Hindawi
Wireless Communications and Mobile Computing
Volume 2023, Article ID 2852085, 16 pages
https://doi.org/10.1155/2023/2852085

Research Article

A Caching-Enabled Permissioned Blockchain Scheme for Industrial Internet of Things Based on Deep Reinforcement Learning

Peng Liu[1,2] **Chuanjian Yao**[1] **Chunpei Li**[1] **Shuyi Zhang**[1] **and Xianxian Li**[1,2]

[1]*School of Computer Science and Engineering & School of Software, Guangxi Normal University, Guilin 541004, China*
[2]*Guangxi Key Lab of Multi-Source Information Mining & Security, Guangxi Normal University, Guilin 541004, China*

Correspondence should be addressed to Xianxian Li; lixx@gxnu.edu.cn

Received 11 September 2022; Revised 6 November 2022; Accepted 17 April 2023; Published 4 May 2023

Academic Editor: Celimuge Wu

The integration of the industrial internet of things (IIoT) and blockchain has become a popular concept that provides IIoT with a trustworthy computing environment. Numerous IIoT nodes together form a decentralized network with rich location-aware computation resources, which can offer great data processing capabilities and low-latency services. However, we still face the challenges of how to efficiently process the massive IIoT data on resource-constrained IIoT nodes by blockchain smart contracts, as their storage capacity only allows them to store limited blockchain data. This work is aimed at improving the smart contract execution efficiency on these IIoT nodes by caching based on deep reinforcement learning. On the one hand, focusing on the characteristics of IIoT, the ledger structure, network architecture, and transaction flow are optimized. IIoT nodes are enabled to store and cache part of block data without affecting global data consistency. On the other hand, we formulated the blockchain caching problem as a Markov decision process and implemented a lightweight caching agent based on deep Q-learning. Proper features and a reward function are defined to minimize the execution delay of smart contracts. The extensive experimental results show that our proposed scheme can effectively reduce the data dissemination costs and smart contract execution delays of IIoT nodes that hold limited blockchain data.

1. Introduction

IoT is a things-connected network, where devices can exchange data and process data according to predefined schemes [1]. There is a growing interest in using IoT technologies in various industries, including manufacturing, transportation, and energy [2]. As a subset of border IoT, industrial IoT (IIoT) focuses on the utilization of IoT facilities in the aforementioned industries. Numerous IoT devices, ranging from small sensors to complex controllers, are associated to optimize industrial production procedures, enhance customer experience, reduce costs, or improve efficiency [3].

However, the intrinsic features of IIoT have resulted in some problems, such as poor interoperability, privacy, and security vulnerabilities [4]. The emerging blockchain technology is considered a suitable complement for IIoT to overcome the above problems. Blockchain is a tamper-resistant digital ledger implemented in a distributed fashion [5]. With blockchain, a trustworthy environment can be built to improve the interoperability, privacy, and security of the IIoT. Currently, blockchain has been adopted in some IIoT applications, such as healthcare, smart factory, and energy trading [6–8].

Numerous IIoT nodes together form a decentralized network with rich location-aware computation resources, which can provide great data processing capability and support low-latency services. However, when we use smart contracts to process massive industrial data on IIoT nodes, the efficiency still faces serious challenges. A smart contract is an immutable autonomous program that is deployed on blockchain and used for general-purpose computations. With smart contracts, nontrusting members can interact with each other without a trusted intermediary [9]. Taking industrial production procedures as an example, massive raw data are being continuously collected from numerous sensors. Some of these data may

need to be processed in smart contracts on IIoT nodes and finally revealed on blockchains. However, IIoT nodes are usually characterized by constrained resources, which means that they probably cannot provide blockchain data with sufficient storage capacity. The size of blockchains can be extremely large since the recorded IIoT data grow rapidly. In practice, blockchain data are stored in cloud storage services or distributed storage systems [10, 11]. In this situation, IIoT blockchain nodes have to frequently request data from elsewhere while executing smart contracts. The expensive communication costs limit the ability of these IIoT nodes to offer those delay-sensitive applications satisfactory smart contract execution efficiency.

To the best of our knowledge, there are still no published works on improving smart contract execution efficiency for storage-constrained IIoT nodes. In this paper, for the first time, we propose a caching-enabled permissioned blockchain scheme for IIoT based on deep reinforcement learning (DRL), where storage-constrained IIoT nodes are enabled to store and cache an appropriate part of blockchain data in their local storage to improve the execution efficiency of smart contracts. The ledger structure, network architecture, and transaction flow are optimized to better support our cache-enabled scheme. The consensus processes become more efficient because of the reduction of data dissemination costs in blockchain networks.

Moreover, we propose a DRL caching agent to help the IIoT nodes determine which part of blockchain data should be cached in their local storage to minimize the execution delay of smart contracts. To achieve this aim, we formulate the caching problem as a Markov decision process (MDP). Complex features related to networks, blockchain data, and smart contracts are taken into consideration during the decision-making process. Meanwhile, a proper reward function is defined to guide the agent to take expected actions and improve its convergence performance. As a result, we implement the caching agent based on deep Q-learning [12–14].

In the experiments, we evaluate the performance of data dissemination and smart contract execution of our proposed scheme. The extensive simulation results show that our proposed scheme can effectively reduce the data dissemination costs in blockchain networks and smart contract execution delays on resource-constrained IIoT nodes.

The main contributions of this paper are summarized as follows:

(1) We propose a caching-enabled permissioned blockchain scheme based on DRL, which is aimed at improving the efficiency of smart contract execution on storage-constrained IIoT nodes

(2) We optimize the ledger structure, network architecture, and transaction flow to enable IIoT nodes to store and cache appropriate parts of blockchain data, offering low data dissemination costs

(3) We present a DRL caching agent for our scheme. Proper features and a reward function are defined in the MDP to minimize the execution delay of smart contracts. In addition, an implementation based on deep Q-learning is presented

The remainder of this paper is organized as follows. In Section 2, we overview the literature related to blockchain storage optimization approaches and DRL-based caching schemes. In Section 3, we define the ledger structure, network architecture, and transaction flow of our proposed scheme. In Section 4, we formulate the MDP of our proposed caching agent and present an implementation based on deep Q-learning. In Section 5, we evaluate several metrics to prove the effectiveness of our scheme. Finally, we draw conclusions in Section 6.

2. Related Work

Currently, there have been many works that make effort to reduce the storage requirement of blockchains. Liu et al. [15] proposed a scheme named LightChain, where unrelated blocks are offloaded through unspent transaction output (UTXO) analysis to reduce storage, but only those UTXO-based blockchains can adopt their scheme. In [11, 16], blockchain data are designed to be addressable in a decentralized network and retrievable with low communication complexity. In [17, 18], blockchain data are redundantly encoded into segments and stored on different industrial nodes. In [19–21], sharding technology is applied, where nodes in a blockchain network are divided into groups, and each group maintains an independent blockchain ledger. With the above schemes, less storage is required on blockchain nodes since data are scattered. Unfortunately, the communication costs are still expensive if a blockchain node has to frequently request block data from elsewhere while executing smart contracts.

When it comes to specific IIoT context, Toyoda et al. [22] limit the data that can be stored on blockchains and losslessly compress the sensor data on blockchains periodically to relieve storage pressure. Xu et al. [23] store the massive industry supply chain data in an off-chain database and the hashes of them on chain. Jeong et al. [24] applied a scheme that combines on-chain and off-chain storage to their vehicle data marketplace platform to securely and effectively handle black box videos. Gao et al. [25] proposed a multichannel blockchain scheme for the internet of vehicles, where the blockchain ledger is divided into multiple channels with different block sizes according to vehicle density to save storage on infrastructure nodes. However, these solutions are strongly related to their context.

At present, caching technology based on DRL has been proven effective in improving industrial applications in some aspects. For example, He et al. [26] proposed a caching agent based on dueling DQN to improve the quality of experience (QoE) of edge-enabled IoT. Wang et al. [27] proposed a novel framework DeepChunk, which leverages deep Q-learning for chunk-based caching in wireless data processing networks. Shi et al. [28] proposed a novel DRL-based vehicular caching scheme that responds to driving safety-related content requests from vehicle users. Li et al. [29] proposed a scheme based on deep Q-learning to make decisions on which user is selected to deliver content in a device-to-device network, aiming at improving the QoE. These works

show that caching could be a promising solution to help those storage-constrained IIoT nodes improve their efficiency. Chen et al. [30] proposed an approach that uses distributed proximal policy optimization (DPPO) to cache the frequently used blocks on users. Their approach can work well in blockchain schemes like [31], a privacy-preserving distributed identity for IoT where light nodes frequently verify transactions. However, it mainly aims to help users faster verify transactions without considering more complex circumstances like general-purpose computation in smart contracts.

3. System Model

Our system model is mainly abstracted from Hyperledger Fabric [32], which is a high-performance permissioned blockchain platform that supports smart contracts implemented in general-purpose programming languages. Hyperledger Fabric has a highly modular and configurable architecture, enabling innovation, versatility, and optimization for a broad range of industry use cases. Up to now, Hyperledger Fabric has been adopted by many industrial and IoT applications [33–36]. In this section, we will introduce the ledger structure, network architecture, and transaction flow of our system model.

