distribution.
	Jobs	30,000	Total number of jobs
	Tick	0.01 s	Interval for the simulation platform to process jobs
Threshold	Resource-load threshold	90%	Threshold of resource load
	Job-load threshold	50%	Threshold of job load

The cluster consisted of 50 homogeneous physical machines, and the CPU, memory, and disk resources of each physical machine were the same and equal to 10. Each physical machine contained only one virtual machine, so there were 50 virtual machines with the same configuration in the cluster. The Hyperledger Fabric network contained one channel, which was participated by 5 organizations, and each organization had 10 peer nodes. Therefore, the network had 50 peer nodes, and exactly one virtual machine/physical machine could be used by each peer node. The input job workload included 30,000 BaaS job requests, which randomly called 20 chaincodes defined by the above channel, and the requests arrived with fixed workload sizes according to $\lambda = 0.5, 1.0, \dots, 4.0$. λ represents the number of jobs arriving on average within 0.01 s. The resource workload threshold was set to 90%, and the nodes with a resource workload greater than 90% were considered high-resource-load nodes. The job-workload threshold was set to 50%, and the nodes with a job workload greater than 50% were considered low-resource-workload nodes.

6.2. Experiment Design

We generally have 3 types of experiments: “Job WorkLoad Experiment”, “Allocation Model Training Experiment”, and “Experiments under Different Workload Scenarios”. In this section, we illustrate the experiment design in detail.

6.2.1. Job-Workload Experiment

The purpose of this experiment is to investigate the changes in the resource utilization, job throughput, and job latency of a BaaS platform with varying job workloads. Specifically, the experiment uses a job-workload generation module to generate jobs, following a Poisson distribution with a parameter λ . The job-workload generation module randomly generates the channel, chaincode, function, and arrival time of job requests. In the experiment, different job workloads are generated by adjusting the λ parameter. The changes in cluster utilization, job throughput, and job latency with the size of the job workload are observed.

6.2.2. Allocation Model Training

In a high-workload scenario, we adopt HDQN as our allocation method. We use the Boltzmann action exploration strategy to train the model. In this section, we compare the training process with the Boltzmann action exploration strategy and the training process with the ϵ -greedy strategy. The neural network modules are fully connected networks with ReLu activation functions and MES loss functions. The reinforcement learning parameters are given in Table 4.

Table 4. Parameters for DQN training.

Symbol	Value	Description
α	0.01	Learning rate for updating value
γ	0.9	Reward discount
$\omega_C \omega_M \omega_D$	1/3	The weight of each resource in the reward
Batch size	32	Size of mini-batch during training
Learning rate	0.001	Learning rate for updating network
ϵ	0.1	The probability of randomly selecting actions under ϵ -greedy
R	1000	The capacity of the experience replay buffer
C	50	The interval steps for updating the target network

6.2.3. Experiments under Different Workload Scenarios

We conducted simulation comparison experiments in three different scenarios, i.e., fixed low job workload, fixed high job workload, and time-varying job workload, by comparing Balanced-DRL (Ours), and random allocation (R), round-robin allocation (RR), and minimum resource allocation (MR). The performance of the algorithms in cluster utilization, job throughput, job latency, and load balancing were measured.

R randomly selects a node from the candidate nodes to allocate the job. RR distributes job requests to candidate nodes in turn. MR allocates job requests to the candidate node with the lowest resource utilization.

For the fixed workload experiment, we used the platform configuration based on Table 3. For the time-varying-workload experiment, to simulate a more realistic scenario, we conducted comparison experiments on a multi-channel heterogeneous BaaS cluster. Table 5 provides the parameter configuration of the BaaS platform.

Table 5. Parameters for heterogeneous BaaS Platform with multiple channels.

Class	Item	Value	Description
Platform configuration	Physical machine	60	Total number of physical machines in the cluster
	Resource of physical machine	< 15, 10, 10 >	Resource configuration of the physical machine
		< 10, 15, 10 >	
		< 10, 10, 15 >	
		< 15, 15, 15 >	
Virtual machine	120	Total number of virtual machines in the cluster	
Resource of virtual machine	< 10, 10, 10 >	Resource configuration of the virtual machine	
	Maximum containers		15
Hyperledger	Organizations	20	Total number of organizations
	Channels	5	Total number of channels
	Peers	120	Total number of peer nodes
	Chaincodes	20	Total number of chaincodes
Job workload	λ	Time varying	Size of job workload under Poisson distribution
	Jobs	111,150	Total number of jobs
	Tick	0.01 s	Interval for the simulation platform to process jobs
Threshold	Resource-load threshold	90%	Threshold of resource load
	Job-load threshold	50%	Threshold of job load

As shown in Table 5, the platform consisted of 60 heterogeneous physical machines, with 4 types of resource configurations, i.e., <15, 10, 10>, <10, 15, 10>, <10, 10, 15>, and <15, 15, 15>, with 15 physical machines for each configuration. Each physical machine hosted 2 virtual machines that occupied half of the resources, so there were a total of 120 virtual machines in the cluster. Hyperledger Fabric included 5 channels, each with 4 organizations, and each organization had 6 peer nodes so that each node could be assigned exactly one virtual machine.

In addition to the above configuration, the time-varying job workload refers to the Alibaba cluster-trace V2017 dataset generated and open-sourced on GitHub (<https://github.com/alibaba/clusterdata>, accessed on 10 May 2023). This dataset records the job instance requests and resource changes on 1300 physical machines in the cluster over a 24 h period. In this experiment, according to the job instance workload shown in Figure 5, the time-varying workload of BaaS jobs was generated by adjusting the value of λ . This generated workload contained 110,000 job requests, which made it more fine-grained than the original real workload while also ensuring the coverage of the corresponding high- and low-workload sizes for the multi-channel heterogeneous cluster.

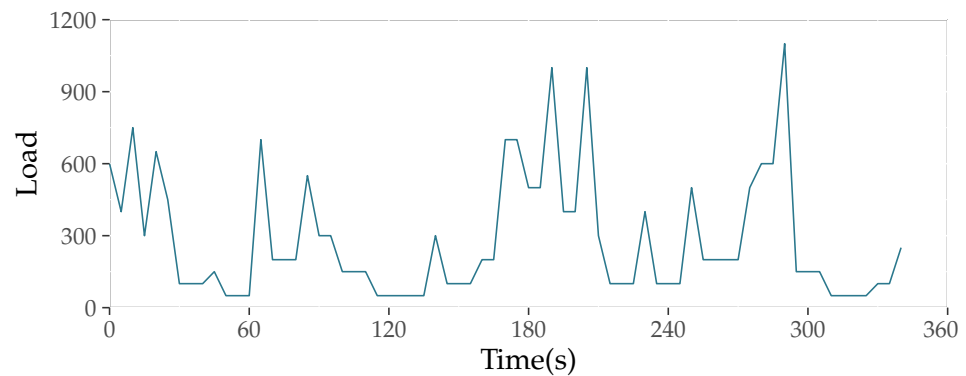


Figure 5. Time-Varying Job Workload Generated from Real Traces.