3.1. Ledger Structure. Specific changes are made in our proposed ledger structure to help it better support caching. As illustrated in Figure 1, a ledger comprises a state database and a blockchain. The state database is a collection of key-value pairs, which holds the latest values of keys. The blockchain is a transaction log that records all changes that have resulted in the current state database. In short, the blockchain determines the state database, and the state database is derived from the blockchain. The state database frequently changes as keys are created, updated, and deleted, but the blockchain is append-only. Through the state database, smart contracts can access blockchain data quickly rather than traversing the entire blockchain.

Once a smart contract call is successful, a new transaction that includes the contract function name, input data, and execution result will be revealed on the blockchain. The execution result specifies what keys in the state database are read and written along with their versions and new values (if existing). Besides, a signature must be attached to each transaction to ensure its proposer has proper permission to participate in the blockchain network.

The same as the most typical chain structure, the blockchain in our proposed ledger consists of a sequence of blocks, where each block links to the previous one through a cryptographic hash except the first. A block comprises a header and a body. A header requires very little storage because it contains only the key properties of the block, such as block hash, previous block hash, Merkle root hash, and timestamp. A body contains a set of concrete transactions. The following changes are applied to our ledger structure.

Firstly, a new field *VariationSet*, which is also a collection of key-value pairs, is introduced into block headers. Suppose that the values of keys $\{k_1,\cdots,k_n\}$ are, respectively, affected by operations $\{op_1,\cdots,op_n\}$ in the transactions of the block $\mathscr{B}_i$, where an operation is only allowed to be *create*, *update*, or *delete*. The *VariationSet* of $\mathscr{B}_i$ is expressed as

$$\mathscr{V}_i = \left\{ \left(k_j, op_j \right) | 1 \le j \le n \right\}. \tag{1}$$

Secondly, as in the scheme in [11], the body of a block is stored on only a responsible part of nodes according to the hash function of a distributed hash table (DHT). A DHT is a decentralized system that provides a lookup service, where the responsibility for maintaining mappings from keys to values is distributed among the nodes. Any participant can efficiently retrieve the value of a given key. Besides, blockchain nodes can use their idle storage space to cache some other block bodies, on behalf of the execution efficiency of smart contracts.

Thirdly, if the body of $\mathscr{B}_i$ is not in local storage, any key k_j, whose latest value is defined in the transaction that $\mathscr{B}_i$ contains, will be mapped to the block number of $\mathscr{B}_i$ instead of the actual value in the state database. Hence, the size of the state database can be reduced, without undermining the ability to locate values on blockchains.

With the above changes, our proposed ledger structure supports the following operations.

(1) Append a new block header: once a new block is generated, its header will be delivered to all blockchain nodes. As soon as the header $\mathscr{H}_i$ of a new block $\mathscr{B}_i$ is received, a node will first update the state database of its ledger according to $\mathscr{V}_i$ in $\mathscr{H}_i$. There are three conditions: (1) if a key k_j is created in $\mathscr{B}_i$, a mapping from k_j to the block number $\mathscr{I}_i$ of $\mathscr{B}_i$ will be put in the state database. (2) If a key k_j is updated in $\mathscr{B}_i$, the current value that k_j is mapping to in the state database will be replaced with the block number $\mathscr{I}_i$ of $\mathscr{B}_i$. (3) If a key k_j is deleted in $\mathscr{B}_i$, it will be directly removed from the state database together with the value it is mapping to. After updating the state database, $\mathscr{H}_i$ will be appended to the tail of the blockchain of the ledger as a result. The details are shown in Algorithm 1. This operation ensures nodes can keep consistent with the latest-version global ledger without concrete block data. Moreover, the data dissemination costs can be reduced because only lightweight block headers are required to be transmitted

(2) Insert block body: if a blockchain node intends to proactively cache or is assigned to store a complete block replica $\mathscr{B}_i$, it needs to insert the corresponding block body $\mathscr{D}_i$ into its ledger while appending the header $\mathscr{H}_i$ to the ledger. The same as appending block headers, the state database of the ledger should be updated first. If the latest value of a key k_j is defined in $\mathscr{B}_i$, the block number $\mathscr{I}_i$ that k_j currently maps to in the state database will be replaced with the actual latest value. Finally, insert $\mathscr{D}_i$ into the blockchain of the ledger. The details are shown in Algorithm 2

(3) Delete block body: when data in a block $\mathscr{B}_i$ are deprecated because newer versions of data have emerged or are unwanted by the caching policy, the blockchain

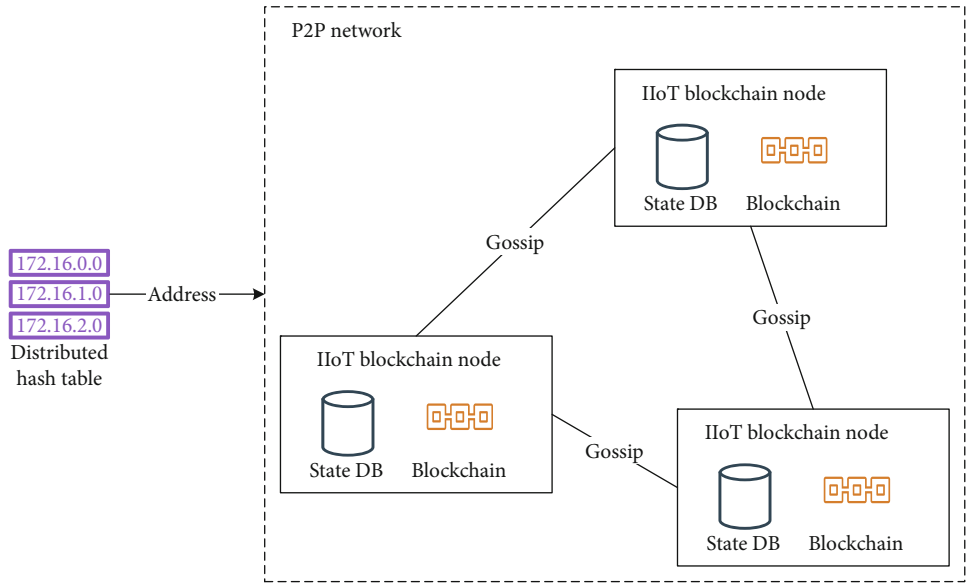

Figure 1: Ledger structure. The ledger consists of a state database and a blockchain. The blockchain includes a sequence of blocks, where each block contains a set of transactions that change the key-value pairs in the state database. The DHT is responsible to decide which part of IIoT nodes should store the body of a block, according to a hash function. Through the DHT, all block bodies and transactions can be quickly addressed.

Require: New Block header $\mathcal{H}_i$, Blockchain $\mathcal{BC}$, State Database *SDB*.
1: $\mathcal{I}_{i-1} \Longleftarrow$ the last block number in $\mathcal{BC}$
2: $\mathcal{I}_i \Longleftarrow$ the block number of $\mathcal{H}_i$
3: **if** $\mathcal{I}_i = \mathcal{I}_{i-1} + 1$ **then**
4: $\mathcal{V}_i \Longleftarrow$ the *VariationSet* of $\mathcal{H}_i$
5: **for each** key-value pair (k, op) **in** $\mathcal{V}_i$ **do**
6: **if** op = *create* **or** op = *update* **then**
7: $SDB[k] \Longleftarrow \mathcal{I}_i$
8: **else if** op = *delete* **then**
9: Delete k from *SDB*
10: Append $\mathcal{H}_i$ to $\mathcal{BC}$

Algorithm 1:Append new block header.

Require: Target block number $\mathcal{I}_i$, Blockchain $\mathcal{BC}$, State Database *SDB*
1: $\mathcal{V}_i \Longleftarrow$ the *VariationSet* of the header $\mathcal{H}_i$
2: **for each** transaction tx **in** the body $\mathcal{D}_i$ **do**
3: **for each** affected key-value pair (k, v) **in** tx **do**
4: **if** $SDB[k] = \mathcal{I}_i$ **and** $\mathcal{V}_i[k] \neq$ *delete* **then**
5: $SDB[k] \Longleftarrow v$
6: Insert $\mathcal{D}_i$ into $\mathcal{BC}$

Algorithm 2: Insert block body.

node will use this operation to delete $\mathcal{D}_i$ from the ledger to release the occupied storage space. Thereafter, if the latest value of a key k_j in the state database is defined in $\mathcal{B}_i$, the value that k_j is mapping to in the state database will be replaced with the number $\mathcal{I}_i$ in $\mathcal{B}_i$. The details are shown in Algorithm 3

3.2. Network Architecture. Since IIoT blockchains are usually maintained by organizations rather than individuals, we assume that each entity in blockchain networks belongs to a specific organization. As shown in Figure 2, we design a general blockchain network structure for IIoT, where each node can be assigned the following roles.

Require: Target block number $\mathcal{I}_i$, Blockchain $\mathcal{BC}$, State Database SDB
1: $\mathcal{V}_i \Longleftarrow$ the *VariationSet* of the header $\mathcal{H}_i$
2: **for each** transaction tx **in** the body $\mathcal{D}_i$ **do**
3: **for each** affected key-value pair (k, v) **in** tx **do**
4: $ver_v \Longleftarrow$ the version of v
5: $ver_k \Longleftarrow$ the version of $SDB[k]$
6: **if** $ver_v = ver_k$ **and** $\mathcal{V}_i[k] \neq$ *delete* **then**
7: $SDB[k] \Longleftarrow \mathcal{I}_i$
8: Delete $\mathcal{D}_i$ from $\mathcal{BC}$

Algorithm 3: Delete block body.