6.3. Experimental Results

Regarding the experiment design, this section presents the experimental results of the job-workload experiment, allocation model training, fixed-low-job-workload experiment, fixed-high-job-workload experiment, and time-varying-job-workload experiment and analyzes the reasons for the corresponding results.

6.3.1. BaaS Job-Workload Experiment

The input job workload consisted of 30,000 BaaS job requests, which randomly invoked the chaincode on the platform, and the requests arrived with fixed workload sizes of $\lambda = 0.5, 1.0, \dots, 4.0$. Below, Figure 6 shows the changes in utilization, throughput, and latency with workload under default random job allocation.

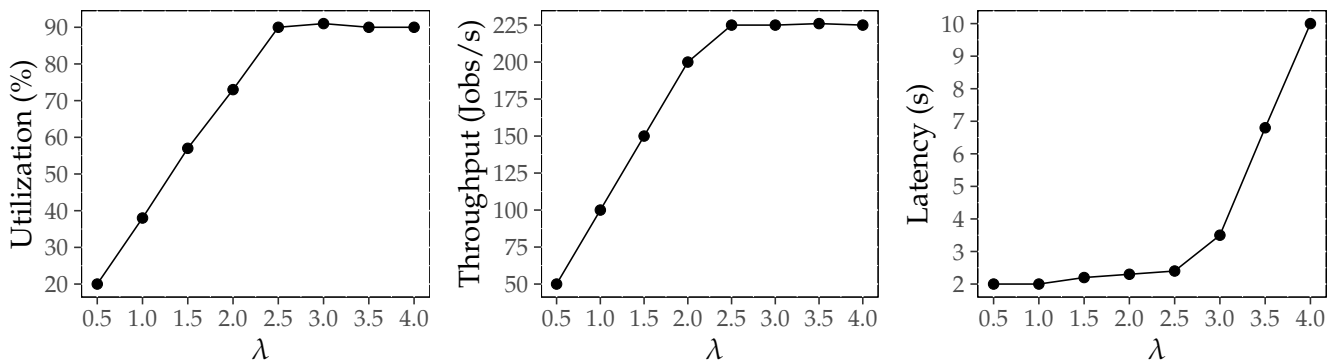


Figure 6. Metrics of BaaS Changing with Job-Workload Factor λ .

From Figure 6, we can see that cluster utilization, job throughput, and job latency all exhibited different changes around $\lambda = 2.5$. Therefore, for the BaaS configuration, when the job-workload factor $\lambda < 2.5$, it was considered a low job workload, while when the job-workload factor $\lambda \geq 2.5$, it was a high job workload. Regarding utilization, it linearly increased with the increase in workload under low workloads. At the same time, it grew very slowly. It never approached 100% under high workloads, indicating that the BaaS could not fully utilize the resources due to internal resource fragmentation under a high workload. Throughput increased linearly with the increase in workload under low workloads, but it did not increase with the increase in workload under high workloads, maintaining a maximum throughput of around 230 Job/s. Regarding latency, job completion latency was smaller under low workloads, but it significantly increased as it approached high workloads and always increased with the increase in workload. In addition, RR and MR also had similar results, all changing around $\lambda = 2.5$.

Overall, this experiment shows that the utilization and throughput of the BaaS cluster increased with the increase in workload, while job latency decreased with the decrease

in workload. In addition, the BaaS job workload could be clearly divided into low and high workloads, and $\lambda = 2.5$ could be measured as the boundary between high and low workloads for this simulation configuration.

6.3.2. Allocation Model Training

As mentioned in the previous section, deep reinforcement learning only trains job allocation strategies when nodes are at high utilization, so training used high job workloads at $\lambda = 2.5, 3.0, 3.5,$ and 4.0 as inputs. Figure 7 shows the loss curve of two training processes based on Boltzmann and ϵ -greedy policies with $\lambda = 3$. The training loss of Boltzmann converges faster than that of ϵ -greedy. This is because the Boltzmann strategy has stronger action selection, considers more nodes during allocation training, and learns more interaction data between jobs and node allocation.

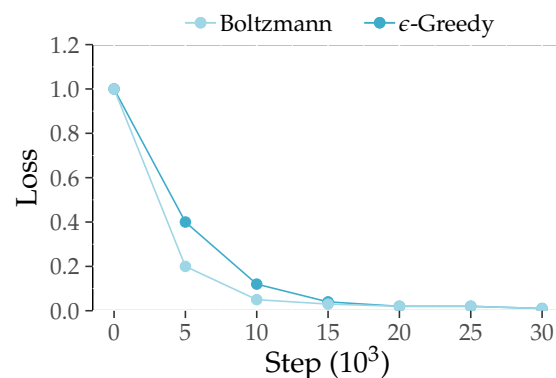


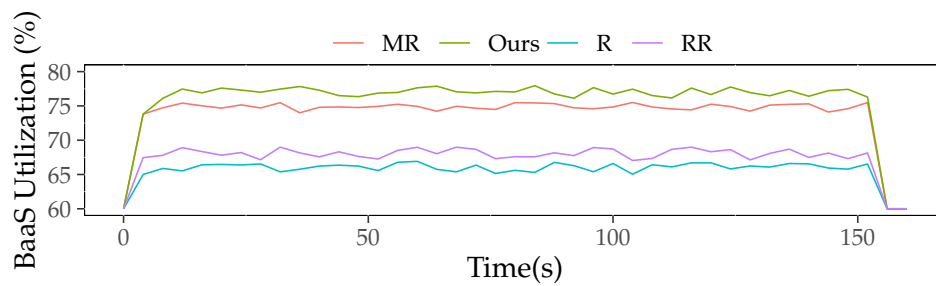
Figure 7. Training Loss of Balanced-DRL.

6.3.3. Fixed-Low-Workload Experiment

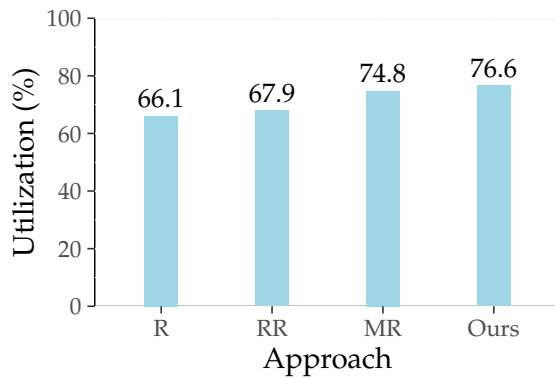
This experiment compared the relative performance of R, RR, MR, and Balanced-DRL (Ours) under a low job workload ($\lambda = 2$). Figure 8a shows the changes in utilization of these allocation algorithms during the simulation process.

As shown in Figure 8a, the BaaS utilization of each algorithm was not high when handling low job workloads, mainly due to the influence of job size. Specifically, the average utilization rates of Balanced-DRL and MR were around 77% and 75%, respectively, which were significantly higher than the 66% and 68% rates of R and RR. As mentioned in the previous section, Balanced-DRL, similarly to the MR algorithm, allocates jobs to nodes with the lowest utilization under low job workloads. This not only maintains resource-load balancing among nodes but also increases cluster utilization by consuming the resources of low-utilization nodes. The reason why the Balanced-DRL algorithm has a slightly higher utilization rate than the MR algorithm is that in low-workload scenarios, there are still nodes with high resource loads, which leads to the adoption of the HDQN algorithm for job allocation.

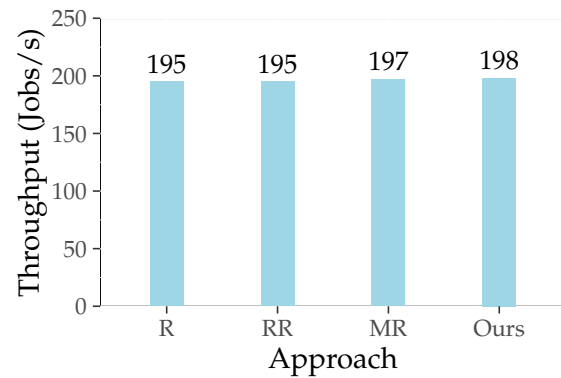
To further validate the above content and eliminate the influence of randomness, we performed a significance analysis of the experimental results. We assumed that the resource utilization rates of different algorithms followed a normal distribution and analyzed the significance of the results between three other algorithms and Balanced-DRL. Since all four algorithms were executed in the same environment, it is reasonable to assume that the four normal distributions had the same variance. This problem was transformed into a hypothesis-testing problem for the means of two normal populations with equal variances. The p -values for the RR, R, and MR algorithms compared with Balanced-DRL were 1.19×10^{-16} , 4.11×10^{-22} , and 0.042, respectively. It can be seen that the proposed algorithm has significant differences from the RR and R algorithms. The p -value with respect to the MR algorithm was less than 0.05, indicating that there was still statistical significance.



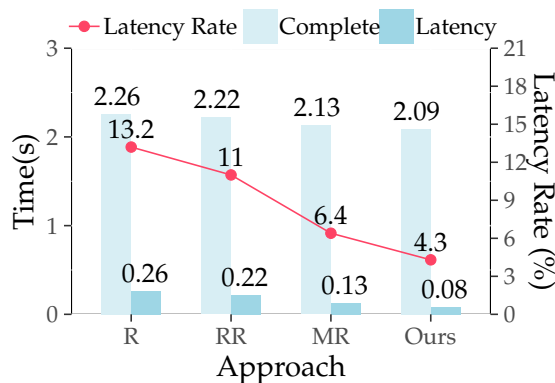
(a) Utilization Rate of BaaS under Low Job Workload



(b) Utilization Comparison



(c) Throughput Comparison



(d) Latency Comparison



(e) Balance Indicator Comparison

Figure 8. Comparison Results under Low Job Workload.

Figure 8b–e summarize the simulation results of each algorithm when handling low job workloads.

According to Figure 8c, the total job throughput of each algorithm was close to 200 Job/s, which was determined by the job-workload size under low job workloads. According to Figure 8d, the average job duration generated by the simulation system was 2 s, and the average job completion time of each algorithm was greater than 2 s. The latency indicators of Balanced-DRL and MR were smaller than those of the other two algorithms. This is because the former two algorithms always allocate jobs to nodes with the lowest utilization to satisfy the job’s required resources as much as possible and avoid increasing latency. In a fixed-low-load scenario, the Balanced-DRL algorithm uses the MinRes algorithm to allocate tasks. Hence, the time cost of task allocation is very small and can be considered negligible, similar to the other three algorithms.

According to Figure 8e, the resource allocation was more balanced for Balanced-DRL and MR, while the job workload was more balanced for RR.

This experiment compared the performance of various allocation algorithms in handling low job workloads on a BaaS platform. The results show that the utilization of Balanced-DRL (77%) was higher than that of R and RR (66% and 68%, respectively), because the former algorithm effectively utilizes the resources of each node by ensuring resource-load balancing, thus increasing cluster utilization. In addition, the job completion time of Balanced-DRL was also reduced by 6% and 8% compared with that of R and RR. This is also due to the resource-load balancing mechanism of the Balanced-DRL algorithm, which avoids latency caused by overloaded nodes.

6.3.4. Fixed-High-Workload Experiment

This experiment compared the performance of four algorithms under a fixed high job workload ($\lambda = 4$). Figure 9a presents the utilization rates for each algorithm. As shown in Figure 9a, compared with the low job workload, the utilization rates of each algorithm significantly increased under the high workload and remained above 90%. At this point, the difference in utilization rates is related to the allocation algorithms. Specifically, the average utilization rate of Balanced-DRL was approximately 97%, while the utilization rates of RR, R, and MR were 93%, 91%, and 91%, respectively. We had a utilization rate 4% to 6% higher than that of the other algorithms. Balanced-DRL uses a Q-network trained by DQN under a high workload to reduce resource fragmentation and improve utilization rates.

To avoid the influence of randomness on the experimental results, a significance analysis was performed on the low-workload scenarios. After calculation, the p -values for RR, R, and MR algorithms compared with Balanced-DRL were 4.41×10^{-5} , 2.62×10^{-8} , and 4.18×10^{-10} , respectively. Therefore, statistically, the proposed algorithm in this paper is indeed superior to the other three algorithms and not affected by random factors.