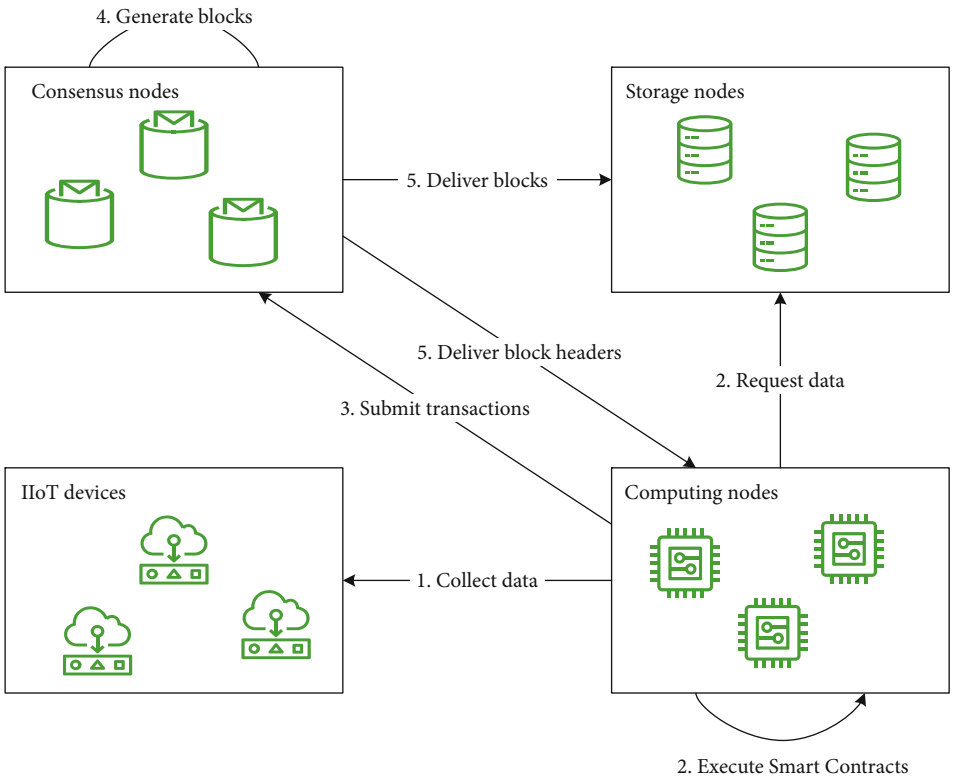

Figure 2: Network architecture. A network includes a number of organizations as its participants. Each organization holds consensus nodes, storage nodes, storage-constrained computing nodes, and many IIoT devices. The computing nodes collect data from IIoT devices and handle them by smart contracts. New transactions will be submitted to the consensus nodes. These transactions will be ordered, assembled into new blocks, and delivered. Finally, the new blocks will be stored completely in the specific storage nodes, and their headers will be delivered to all other nodes.

(1) Consensus node: the order of transactions needs to be agreed by all stakeholder organizations in a blockchain network. Thus, each of them should offer at least one node to participate in transaction ordering. These consensus nodes together form a consensus service, which receives transactions from the transaction proposers, arranges batches of them into a well-defined sequence, assembles them into blocks, and delivers. The service usually uses underlying algorithms such as DPoS, Raft, and PBFT [37, 38] to provide high-performance consensus. The header of a block will be delivered to all blockchain nodes, but the body will only be sent to a specific set of storage nodes according to the DHT

(2) Storage node: to reduce global storage consumption, we adopt the storage scheme in [11], where the blockchain data are uniformly, dispersedly, and redundantly stored in the whole blockchain network. Each storage node is responsible for maintaining a specific set of block bodies according to a hash function, and all of them together form a DHT-based storage service. All block bodies and transactions can be addressed and retrieved with low complexity

in the DHT. Every time a new block is generated, a set of storage nodes, which are responsible to store complete replicas, will always be chosen by the consensus nodes according to the hash function. These storage nodes handle the data requests from other nodes and offer them the block data it has stored. Although an organization does not maintain a complete ledger anymore, it can still be easily figured out whether a transaction or a block body was tampered with simple payment verification (SPV)

(3) Computing node: a computing node is usually an IIoT node with limited storage resources and serves a specific group of IIoT devices nearby. As soon as data are collected from IIoT devices like sensors, they will be instantly handled by computing nodes. Using these data as inputs, computing nodes execute the smart contracts to get execution results and reveal them on blockchains in the form of transactions. During smart contract execution, existing data might be frequently read from the ledger. To minimize the costs of requesting data from the storage nodes, each computing node always utilizes an independent caching agent, which we will discuss in Section 4, to cache an appropriate part of the blockchain data locally. Computing nodes and storage nodes use the same ledger structure, but the formers proactively choose what data should be stored, and the latters are passively chosen by the hash function of the adopted DHT scheme

Note that a physical node can serve different roles at the same time. Physical nodes can select proper roles according to their computing, storage, and network resources, adapting to the heterogeneity of IIoT nodes and improving the utilization of hardware. Taking the most common conditions as examples, if a cloud server has high bandwidth and sufficient storage space, it can play a consensus node and a storage node simultaneously on behalf of the consensus throughput and data addressing performance (shorter search paths are required in DHT). If an edge node, which mainly serves as a computing node to provide low-latency services by smart contracts to the edge devices, has considerable unoccupied storage space, being a storage node can lower the data request costs of the nearby computing nodes.

3.3. Transaction Flow. The transaction flow describes the lifecycle of transactions from being created to being confirmed. As illustrated in Figure 2, the transaction flow can be summarized in the following phases.

(1) Data collecting phase: each computing node listens to its nearby IIoT devices through wireless connections for new data events. Once an event arrives, the computing node will immediately read the new data from the corresponding IIoT device

(2) Smart contract execution phase: functions of smart contracts will be called. While a computing node is executing the smart contract, it might need to frequently read the values of existing keys from the ledger. In general, if the latest value of a key is stored or cached locally, it will be read directly from the local ledger; otherwise. it will be searched from the DHT and requested from other storage nodes. As a result, a new transaction will be created, signed, and sent to one of the consensus nodes

(3) Consensus phase: the aforementioned consensus service establishes a total order on all the received transactions through its consensus algorithm. Once the block interval or block size reaches the limit, the consensus service will package the ordered transactions into a new block. In a legal block, every transaction must be validated by the following steps. (1) Each consensus node will perform the permission check and read-write conflict check. The former verifies the membership of transaction initiators through their signatures. The latter ensures that no other transactions have changed the data that a transaction depends on. (2) Each stakeholder organization needs to check whether transactions are honestly executed by others. The transactions, whose initiators belong to other organizations, will be endorsed by the trustworthy computing nodes by checking the correctness of transactions through duplicate executions. A transaction is valid if it is confirmed by most consensus nodes

(4) Data delivery phase: each consensus node delivers the new block replica and the header, respectively, to the specific storage nodes and all other nodes. Note that a consensus node only delivers data to storage nodes or computing nodes that belong to the same organization. If a storage node or computing node received a new block or block header from others, it will update its ledger (with Algorithm 1 and Algorithm 2) and disseminate the header to its neighbors through Gossip. This phase continues until data consistency is achieved in the whole blockchain network

4. Caching Agent

In this section, the caching problem will be formulated as an MDP, aiming at guiding the agent to learn from context features and minimize smart contract execution delay. Thereafter, we will show how to implement the caching agent using deep Q-learning.

4.1. Problem Formulation. According to our defined system model, a caching agent is responsible for making decisions on which part of block bodies should be cached on a computing node in order to improve the smart contract execution efficiency. Our proposed caching agent works with cache replacements. Each time the caching agent is called, an action is taken that replaces a block body in the ledger with another one.

Before formulating the MDP, a series of symbols need to be defined. $\{k_1, k_2, k_3, \cdots\}$ denotes all keys that have been created, despite their current existence. Let T be the most recent

period of time, $\mathcal{NR}_j$ indicates how many times k_j has been requested in T; $\mathcal{NU}_j$ indicates how many times that k_j has been updated in T; $\mathcal{DL}_j$ indicates the total delay of requesting k_j from other storage nodes in T. $\{\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3, \cdots\}$ and $\{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \cdots\}$, respectively, denote all confirmed blocks and their bodies. K_i indicates the set of keys whose latest values are defined in $\mathcal{B}_i$. $\mathcal{DP}_i$ indicates the current percentage of deprecated transactions in $\mathcal{B}_i$, where a transaction is deprecated once the values that it created or updated are deleted or overwritten in newer blocks. $\{\mathcal{SC}_1, \mathcal{SC}_2, \mathcal{SC}_3, \cdots\}$ represents all smart contracts deployed on the blockchain. $\mathcal{CT}_{i,j}$ indicates the number of transactions that are related to $\mathcal{SC}_j$ in $\mathcal{B}_i$. $\mathcal{CF}_j$ indicates the frequency of $\mathcal{SC}_j$ being called in T.