According to Figure 9b, the average utilization rate of Balanced-DRL remained significantly higher than that of other allocation algorithms.

According to Figure 9c, for a high job workload of $\lambda = 4$, namely, an average of 400 Jobs/s, Balanced-DRL achieved a maximum throughput of 244 Job/s, which was 5% to 8% higher than the maximum throughput of other algorithms.

According to Figure 9d, due to the job workload exceeding the cluster's processing capacity, job requests accumulated continuously, causing the average job completion time to increase from over 2 s under low workloads to 8–11 s under high workloads. Nevertheless, the average job completion time of Balanced-DRL was still reduced by 5% to 21% compared with other algorithms.

During the task allocation process using the HDQN algorithm, the actual time consumption is on the millisecond level. Compared with the degree of reduced latency, the time consumed by the allocation algorithm can be considered negligible.

According to Figure 9e, the node resource workloads for all algorithms were relatively balanced. In contrast, the job workloads on nodes for Balanced-DRL were more evenly distributed than for other algorithms.

The results show that the utilization rate of Balanced-DRL was approximately 4% to 6% higher than that of other algorithms, with an average utilization rate of 97%, due to using HDQN to reduce node resource fragmentation and improve BaaS utilization. Correspondingly, the higher utilization rate led to a 5% to 8% increase in throughput for Balanced-DRL. Latency for Balanced-DRL was significantly reduced by 5% to 21%, partly due to a higher utilization rate, which allowed jobs to be allocated more resources and be executed more quickly, and partly due to job-workload balancing, which prevented excessive job latency on certain nodes. Finally, although R and RR also consider job-workload balancing, their job balancing factors are inferior to those of Balanced-DRL, as they achieve balance from a distribution perspective. In contrast, Balanced-DRL achieved balance from a job quantity perspective based on node job workloads.

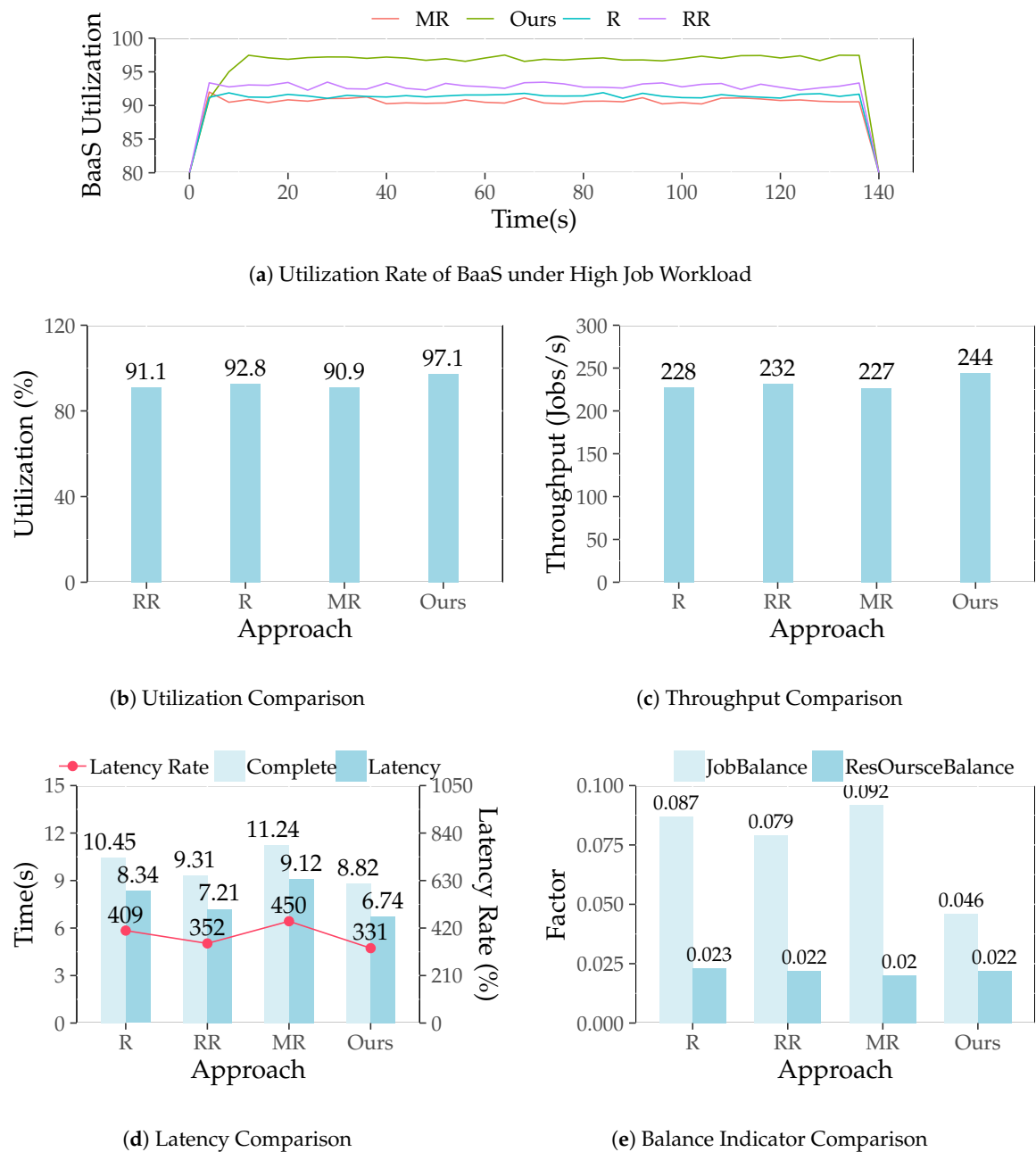
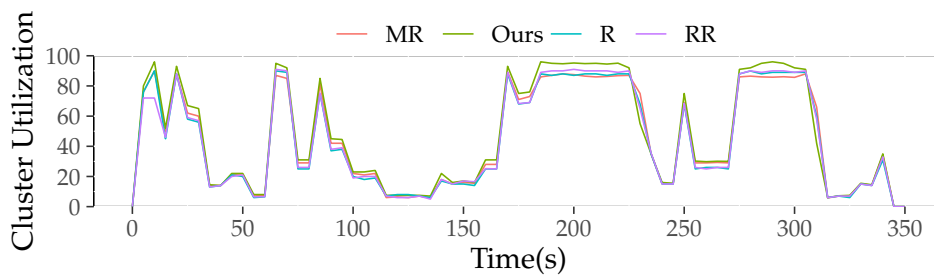


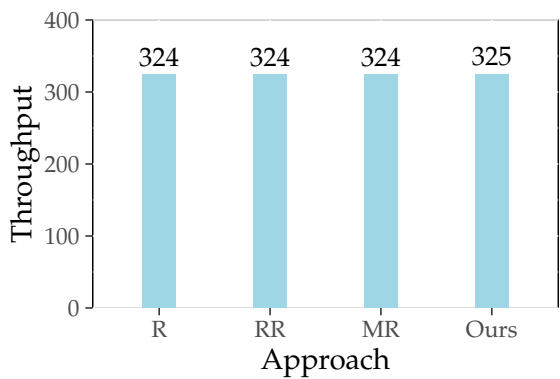
Figure 9. Comparison Results under High Job Workload.

6.3.5. Time-Varying-Workload Experiment

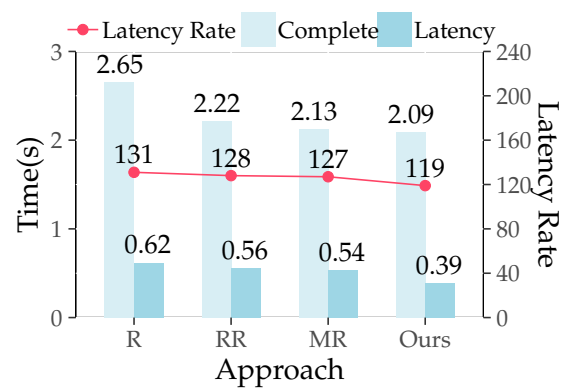
Based on the time-varying platform configuration and job workload mentioned in the design, Figure 10a presents the utilization rate changes for each algorithm. The utilization rates for each algorithm exhibited consistent changes with the time-varying workload, as shown in Figure 5. Under high job workloads, Balanced-DRL achieved a cluster utilization rate of 96%, which was significantly higher than the utilization rates of other algorithms, which ranged from 87% to 91%. Under low job workloads, the utilization rates for each algorithm were similar, with Balanced-DRL and MR having slightly higher utilization rates than the other two algorithms. These results are consistent with those of fixed low and fixed high job workloads in the previous experiments. Overall, in the process of handling time-varying job workloads, Balanced-DRL consistently maintained the highest resource utilization rate.



(a) Utilization Rate of BaaS under Time-Varying Job Workload



(b) Utilization Comparison



(c) Throughput Comparison

Figure 10. Comparison Results under Time-Varying Job Workload.

In the time-varying-workload scenarios, besides random factors, the factors that cause changes in resource utilization rates include changes in job workload. Therefore, it is impossible to assume that the changes in resource utilization rates follow a normal distribution, and hypothesis testing cannot be used for significance analysis. At the same time, the significance analysis of low and high job workloads is analyzed in the previous section, and the scenario of time-varying job workloads combines these two scenarios. Therefore, significance analysis was not performed in this case.

Figure 10b,c present the running results of each job under time-varying job workloads, including only the job total throughput and latency results and not the average utilization rate and load factor, as the latter two vary significantly with time and have little meaning in terms of their average values.

From Figure 10c, it can be seen that the throughput for different algorithms was similar, as most of the time, in the time-varying job workload scenario, we had low job workloads, similar to the real workload. Therefore, the total time required for the cluster to process the job workload was also similar.

From Figure 10b, it can be seen that all latency indicators of Balanced-DRL were smaller than those for other algorithms, indicating that under time-varying workloads, Balanced-DRL has a certain optimization effect on job completion latency.

This experiment compared the performance of various allocation algorithms for BaaS clusters under time-varying job workloads. The experimental results show that the utilization rate and latency indicators of Balanced-DRL were superior to those of other algorithms. Obviously, since any time-varying job workload can be viewed as a sequence of high and low job workloads over time and the indicators of Balanced-DRL under high and low job loads were superior to those of other allocation algorithms, this algorithm can achieve certain optimizations for any job workload overall.

7. Discussion

In this section, we first discuss the experimental results presented in the paper. Furthermore, a comparison is made between the algorithm proposed in this paper and some representative algorithms from related work.

7.1. Experimental Results Discussion

Balanced-DRL, the job allocation algorithm proposed in this paper, can achieve corresponding optimization of the resource utilization rate handling various job workloads, thereby improving job throughput and reducing job latency compared with the R, RR, and MR algorithms. Under the given cluster configuration, with low job workloads, the Balanced-DRL algorithm could improve the utilization rate by about 10% and reduce latency by 6% to 8% compared with the R and RR algorithms. Under high job workloads, the algorithm could increase the utilization rate by 4% to 6%, reduce job latency by 5% to 21%, and increase the maximum throughput by 5% to 8% compared with other algorithms. Under time-varying workloads, the algorithm could reduce job latency by 6% to 10% compared with other algorithms.

7.2. Comparison Discussion

The comparison between Balanced-DRL and some typical works in related research in terms of field, problem type, approach, and achievements is shown in Table 6.

Table 6. Comparison between Balanced-DRL and Related Works: O = Optimization; F = Framework; CM = Consensus Mechanism.

Ref.	Field	Problem Type	Approach	Objective	Performance Metric	Dataset
[10]	Cloud computing	O	Bin packing	Latency and energy consumption	Resource utilization Time Energy	Private
[16]		O	Heuristic	Latency and energy consumption	Resource utilization Time Energy	Randomly generated
[19]		O	Machine learning	Latency and Energy consumption	Resource utilization Time Energy	Randomly generated
[22]		O	Reinforcement learning	Latency and energy consumption	Time Energy	Randomly generated
[32]	Block-chain	O	Topology optimization	Latency	Time	Simulated
[39]		O	Data-layer optimization	Throughput	Time	Simulated
[37]		O	Multi-leader design	Throughput and scalability	CPU Cost Network traffic	Simulated
[35]		O	Consensus mechanism design	Throughput	Throughput	Simulated
[46]	BaaS	F	Architecture design	Scalability and usability	-	-
[47]		CM	Machine learning	Usability	TPS	Simulated
[41]		F	Architecture design	Reliability	Throughput	-
Ours		O	Reinforcement learning	Throughput and latency	Resource utilization Time Throughput	Generated from Alibaba cluster-trace dataset

The papers [10,16,19,22] focus on optimizing job allocation in cloud computing platforms. Each paper proposes optimization algorithms with varying key performance indicators, such as latency and energy consumption, while employing different types of algorithms. Most of these optimization algorithms optimize performance metrics such as resource utilization, time, and energy consumption on randomly generated datasets. However, we cannot utilize these research results on the BaaS platform due to the dynamic nature of BaaS jobs and the BaaS environment. Firstly, the resource requirements of BaaS jobs fluctuate and have to comply with the endorsement policies of blockchain. Additionally, containers are deployed in BaaS rather than virtual machines in a cloud environment, in which resource occupation and resource release are more flexible. To address these challenges, this paper leverages reinforcement learning algorithms, which possess strong flexibility and adaptability.