To find the optimal caching agent π, we formulate the caching problem as an MDP, which is denoted as a tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{M}(s_{n+1}|s_n, a_n), \mathcal{R}(s_n, a_n), \gamma\}$.

$\mathcal{S}$ is the collection of states. We denote the state of an arbitrary block $\mathcal{B}_i$ as

$$\mathcal{BF}_i = \left(\sum_{k_j \in K_i} \mathcal{NR}_j, \sum_{k_j \in K_i} \mathcal{NU}_j, \sum_{k_j \in K_i} \mathcal{DL}_j, \mathcal{DP}_i, \sum_{\mathcal{SC}_j} \mathcal{CT}_{i,j} \times \mathcal{CF}_j \right). \tag{2}$$

We assume that a computing node can cache at most $\mathcal{C}$ block bodies. At timestep n, if a computing node needs to decide whether to store the candidate block body $\mathcal{D}_{\mathcal{C}+1}$ as there are already $\mathcal{C}$ block bodies $\{\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_{\mathcal{C}}\}$ cached in its ledger, the current state s_n is denoted as

$$s_n = \{\mathcal{BF}_{\mathcal{C}+1} - \mathcal{BF}_1, \cdots, \mathcal{BF}_{\mathcal{C}+1} - \mathcal{BF}_{\mathcal{C}}\}, \tag{3}$$

where $\mathcal{D}_{\mathcal{C}+1}$ is explicitly compared with $\{\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_{\mathcal{C}}\}$ to help the caching agent directly learn from the difference between states, rather than letting it learn how to compare itself.

$\mathcal{A}$ is the collection of actions. The action at timestep n is denoted as a_n. There are $\mathcal{C}+1$ possible actions for a state, following the action space definition in [39, 40]. Let

$$\mathcal{A} = \{0, 1, 2, \cdots, \mathcal{C}\}, \tag{4}$$

where $a_n = 0$ means that the ledger will keep the same and $a_n \in \{1, 2, \cdots, \mathcal{C}\}$ means that the candidate $\mathcal{D}_{\mathcal{C}+1}$ will be requested from other storage nodes and used to replace $\mathcal{D}_{a_n}$.

$\mathcal{M}$ is the state transition function that maps a state-action pair (s_n, a_n) at timestep n to the probability distribution of states at timestep $n+1$.

$\mathcal{R}$ is the immediate reward function that determines the reward for performing the action a_n at state s_n. A proper reward function is important to the convergence performance of DRL agents. However, the sparse reward problem is notable in caching environments because the impact of a single action on the environment is too slight to be observed. The metrics, such as QoE and cache hit ratio, are often taken into consideration by the reward functions for DRL-based caching schemes. [26, 39–42]. These reward functions offer poor convergence performance. Reward shaping, an effective technique for incorporating domain knowledge into RL [43, 44], could be a feasible way to solve the sparse reward problem in caching. For example, Wu et al. [45] defined handcrafted extrinsic rewards, which are related to living, health loss, ammo loss, etc., to teach their agent to take expected actions in the first-person shooter game. Likewise, we define an extrinsic reward in our reward function to teach our agent to store those items with expected features. When action a_n replaces $\mathcal{D}_u$ with $\mathcal{D}_v$, then the extrinsic reward is expressed as

$$\mathcal{R}_{ets}(s_n, a_n) = \frac{\mathcal{DL}_i \sum_{k_u \in K_i} (\mathcal{NR}_u - \mathcal{NU}_u)}{\mathcal{DL}_j \sum_{k_v \in K_j} (\mathcal{NR}_v - \mathcal{NU}_v)}. \tag{5}$$

The extrinsic reward is positive only if caching $\mathcal{D}_u$ brings more execution delay reduction than $\mathcal{D}_v$ in T. The agent will receive higher extrinsic rewards as it tries to cache those block bodies whose data are requested more frequently but less updated. Otherwise, a negative extrinsic reward will be returned as a punishment to avoid the agent to take those bad actions. However, note that the extrinsic reward is auxiliary; our agent aims to minimize the smart contract execution delay. Thus, we define intrinsic reward $\mathcal{R}_{its}(s_n, a_n)$ for action a_n, which is the average execution delay reduction observed in a fixed number of timesteps after performing a_n. As a result, the complete reward function is a weighted sum of the extrinsic reward and the intrinsic reward

$$\mathcal{R}(s_n, a_n) = r_n = \beta \bullet \mathcal{R}_{its}(s_n, a_n) + (1 - \beta) \bullet \mathcal{R}_{ets}(s_n, a_n) \tag{6}$$

where β is a dynamic weight factor. Therefore, the smart contract execution delay can be reduced as the reward increases. After the reward became stable, the value of β should be gradually increased to deprecate the extrinsic part of the reward function.

$\gamma \in (0, 1]$ is the discount factor that determines the effect of future rewards on the current decision-making process. Given the accumulated reward definition $R_n = \sum_{m=0}^{\infty} r_{n+m} \gamma^m$ at timestep n, the lower the value of γ, the more significant the impact of immediate rewards.

Let $\pi = \pi(s_n, a_n)$ be a mapping from the state s_n to the probability of action a_n. As shown in Figure 3, the following steps are repeated:

(1) Before timestep n, observe the features of the candidate block body and cached block bodies to obtain the state s_n. The candidate is supposed to be a block body whose data is currently requested from other storage nodes

(2) Utilize π to predict the probabilities of all actions and perform the action a_n that maps to the maximum probability

(3) The reward $r_n = \mathcal{R}(s_n, a_n)$ is received, and the state transits from s_n to s_{n+1}, respectively, according to $\mathcal{R}(s_n, a_n)$ and $\mathcal{M}(s_{n+1}|s_n, a_n)$

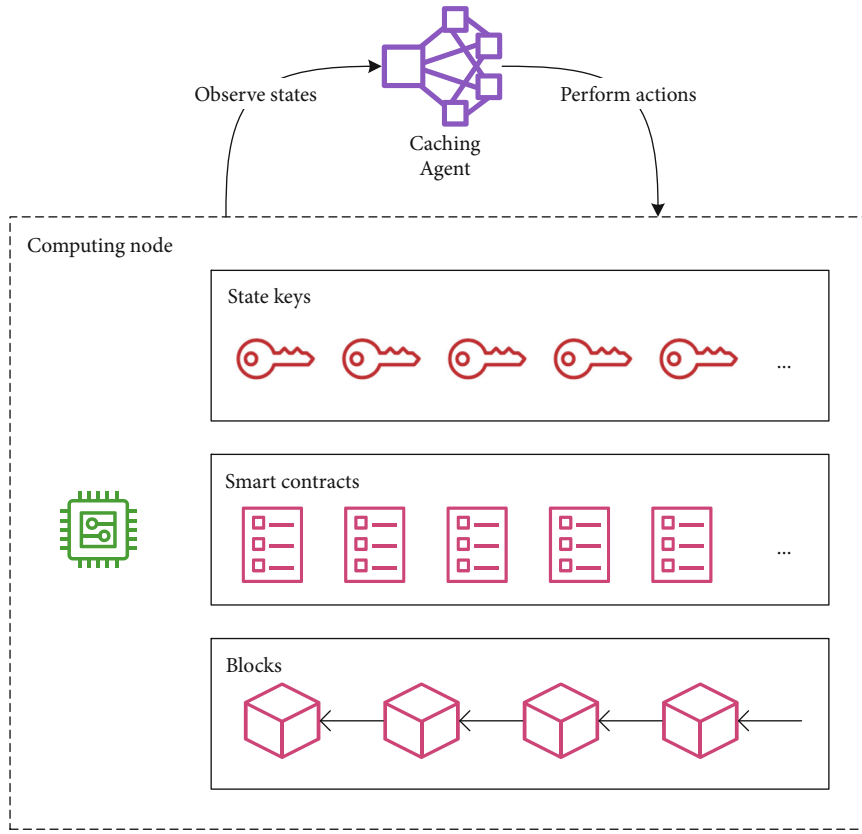

Figure 3: Caching agent overview. The caching agent can observe states from the environment, where a state contains statistical features of keys, smart contracts, and blocks. To minimize the average execution delay of smart contracts, each action performed by the agent might replace a cached block body with another one to maximize the reward.

We aim to find the optimal caching agent π^* that can achieve the maximum expected R_n at different states. It is denoted as

$$\pi^* = \underset{\pi}{\operatorname{argmax}} E[R_n|\pi]. \tag{7}$$

The value function is defined to measure how good π is. Meanwhile, it represents the expected R_n for deterministically selecting a_n as the initial action at the state s_n following π. It's denoted as

$$Q^{\pi}(s,a) = E_{\pi}[R_n|s_n = s, a_n = a]. \tag{8}$$

The optimal value function for given state s and action a is defined as $Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$. The optimal agent can be obtained by greedily selecting the action a_n, which maximizes $Q^*(s,a)$, from all available actions at the state s_n. Q^{π} can be decomposed into a Bellman Equation according to Markov Property, which is expressed as

$$Q^{\pi}(s,a) = \sum_{s',r} p\left(s', r|s, a\right)\left[r + \gamma \sum_{a'} \pi\left(a'|s'\right) Q^{\pi}\left(s'|a'\right)\right], \tag{9}$$

where $p(s', r|s, a)$ is the probability that reward r is received and transits to the state s' after performing action a at state s. The distribution of $p(s', r|s, a)$is affected by $\mathscr{M}$ and $\mathscr{R}$. Thus, if $\mathscr{M}$ and $\mathscr{R}$ are available, we can leverage policy evaluation to continually improve $Q^{\pi}(s,a)$ by greedily selecting until an optimal agent is found.

4.2. Implementation. Deep Q-learning is a practical DRL approach for tasks with continuous state spaces and discrete action spaces. Watkins et al. [46] proposed Q-learning, which provides agents the learning capability to act optimally in Markovian domains by updating action selection using the Bellman optimal equation and the Epsilon-Greedy scheme. Mnih et al. [12] proposed the Deep Q-Network (DQN), where the Q-function is approximated by a deep neural network, to enable Q-learning to handle tasks with continuous state spaces. The agents are enabled to directly learn from raw, high-dimensional features without any prior knowledge about environments. Q-learning updates $Q^{\pi}(s,a)$ according to the following equation

$$Q^{\pi}(s_n, a_n) = (1 - \eta_n) Q^{\pi}(s_n, a_n) + \eta_n \left[R_n + \gamma \max_{a \in \mathscr{A}} Q^{\pi}(s_{n+1}, a)\right], \tag{10}$$

where η_n is the learning rate at timestep n. By greedily selecting the action a_n that maximizes $Q^{\pi}(s_n, a_n)$ from all available actions at the state s_n, the Q values and the rewards can be continuously improved. As a result, we will obtain π^* that maximizes R_n. Hasselt et al. [13] proposed double Q-learning, where a target network is introduced to decompose the value selection and evaluation and alleviate the overestimation problem of Q-learning. The parameters of the main network are copied to the target network after every fixed number of timesteps in order to stabilize the estimation of target Q-values. Besides, Wang et al. [14] proposed dueling DQN that decouples the value function and advantage function and led to dramatic improvements over the existing Q-learning methods. Our caching agent is implemented based on a double-dueling DQN. It comprises a main network and a target network, which are two dueling DQNs with completely the same architecture.

Once limit $\mathscr{C}$ is reached, the agent will keep updating the ledger. Each time a key, whose latest value is nonexistent in the state database, is requested, the corresponding block body will be treated as a candidate. Upon using the previously defined state s_n as the input of π, action a_n will be selected according to the output Q values. Otherwise, if an Epsilon Greedy scheme is followed, a_n will be selected randomly from $\mathscr{A}$ with probability $1 - \epsilon$ despite the Q values, which helps the agent avoid overfitting and behave more aggressively while exploring the environment. Every state transition tuple (s_n, a_n, r_n, s_{n+1}), where r_n is the reward for performing a_n and s_{n+1} is the next state, will be saved in an experience buffer. Experience replay is performed every fixed number of timesteps and samples a minibatch of transitions from the experience buffer as the input to train π by minimizing the loss function

$$\mathscr{L}(\theta) = \frac{1}{\mathscr{M}_{\mathscr{B}}} \sum_{i \in \mathscr{M}_{\mathscr{B}}} \left(y_i - Q^{\theta}(s_i, a_i) \right)^2, \tag{11}$$

where θ is the parameters of the main network; $\mathscr{M}_{\mathscr{B}}$ is the size of a mini-batch; and $y_i = r_i + \max_{a_i'} Q^{\theta^-}(s_i', a_i')$ is the target Q-value output by the target network, whose parameters are θ^-.

While training a caching agent on a computing node, the growth of the training timestep can only be triggered by two events that affect the current state that the agent is observing. The first one is that a new block was generated, and the second one is that a query request was sent by the maintaining smart contracts. Both of these events can change the block states by read or write actions therefore affecting the current environment state. Each block affected by the above events will be treated as a candidate if its body has not been cached. Subsequently, the agent will use the aforementioned steps to decide whether to replace a cached item with the candidate. If a replacement action is chosen, the computing node will search for the candidate on the DHT, request it from a responsible storage node, and do the replacement. The details of the above process are shown in Algorithm 4.

5. Experiment

5.1. Data Dissemination Performance. The simulation is conducted in a local P2P network, whose topology is predefined using the Watts-Strongatz model. This is a random graph generation model that produces graphs with small-world properties, including short average path lengths and high clustering. Each node is a Raspberry Pi 4 in a local area network and serves as a computing node and a storage node at the same time. Besides, a server in the same local area network, which is equipped with an Intel Xeon E5-2630 v2 CPU (2.60 GHz, 12 cores, 24 logical processors) and 32 GB of RAM, is used as a consensus node to disseminate block data to the P2P network. We assume that all nodes have an equivalent distance to any of their neighbors, and the distance between two arbitrary nodes is the length of the shortest path between them. 10,000 blocks, whose sizes are subject to a scaled standard normal distribution ranging from 128 KB to the block size limit (1 MB, 2 MB, 4 MB, or 8 MB), are disseminated in sequence. For each new block, the consensus node always randomly chooses 30 percent of the storage nodes according to the hash function of DHT and directly delivers them a complete block replica. Afterward, these nodes disseminate the block header to all other computing nodes and storage nodes through Gossip [47], where the nodes always transmit newly arrived block headers to a random part of their neighbors.

We count the total dissemination costs by aggregating the sizes of all end-to-end messages. The transmission cost for the consensus node to send a complete block to a specific storage node is defined as the product of the block size and the shortest distance between them. Since our proposed scheme is implemented based on Hyperledger Fabric, we use it as the baseline scheme to compare. In Hyperledger Fabric, the nodes disseminate complete blocks to each other, instead of block headers that are much more lightweight.

In Figure 4(a), we evaluate the total dissemination costs in networks with different numbers of nodes from 20 to 100. The fan-out of each node is 2, and the maximum block size is 2 MB. The value of fan-out represents how many neighbors at most a node randomly selects to forward the data it received. Thus, once a node receives a new block header, it will select 2 random nodes from all of its neighbors, except the one the header came from, and forward the header to them. As the number of nodes grows, the costs of Hyperledger Fabric increase much faster than our scheme. This is because it is hard for a node to avoid forwarding messages to the neighbors that have held the duplicates as the network topology becomes more complex. It leads to considerable redundant transmission costs as a result. Compared to Hyperledger Fabric, since block headers require fewer costs to be transmitted, our scheme is more efficient, especially in larger-scale networks. The main costs come from the processes where consensus nodes send complete blocks to specific nodes. Thus, our scheme performs better in large-scale IIoT networks.

In Figure 4(b), we study the relationship between block size limit and dissemination costs. A higher block size limit allows the consensus service to pack more transactions into

Require: Cache Agent π, Main Network Update Frequency $\mathscr{F}_\theta$, Target Network Update Frequency $\mathscr{F}_{\theta^-}$, Blockchain $\mathscr{BC}$, State Database *SDB*

Initialize the main network of π with θ
Initialize the target network of π with θ^-
Initialize experience buffer $buf \Longleftarrow \varnothing$
Initialize timestep $n \Longleftarrow 0$
for each new transaction tx or smart contract query request req **do**
if $tx \neq null$ **then**
  $ws \Longleftarrow$ the keys whose values were affected by tx
  if $req \neq null$ **then**
    $rs \Longleftarrow$ the keys whose values were read by req
  for each key k **in** ws or rs **do**
    $\mathscr{B}_i \Longleftarrow$ the block that defines the latest value of k in *SDB*
    Update the state $\mathscr{BF}_i$ of $\mathscr{B}_i$ according to Equation (2)
    Treat the block body $\mathscr{D}_i$ of $\mathscr{B}_i$ as a candidate
    if $\mathscr{D}_i$ is not in cache **then**
      Obtain s_n according to Equation (3)
      if $n \neq 0$ **then**
        $buf \Longleftarrow buf \cup \{(s_{n-1}, a_{n-1}, r_{n-1}, s_n)\}$
      with probability $1 - \epsilon$ **select**
        $a_n \Longleftarrow \text{argmax}_{a \in \mathscr{A}} Q^\pi(s_n, a)$
      otherwise
        Select random a_n from $\mathscr{A}$
      if $a_n \neq 0$ **then**
        Delete $\mathscr{D}_{a_n}$ from $\mathscr{BC}$ and update *SDB* according to Algorithm 3
        Address $\mathscr{B}_i$ on DHT for the responsible storage nodes by hashing $\mathscr{I}_i$
        Request $\mathscr{D}_i$ from a storage node with low latency
        Insert $\mathscr{D}_i$ into $\mathscr{BC}$ and update *SDB* according to Algorithm 2
      Obtain r_n according to Equation (7)
      if $n \mod \mathscr{F}_{\theta^-} = 0$ **then**
        Copy θ to the target network of π
      if $n \mod \mathscr{F}_\theta = 0$ **then**
        Sample a mini-batch from *buf*
        Minimize Equation (11) and update θ
      $n \Longleftarrow n + 1$

Algorithm 4: Train Caching Agent on Computing Node.

the same block. Two conditions trigger the generation of new blocks: one is that the total size of the collected transactions exceeded the block size limit, and the other is that the time has reached the block interval limit. Therefore, setting a proper block size limit is important to improve the throughput of a blockchain system. An inappropriate block size limit can break the balance between system throughput and latency. Thus, we evaluate the costs, respectively, under block sizes limit from 1 MB to 8 MB. The fan-out is 2 likewise and the network size is 100. The results show that setting a larger block size limit is helpful to reduce dissemination costs. Our scheme only costs around 30 percent compared to Hyperledger Fabric when the maximum block size is 1 MB. This ratio becomes around 35% when the maximum block size is larger than 4 MB. Thus, our scheme provides better efficiency while using larger block size limits. When it comes to industrial, a higher block size limit is required considering the rapid generation of data, which is preferred in our scheme.