The papers [32,35,37,39] conduct research on blockchain performance optimization to improve throughput or reduce latency. Their focus mainly lies on blockchain itself, such as enhancing the data layer and modifying the consensus mechanism. Their experiments were mainly performed using simulated data, and performance improvements were measured using time or throughput. Our work, however, concentrates on the BaaS platform that deploys blockchain and cloud computing to provide easy access to blockchain-related applications. Rather than improving the blockchain, we optimize the job allocation process in BaaS. Since we still deploy a blockchain in BaaS, our proposal is compatible with these works.

Currently, research related to BaaS [41,46,47] places more emphasis on the usability, convenience, and reliability of the BaaS platform. Most of these research works optimize the platform architecture or job execution process to enable BaaS to be applied in scenarios such as the Internet of Things and big data analysis. Although we also concentrate on the BaaS platform, we differ from them in that we do not modify the BaaS architecture. Instead, we only focus on the job allocation process.

In conclusion, our proposal focuses on optimizing BaaS performance by improving job throughput and reducing latency in the BaaS job allocation process. Our main contribution is proposing the Balanced-DRL algorithm based on DQN, which can optimize the performance of BaaS without affecting the blockchain itself. The experimental dataset used in this paper was generated from real datasets, which have higher credibility. Regarding resource utilization, the latency, throughput, and performance of Balanced-DRL are improved compared with traditional algorithms. The basic idea of Balanced-DRL was inspired by job allocation optimization in cloud computing. Optimization methods developed for blockchain and BaaS are compatible with our proposal and can further improve BaaS performance.

8. Conclusions

The contributions of this paper are as follows: (1) We modeled the allocation problem of BaaS by analyzing Hyperledger Fabric and its job execution process. (2) We summarized the characteristics of BaaS jobs by analyzing the job chaincode. (3) We propose a BaaS job allocation algorithm, Balanced-DRL, which adopts different allocation strategies for different job workloads. (4) Our experiments demonstrate that the proposed Balanced-DRL algorithm can effectively improve the utilization of BaaS when processing jobs, ensure load balancing, increase job throughput, and reduce job latency compared with other algorithms, such as random allocation, round-robin allocation, and minimum resource allocation.

One limitation of this paper is using a rule-based minimum resource allocation strategy for low-job-workload scenarios. In future work, we plan to adopt a deep reinforcement learning algorithm that does not distinguish among job workloads to satisfy all application scenarios. Additionally, we only consider performance optimization in the paper. We want to introduce an energy consumption optimization objective or an energy-efficiency objective in future work.

Author Contributions: Conceptualization, C.G., M.X., S.H. and J.S.; data collection, C.G. and M.X.; formal analysis, C.G., M.X., S.H. and J.S.; investigation, M.X. and S.H.; methodology, C.G. and M.X.; project administration, C.G.; resources, C.G. and M.X.; validation, S.H.; visualization, M.X. and S.H.; writing—original draft, C.G. and M.X.; writing—review and editing, C.G. and J.S. All authors have read and agreed to the published version of the manuscript.

Funding: This paper was supported by Fundamental Research Funds for the Central Universities (No. N2317005).

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

Abbreviation	Full Name
BaaS	Blockchain as a Service
DRL	Deep reinforcement learning
QoS	Quality of service
VM	Virtual machine
DQN	Deep Q-learning network
R	Random
RR	Round-robin
MR	Minimum resource
Ours	Balanced-DRL

References

- Song, J.; Zhang, P.; Alkubati, M.; Bao, Y.; Yu, G. Research Advances on Blockchain-as-a-Service: Architectures, Applications and Challenges. *Digit. Commun. Netw.* **2021**, *8*, 455–475. [\[CrossRef\]](#)
- Alshurafa, S.M.; Eleyan, D.; Eleyan, A. A Survey Paper on Blockchain as a Service Platforms. *Int. J. High Perform. Comput. Netw.* **2021**, *17*, 8. [\[CrossRef\]](#)
- Sahal, R.; Alsamhi, S.H.; Brown, K.N.; O’Shea, D.; McCarthy, C.; Guizani, M. Blockchain-Empowered Digital Twins Collaboration: Smart Transportation Use Case. *Machines* **2021**, *9*, 193. [\[CrossRef\]](#)
- Chuang, I.H.; Chiang, S.H.; Chao, W.C.; Huang, S.H.; Zeng, B.L.; Kuo, Y.H. A Hierarchical Blockchain-based Data Service Platform in MEC Environments. In Proceedings of the 2nd International Conference on Blockchain Technology, Hilo, HI, USA, 12–14 March 2020; pp. 95–99. [\[CrossRef\]](#)
- Lu, Q.; Xu, X.; Liu, Y.; Weber, I.; Zhu, L.; Zhang, W. uBaaS: A Unified Blockchain as a Service Platform. *Future Gener. Comput. Syst.* **2019**, *101*, 564–575. [\[CrossRef\]](#)
- Kuzlu, M.; Pipattanasomporn, M.; Gurses, L.; Rahman, S. Performance Analysis of a Hyperledger Fabric Blockchain Framework: Throughput, Latency and Scalability. In Proceedings of the 2019 IEEE International Conference on Blockchain, Atlanta, GA, USA, 14–17 July 2019; pp. 536–540. [\[CrossRef\]](#)
- Hassan, W.; Chou, T.S.; Tamer, O.; Pickard, J.; Appiah-Kubi, P.; Pagliari, L. Cloud Computing Survey on Services, Enhancements and Challenges in the Era of Machine Learning and Data Science. *Int. J. Inform. Commun. Technol.* **2020**, *9*, 117. [\[CrossRef\]](#)
- Hasan, M.; Balamurugan, E.; Almamun, M.; Sangeetha, K. An Intelligent Machine Learning and Self Adaptive Resource Allocation Framework for Cloud Computing Environment. *EAI Endorsed Trans. Cloud Syst.* **2020**, *6*, 165501. [\[CrossRef\]](#)
- Li, C.; Tang, X. On Fault-Tolerant Bin Packing for Online Resource Allocation. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *31*, 817–829. [\[CrossRef\]](#)
- Gupta, N.; Gupta, K.; Gupta, D.; Juneja, S.; Turabieh, H.; Dhiman, G.; Kautish, S.; Viriyasitavat, W. Enhanced Virtualization-Based Dynamic Bin-Packing Optimized Energy Management Solution for Heterogeneous Clouds. *Math. Probl. Eng.* **2022**, *2022*, 1–11. [\[CrossRef\]](#)
- Thakkar, P.; Nathan, S.; Viswanathan, B. Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform. In Proceedings of the 2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), Milwaukee, WI, USA, 25–28 September 2018; pp. 264–276. [\[CrossRef\]](#)
- Liu, M.; Tang, X. Dynamic Bin Packing with Predictions. *Proc. Acm Meas. Anal. Comput. Syst.* **2022**, *6*, 1–24. [\[CrossRef\]](#)
- Kumar, A.; Kumar, R.; Sharma, A. Energy Aware Resource Allocation for Clouds Using Two Level Ant Colony Optimization. *Comput. Inform.* **2018**, *37*, 76–108. [\[CrossRef\]](#)
- Li, L.; Lin, Q.; Wang, J.; Chen, J.; Ming, Z. A Novel Multiobjective Particle Swarm Optimization Algorithm with Dynamic Resource Allocation. In Proceedings of the 2019 IEEE Congress on Evolutionary Computation, Wellington, New Zealand, 10–13 June 2019; pp. 904–911. [\[CrossRef\]](#)