In Figure 4(c), we explore the impact of fan-out on data dissemination performance. With a proper value of fan-out, the consensus of new blocks can be reached faster in the network. An unfit fan-out might lead to network congestion as too many messages are transmitted simultaneously. Since Hyperledger Fabric mainly uses the push action of Gossip to broadcast new blocks, where a push action implies a node forwards a message it newly received to another one, we count the number of total push actions as a measure of data disseminating performance. However, note that fan-out is unlimited in our proposed scheme. Because the size of a block header is small enough, which means far fewer data are required to be transmitted, disseminating a block header to all neighbors is allowed without concern about the congestion it might bring to the network. The values of fan-out from 1 to 5 are applied while evaluating Hyperledger Fabric. As shown, it needs to set a fan-out value that is not smaller than 5 to achieve a total push number approximate to our scheme. To avoid conflicts between transactions, new blocks are usually generated very frequently in a permissioned blockchain, where the interval is as short as a few seconds. While frequently disseminating blocks in the local network, high fan-out values can cause unexpected

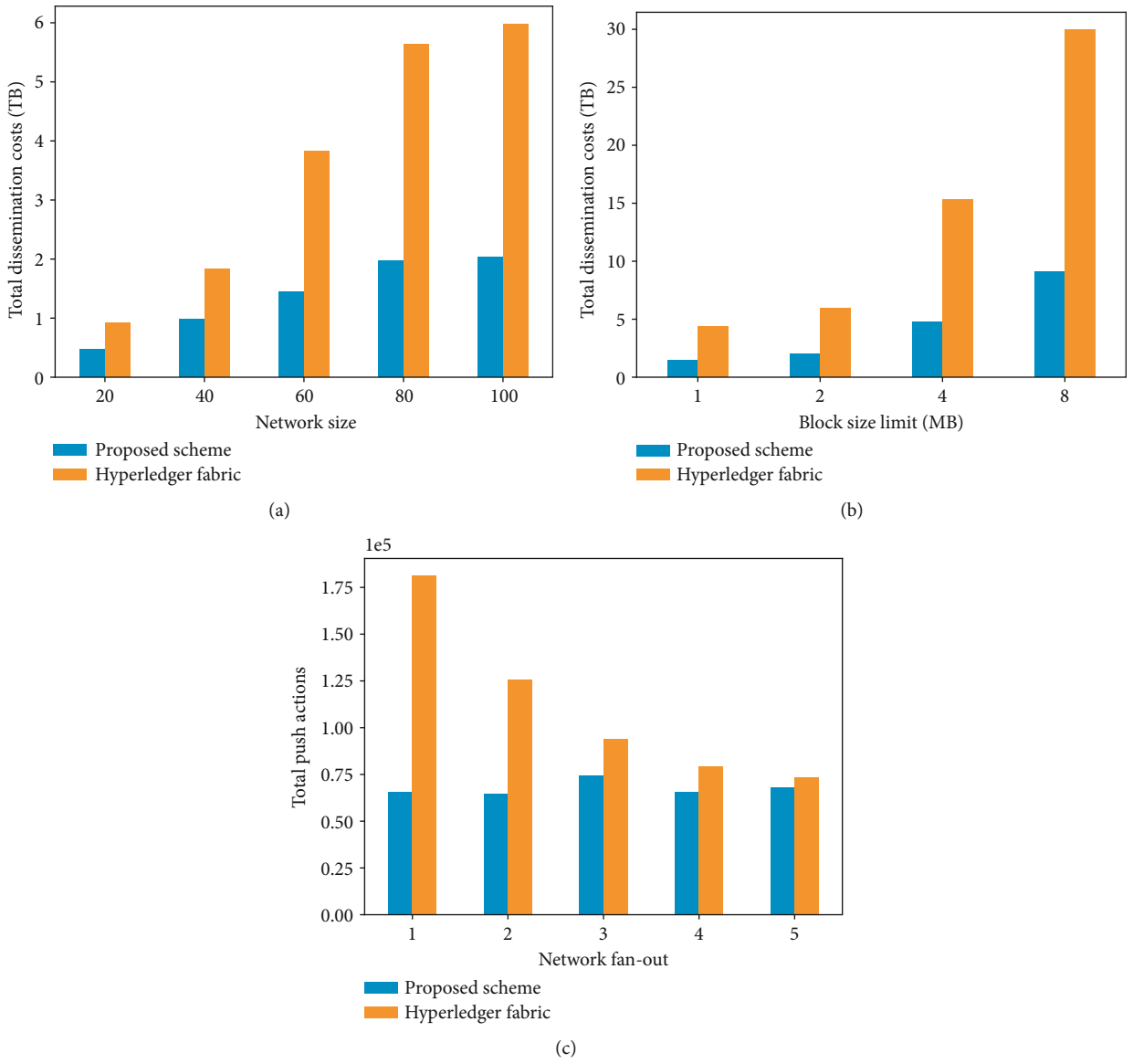

Figure 4: (a) The proposed scheme vs. Hyperledger Fabric. Comparison of blockchain data dissemination costs under different network sizes from 20 to 100. (b) The proposed scheme vs. Hyperledger Fabric. Comparison of blockchain data dissemination costs under different block size limits from 1 MB to 8 MB. (c) The proposed scheme vs. Hyperledger Fabric. Comparison of the total numbers of push actions under different Gossip fan-out values from 1 to 5.

network congestion. Due to fewer communication times, our scheme can improve the performance of reaching a consensus on new blocks in dense networks.

5.2. Caching Performance. We conduct the caching agent (using Python 3.8 and PyTorch 1.10) and deploy it on a Raspberry Pi, which serves as a computing node only. The model of the agent sequentially contains a convolutional layer with a kernel size 1×5, two parallel DNNs that are, respectively, a value function network and an advantage function network, and an output layer with 128 neurons. The value function network contains two linear layers, sequentially with 128 neurons and 64 neurons. The advantage function network contains two linear layers, sequentially with 128 neurons and 1 neuron. The input (state space) size and the output (action space) size of the agent are, respectively, 128×5 and 128, following the aforementioned MDP definition. It means that our agent can consider 128 cached blocks each time it makes a decision. The capacity of the experience buffer is 100,000, which means how many recent interactions with the environment can be remembered while using the experience replay technique to train the agent. The mini-batch size $\mathscr{M}_{\mathscr{B}} = 32$, which means how many interactions will be sampled from the experience buffer and used as a training batch. The discount factor $\gamma = 0.9$, which means how important the agent thinks of the long-term history. The reward function weight factor β starts at 1.0 and gradually declines to 0.0 after keeping the rewards stable, which means that we will first use the hand-crafted

reward to train the agent and slowly transit to the actual reward. The main network is updated every 4 timesteps. The target network is updated every 10,000 timesteps.

A ledger can contain a huge number of block bodies, so the state space and the action space can be very large, depending on the value of $\mathscr{C}$. Training such a caching agent with large input and output in an acceptable time is computationally intensive. Since our scheme is oriented to those IIoT nodes with limited computational resources, it's impracticable for the agent to directly take all cached block bodies as input. It's necessary to reduce the sizes of the state space and action space. Thus, while our agent is obtaining states from the environment, it considers only 128 cached items that are randomly selected from the $\mathscr{C}$ ones. Although the agent cannot fully consider all cached items at once in doing so, it can still work well because each cached item has an equal chance to be selected if the agent interacts with the environment frequently enough. It lightens the model and makes it practical for these resource-constrained nodes to use the agent. Training with experience replay enables the agent to continuously learn from the historical interactions, which means it can take the previously selected cached items into account while making decisions.

The simulated DRL environment includes 30 smart contracts and a ledger that can cache at most $\mathscr{C} = 10,000$ blocks. The delays for requesting the block bodies are subject to an even distribution ranging from 1 ms to 10 ms. A smart contract always reads or writes a random part of specific keys in the ledger. The probabilities that data in block bodies are read or written by the contracts are subject to Zipf distribution, which is implemented using the Numpy Zipf library and with parameter $\alpha > 1.0$. According to Zipf's law, the frequency of an item is the reciprocal of its rank multiplied by the highest frequency. The greater α is, the more concentrated the data distribution will be. Every time data in an uncached block item are requested, the agent will treat it as a candidate and decide on whether to replace a cached item with it. Following the above distribution setting, we keep generating new blocks and evaluate the performance of our agent.

In Figure 5, we plot the rewards returned by the environment under different values of α. As shown, the rewards all increase fast at the beginning, which implies that our agent only requires a few steps to learn how to improve the rewards. These rewards are mainly produced by the extrinsic part of the reward function because most of the initial cached items are not popular enough. Actions that replace them with more popular items can easily receive high extrinsic rewards, according to our definition. It also explains why the rewards start declining after their peaks and stabilize after 200,000 timesteps. As the decision-making process continues, the impact of the extrinsic reward becomes weak, because all the cached items tend to have approximate popularity, and performing a cache replacement action that involves two approximate items will only receive a small reward. Therefore, the rewards stabilize at similar values as intrinsic rewards have taken the dominant position. This figure testifies to the effect of the proposed reward function on the convergence performance.

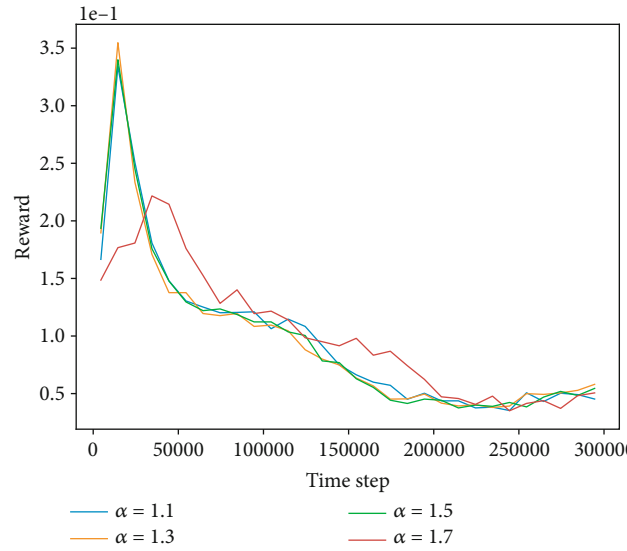

Figure 5: Reward fluctuations. Different Zipf parameters α from 1.1 to 1.7 are applied to the environment.

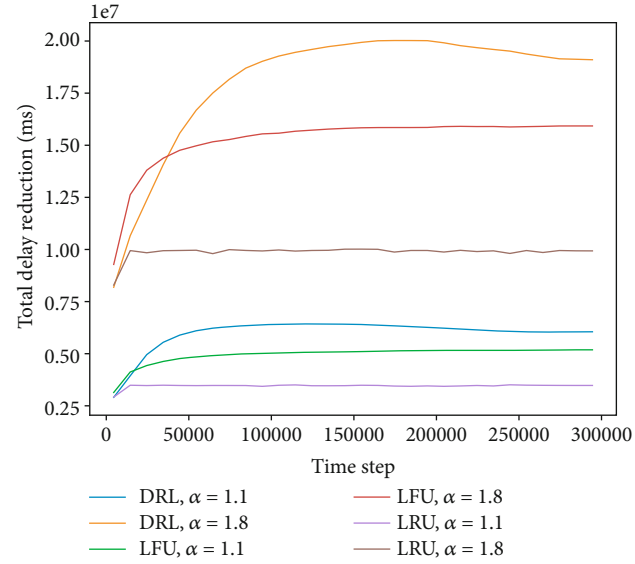

Figure 6: Total delay reduction fluctuations. LFU and LRU are used as baselines. $\alpha = 1.1$ and $\alpha = 1.8$ are applied to the environment.

In Figure 6, we evaluate the effect that our caching agent can make on reducing the time spent requesting data while executing smart contracts. As baselines, we introduce LRU and LFU. LRU records the last time that each block body is cached and selects the least recently requested item. LFU counts the number of requests for each block body and selects the item with the minimum number of requests. We can see that the total delay reductions of our agent lag behind the baselines. This is because our agent needs a few steps to experience various kinds of actions and learn to determine what actions are preferred. LRU and LFU are already deterministic caching strategies that do not require time to learn from the context. As expected, our agent soon outperforms LFU by a large margin under different values of

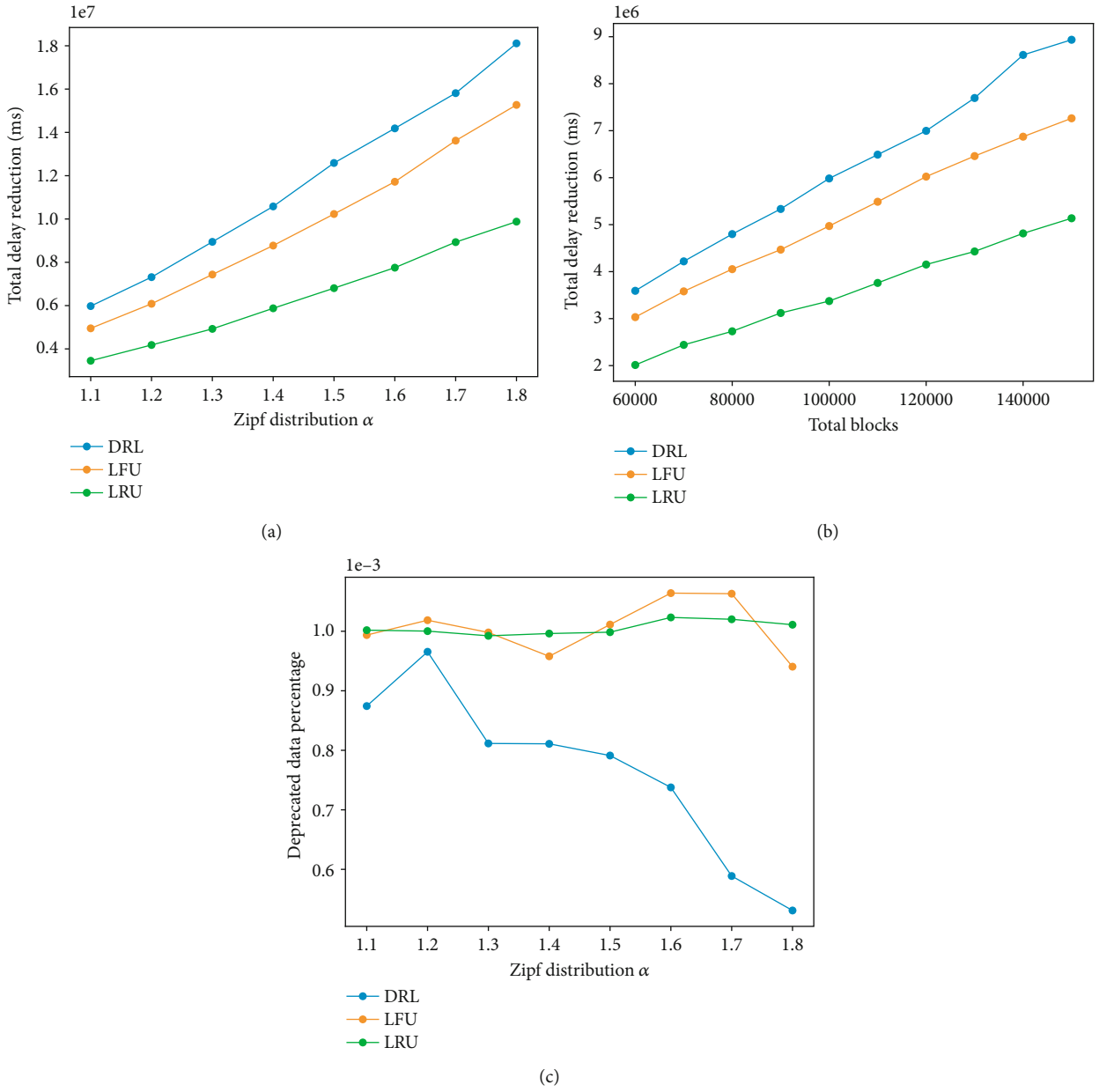

Figure 7: (a) Our proposed caching agent vs. LFU and LRU. Comparison of total delay reductions under various α from 1.1 to 1.8. (b) Our proposed caching agent vs. LFU and LRU. Comparison of total delay reductions under various numbers of total blocks from 60,000 to 150,000. (c) Our proposed caching agent vs. LFU and LRU. Comparison of deprecated data percentages under various α from 1.1 to 1.8.

α. The results show that our agent can effectively shorten the time of requesting data from other nodes and therefore improve the efficiency of smart contracts on IIoT nodes with limited storage.

In Figure 7(a), we study the relationship between parameter α and delay reduction. The smaller the value of α is, the more difficult the caching tasks will be due to more data items that are required to be considered. As illustrated, the total delay reductions of our scheme are around 13% better than LFU when $\alpha = 1.8$. This ratio can increase up to 20% as the value of α decreases to 1.1. The advantage over the baselines is more significant when the popularity of data items is more dispersed. This figure shows that our agent offers good adaptability to environments, especially those with extremely diverse popularity distributions.

In Figure 7(b), we grow the number of total blocks in the environment to explore whether the agent can provide equal caching performance under different cache capacities. Note that, the value of $\mathscr{C}$ we set always equals 10% of the number of total blocks. Setting $\alpha = 1.1$, we, respectively, evaluate the total delay reductions under numbers of blocks from 60,000 to 150,000. Our agent outperforms LFU by around 15% when ledger capacity $\mathscr{C} = 6,000$. As the value of $\mathscr{C}$ increases to 15,000, this advantage can be over 25%. It implies that our agent can offer satisfactory caching performance when it handles a caching problem that involves a large number of data items.