15. Shiekh, S.; Shahid, M.; Sambare, M.; Haidri, R.A.; Yadav, D.K. A Load-Balanced Hybrid Heuristic for Allocation of Batch of Tasks in Cloud Computing Environment. *Int. J. Pervasive Comput. Commun.* **2022**, *ahead-of-print*. [[CrossRef](#)]
16. Infantia Henry, N.; Anbuananth, C.; Kalarani, S. Hybrid Meta-heuristic Algorithm for Optimal Virtual Machine Placement and Migration in Cloud Computing. *Concurr. Comput. Pract. Exp.* **2022**, *34*, e7353. [[CrossRef](#)]
17. Ghobaei-Arani, M.; Shahidinejad, A. A Cost-Efficient IoT Service Placement Approach Using Whale Optimization Algorithm in Fog Computing Environment. *Expert Syst. Appl.* **2022**, *200*, 117012. [[CrossRef](#)]
18. Zhang, J.; Xie, N.; Zhang, X.; Yue, K.; Li, W. Machine Learning Based Resource Allocation of Cloud Computing in Auction. *Comput. Mater. Contin.* **2018**, *56*, 123–135. [[CrossRef](#)]
19. Bal, P.K.; Mohapatra, S.K.; Das, T.K.; Srinivasan, K.; Hu, Y.C. A Joint Resource Allocation, Security with Efficient Task Scheduling in Cloud Computing Using Hybrid Machine Learning Techniques. *Sensors* **2022**, *22*, 1242. [[CrossRef](#)]
20. Talwani, S.; Alhazmi, K.; Singla, J.; Alyamani, H.J.; Bashir, A.K. Allocation and Migration of Virtual Machines Using Machine Learning. *Comput. Mater. Contin.* **2022**, *70*, 3349–3364. [[CrossRef](#)]
21. Yi, D.; Zhou, X.; Wen, Y.; Tan, R. Efficient Compute-Intensive Job Allocation in Data Centers via Deep Reinforcement Learning. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *31*, 1474–1485. [[CrossRef](#)]
22. Yan, J.; Huang, Y.; Gupta, A.; Gupta, A.; Liu, C.; Li, J.; Cheng, L. Energy-Aware Systems for Real-Time Job Scheduling in Cloud Data Centers: A Deep Reinforcement Learning Approach. *Comput. Electr. Eng.* **2022**, *99*, 107688. [[CrossRef](#)]
23. Jayanetti, A.; Halgamuge, S.; Buyya, R. Deep Reinforcement Learning for Energy and Time Optimized Scheduling of Precedence-Constrained Tasks in Edge-Cloud Computing Environments. *Future Gener. Comput. Syst.* **2022**, *137*, 14–30. [[CrossRef](#)]
24. Seid, A.M.; Erbad, A.; Abishu, H.N.; Albaseer, A.; Abdallah, M.; Guizani, M. Blockchain-Empowered Resource Allocation in Multi-UAV-Enabled 5G-RAN: A Multi-agent Deep Reinforcement Learning Approach. *IEEE Trans. Cogn. Commun. Netw.* **2023**, *1*, 3262242. [[CrossRef](#)]
25. Wang, D.; Li, B.; Song, B.; Liu, Y.; Muhammad, K.; Zhou, X. Dual-Driven Resource Management for Sustainable Computing in the Blockchain-Supported Digital Twin IoT. *IEEE Internet Things J.* **2023**, *10*, 6549–6560. [[CrossRef](#)]
26. Gorenflo, C.; Lee, S.; Golab, L.; Keshav, S. FastFabric: Scaling Hyperledger Fabric to 20 000 Transactions per Second. In Proceedings of the 2019 IEEE International Conference on Blockchain and Cryptocurrency, Atlanta, GA, USA, 14–17 July 2019; Volume 30, pp. 455–463. [[CrossRef](#)]
27. Javaid, H.; Hu, C.; Brebner, G. Optimizing Validation Phase of Hyperledger Fabric. In Proceedings of the 2019 International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, Rennes, France, 21–25 October 2019; pp. 269–275. [[CrossRef](#)]
28. Kwon, M.; Yu, H. Performance Improvement of Ordering and Endorsement Phase in Hyperledger Fabric. In Proceedings of the 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS), Granada, Spain, 22–25 October 2019; pp. 428–432. [[CrossRef](#)]
29. Hang, L.; Kim, D.H. Optimal Blockchain Network Construction Methodology Based on Analysis of Configurable Components for Enhancing Hyperledger Fabric Performance. *Blockchain: Res. Appl.* **2021**, *2*, 100009. [[CrossRef](#)]
30. Nakaike, T.; Zhang, Q.; Ueda, Y.; Inagaki, T.; Ohara, M. Hyperledger Fabric Performance Characterization and Optimization Using GoLevelDB Benchmark. In Proceedings of the 2020 IEEE International Conference on Blockchain and Cryptocurrency, Toronto, ON, Canada, 2–6 May 2020; pp. 1–9. [[CrossRef](#)]
31. Zhang, S.; Hua, S.; Pi, B.; Sun, J.; Yamashita, K.; Nomura, Y. Performance Diagnosis and Optimization for Hyperledger Fabric. In Proceedings of the 2020 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS), Paris, France, 28–30 September 2020; pp. 210–211. [[CrossRef](#)]
32. Santiago, C.; Lee, C. Accelerating Message Propagation in Blockchain Networks. In Proceedings of the 2020 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Republic of Korea, 21–23 October 2020; pp. 157–160. [[CrossRef](#)]
33. Aoki, Y.; Shudo, K. Proximity Neighbor Selection in Blockchain Networks. In Proceedings of the 2019 IEEE International Conference on Blockchain (Blockchain), Atlanta, GA, USA, 14–17 July 2019; pp. 52–58. [[CrossRef](#)]
34. Li, M.; Qin, Y.; Liu, B.; Chu, X. Enhancing the Efficiency and Scalability of Blockchain through Probabilistic Verification and Clustering. *Inf. Process. Manag.* **2021**, *58*, 102650. [[CrossRef](#)]
35. Locher, T. Fast Byzantine Agreement for Permissioned Distributed Ledgers. In Proceedings of the Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event USA, 15–17 July 2020; pp. 371–382. [[CrossRef](#)]
36. Geng, T.; Du, Y. Applying the Blockchain-Based Deep Reinforcement Consensus Algorithm to the Intelligent Manufacturing Model under Internet of Things. *J. Supercomput.* **2022**, *78*, 15882–15904. [[CrossRef](#)]
37. Cao, N.; Jiang, D.; Liu, Y.; Zhou, Y.; Du, H.; Qiao, X.; Xia, Y.; Zhu, D.; Yu, F.; Bi, W. Revisit Raft Consistency Protocol on Private Blockchain System in High Network Latency. In *Advances in Artificial Intelligence and Security; Communications in Computer and Information Science*; Sun, X., Zhang, X., Xia, Z., Bertino, E., Eds.; Springer: Cham, Switzerland, 2021; pp. 571–579. [[CrossRef](#)]
38. Jalalzai, M.M.; Busch, C.; Richard, G.G. Consistent BFT Performance for Blockchains. In Proceedings of the 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks—Supplemental Volume (DSN-S), Portland, OR, USA, 24–27 June 2019; pp. 17–18. [[CrossRef](#)]
39. Zhang, L.; Wang, T.; Liew, S.C. Speeding up Block Propagation in Bitcoin Network: Uncoded and Coded Designs. *Comput. Netw.* **2022**, *206*, 108791. [[CrossRef](#)]

40. Zheng, W.; Zheng, Z.; Chen, X.; Dai, K.; Li, P.; Chen, R. NutBaaS: A Blockchain-as-a-Service Platform. *IEEE Access Pract. Innov. Open Solut.* **2019**, *7*, 134422–134433. [[CrossRef](#)]
41. Ma, Z.; Zhao, W.; Luo, S.; Wang, L. TrustedBaaS: Blockchain-Enabled Distributed and Higher-Level Trusted Platform. *Comput. Netw.* **2020**, *183*, 107600. [[CrossRef](#)]
42. Weerasinghe, N.; Hewa, T.; Liyanage, M.; Kanhere, S.S.; Ylianttila, M. A Novel Blockchain-as-a-Service (BaaS) Platform for Local 5G Operators. *IEEE Open J. Commun. Soc.* **2021**, *2*, 575–601. [[CrossRef](#)]
43. Onik, M.M.H.; Miraz, M.H. Performance Analytical Comparison of Blockchain-as-a-Service (BaaS) Platforms. In *Emerging Technologies in Computing*; Miraz, M.H., Excell, P.S., Ware, A., Soomro, S., Ali, M., Eds.; Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering; Springer: Cham, Switzerland, 2019; Volume 285, pp. 3–18. [[CrossRef](#)]
44. Jiang, S.; Cao, J.; Zhu, J.; Cao, Y. PolyChain: A Generic Blockchain as a Service Platform. In *Blockchain and Trustworthy Systems; Communications in Computer and Information Science*; Dai, H.N., Liu, X., Luo, D.X., Xiao, J., Chen, X., Eds.; Springer: Singapore, 2021; pp. 459–472. [[CrossRef](#)]
45. Li, D.; Deng, L.; Cai, Z.; Souri, A. Blockchain as a Service Models in the Internet of Things Management: Systematic Review. *Trans. Emerg. Telecommun. Technol.* **2022**, *33*, e4139. [[CrossRef](#)]
46. Rajendra, Y.; Subramanian, V.; Shukla, S.K. BlockPaaS: Blockchain Platform as a Service. In Proceedings of the 2023 15th International Conference on COMMunication Systems & NETWORKS (COMSNETS), Bangalore, India, 3–8 January 2023; pp. 204–206. [[CrossRef](#)]
47. Zheng, Q.; Wang, L.; He, J.; Li, T. KNN-Based Consensus Algorithm for Better Service Level Agreement in Blockchain as a Service (BaaS) Systems. *Electronics* **2023**, *12*, 1429. [[CrossRef](#)]
48. Cai, Z.; Yang, G.; Xu, S.; Zang, C.; Chen, J.; Hang, P.; Yang, B. RBaaS: A Robust Blockchain as a Service Paradigm in Cloud-Edge Collaborative Environment. *IEEE Access* **2022**, *10*, 35437–35444. [[CrossRef](#)]
49. Zhang, Q.; Liu, L.; Pu, C.; Dou, Q.; Wu, L.; Zhou, W. A Comparative Study of Containers and Virtual Machines in Big Data Environment. In Proceedings of the IEEE International Conference on Cloud Computing, San Francisco, CA, USA, 2–7 July 2018; pp. 178–185. [[CrossRef](#)]
50. Hafiz, A.M. A Survey of Deep Q-Networks Used for Reinforcement Learning: State of the Art. In *Intelligent Communication Technologies and Virtual Mobile Networks; Lecture Notes on Data Engineering and Communications Technologies*; Rajakumar, G., Du, K.L., Vuppapapati, C., Beligiannis, G.N., Eds.; Springer: Singapore, 2023; pp. 393–402. [[CrossRef](#)]
51. Bui, Y.H.; Hussain, A.; Kim, H.M. Double Deep Q-Learning-Based Distributed Operation of Battery Energy Storage System Considering Uncertainties. *IEEE Trans. Smart Grid* **2020**, *11*, 457–469. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.