In Figure 7(c), we try to verify whether the agent can learn to avoid caching those blocks, whose data are not fresh. As aforementioned, the values of keys can be updated or deleted. Some values defined in a block might have been deprecated and will not be read anymore. We call them deprecated data. If the value of a key is frequently updated, the freshness of the block where its latest value is defined will be undermined. In contrast, if the value of a key is frequently read but rarely updated, the block which contains its latest value should be more likely to be cached. As a metric, we use the percentage of deprecated data to measure how fresh a cached block is. As shown, our scheme can effectively reduce the percentage of deprecated data compared to the baselines, especially when the popularity distribution is concentrated. As the value of α increases, the percentage gets lower. When $\alpha = 1.8$, it can provide a deprecated data percentage that is 75% lower than LFU. The results show that our agent has the ability to avoid caching those nonfresh blocks.

6. Conclusions

In this paper, we proposed a caching-enabled permissioned blockchain scheme for IIoT based on DRL, which is aimed at helping IIoT nodes improve smart contract execution efficiency when they do not have sufficient storage resources to maintain complete blockchain ledgers. First of all, we optimized the ledger structure, where block bodies are optional to be stored and a new field *VariationSet* is introduced into the block header to enable IIoT nodes to reach consensus faster without concrete block data. Focusing on the characteristics of the IIoT, we designed a general network architecture and the corresponding transaction flow. Afterward, we formulated the caching problem according to our defined system model as an MDP. Proper features about keys, blocks, and smart contracts are taken into consideration in the decision-making process. Additionally, an extrinsic reward is introduced into the reward function to help the caching agent faster converge. Finally, we implemented a lightweight caching agent based on deep Q-learning. The extensive experimental results show that our proposed scheme can effectively reduce the costs of disseminating blocks and the execution delay of smart contracts on IIoT nodes that can only hold limited blockchain data.

7. Future Work

Blockchain oracles, which provide a mechanism to fetch external information for smart contracts, have shown the potential to improve the interoperability of blockchain systems [48]. We believe that this technology is important for IIoT blockchains to access extremely large-scale off-chain data. However, a smart contract that requires oracles to be involved can generate many transactions that do not contain meaningful blockchain data. To better support the IIoT blockchains that work with oracles, we plan to explore how to enable our scheme to avoid caching those meaningless oracle-related transactions to improve storage efficiency in the future.

Data Availability

The data used or analysed during the current research are available from the corresponding author on reasonable request.

Disclosure

A preprint has previously been published [49].

Conflicts of Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The research was supported in part by the Guangxi Science and Technology Project (No. 2021AA11006), the National Natural Science Foundation of China (Nos. 62166004, U21A20474), the Center for Applied Mathematics of Guangxi Normal University, the Guangxi "Bagui Scholar" Teams for Innovation and Research Project, the Guangxi Talent Highland Project of Big Data Intelligence and Application, the Guangxi Collaborative Center of Multisource Information Integration and Intelligent Processing, Innovation Project of Guangxi Graduate Education (No. YCBZ2021038), and the 14th Five-Year Plan of Guangxi Education Science (No. 2023B324).

References

[1] S. Li, X. Li Da, and S. Zhao, "The internet of things: a survey," *Information Systems Frontiers*, vol. 17, no. 2, pp. 243–259, 2015.

[2] L. Da Xu, W. He, and S. Li, "Internet of things in industries: a survey," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[3] S. Zhao, S. Li, and Y. Yao, "Blockchain enabled industrial internet of things technology," *IEEE Transactions on Computational Social Systems*, vol. 6, no. 6, pp. 1442–1453, 2019.

[4] H.-N. Dai, Z. Zheng, and Y. Zhang, "Blockchain for internet of things: a survey," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8076–8094, 2019.

[5] D. Yaga, P. Mell, N. Roby, and K. Scarfone, "Blockchain technology overview," 2019, https://arxiv.org/abs/1906.11078.

[6] T. Alladi, V. Chamola, R. M. Parizi, and K.-K. R. Choo, "Blockchain applications for industry 4.0 and industrial IoT: a review," *IEEE Access*, vol. 7, pp. 176935–176951, 2019.

[7] Z. Li, J. Kang, Y. Rong, D. Ye, Q. Deng, and Y. Zhang, "Consortium blockchain for secure energy trading in industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 8, pp. 1–3700, 2017.

[8] J. Wan, J. Li, M. Imran, D. Li, and Fazal-e-Amin, "A blockchain-based solution for enhancing security and privacy in smart factory," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3652–3660, 2019.

[9] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[10] B. Liu, X. L. Yu, S. Chen, X. Xu, and L. Zhu, "Blockchain based data integrity service framework for IoT data," in *2017 IEEE International Conference on Web Services (ICWS)*, pp. 468–475, Honolulu, HI, USA, 2017.

[11] Y. Hassanzadeh-Nazarabadi, A. Küpçü, and Ö. Özkasap, "Lightchain: scalable DHT-based blockchain," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 10, pp. 2582–2593, 2021.

[12] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Playing atari with deep reinforcement learning," 2013, https://arxiv.org/abs/1312.5602.

[13] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[14] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International Conference on Machine Learning*, pp. 1995–2003, New York City, NY, USA, 2016.

[15] Y. Liu, K. Wang, Y. Lin, and X. Wenyao, "LightChain: a lightweight blockchain system for industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3571–3581, 2019.

[16] O. I. Khalaf and G. M. Abdulsahib, "Optimized dynamic storage of data (ODSD) in IoT based on blockchain for wireless sensor networks," *Peer-to-Peer Networking and Applications*, vol. 14, no. 5, pp. 2858–2873, 2021.

[17] W. Liang, Y. Fan, K.-C. Li, D. Zhang, and J.-L. Gaudiot, "Secure data storage and recovery in industrial blockchain network environments," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 10, pp. 6543–6552, 2020.

[18] C. Li, J. Zhang, X. Yang, and L. Youlong, "Lightweight blockchain consensus mechanism and storage optimization for resource-constrained IoT devices," *Information Processing & Management*, vol. 58, no. 4, article 102602, 2021.

[19] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 17–30, New York City, NY, USA, 2016.

[20] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: a secure, scale-out, decentralized ledger via sharding," in *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 583–598, San Francisco, CA, USA, 2018.

[21] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: scaling blockchain via full sharding," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pp. 931–948, Toronto, ON, Canada, 2018.

[22] K. Toyoda, M. Shakeri, X. Chi, and A. N. Zhang, "Performance evaluation of ethereum-based on-chain sensor data management platform for industrial IoT," in *2019 IEEE International Conference on Big Data (Big Data)*, pp. 3939–3946, Los Angeles, CA, USA, 2019.

[23] Z. Xu, J. Zhang, Z. Song, Y. Liu, J. Li, and J. Zhou, "A scheme for intelligent blockchain-based manufacturing industry supply chain management," *Computing*, vol. 103, no. 8, pp. 1771–1790, 2021.

[24] B.-G. Jeong, T.-Y. Youn, N.-S. Jho, and S. U. Shin, "Blockchain-based data sharing and trading model for the connected car," *Sensors*, vol. 20, no. 11, p. 3141, 2020.

[25] L. Gao, W. Celimuge, T. Yoshinaga, X. Chen, and Y. Ji, "Multi-channel blockchain scheme for internet of vehicles," *IEEE Open Journal of the Computer Society*, vol. 2, pp. 192–203, 2021.

[26] X. He, K. Wang, and X. Wenyao, "QoE-driven content-centric caching with deep reinforcement learning in edge-enabled IoT," *IEEE Computational Intelligence Magazine*, vol. 14, no. 4, pp. 12–20, 2019.

[27] Y. Wang, Y. Li, T. Lan, and V. Aggarwal, "Deepchunk: deep q-learning for chunk-based caching in wireless data processing networks," *IEEE Transactions on Cognitive Communications and Networking*, vol. 5, no. 4, pp. 1034–1045, 2019.

[28] J. Shi, L. Zhao, X. Wang, W. Zhao, A. Hawbani, and M. Huang, "A novel deep Q-learning-based air-assisted vehicular caching scheme for safe autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 7, pp. 4348–4358, 2021.

[29] L. Li, Y. Xu, J. Yin et al., "Deep reinforcement learning approaches for content caching in cache-enabled D2D networks," *IEEE Internet of Things Journal*, vol. 7, no. 1, pp. 544–557, 2020.

[30] M. Chen, W. Guangming, Y. Zhang, Y. Lin, Y. Zhang, and J. Li, "Distributed deep reinforcement learning-based content caching in edge computing-enabled blockchain networks," in *2021 13th International Conference on Wireless Communications and Signal Processing (WCSP)*, pp. 1–5, Changsha, China, 2021.

[31] J. Yin, X. Yang, Q. Pei et al., "SmartDID: a novel privacy-preserving identity based on blockchain for IoT," *IEEE Internet of Things Journal*, vol. 10, no. 8, pp. 6718–6732, 2023.

[32] E. Androulaki, A. Barger, V. Bortnikov et al., "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference*, pp. 1–15, Porto, Portugal, 2018.

[33] W. Liang, M. Tang, J. Long, X. Peng, X. Jianlong, and K.-C. Li, "A secure fabric blockchain-based data transmission technique for industrial internet-of-things," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3582–3592, 2019.

[34] H. Liu, D. Han, and D. Li, "Fabric-IoT: a blockchain-based access control system in IoT," *IEEE Access*, vol. 8, pp. 18207–18218, 2020.

[35] X. Xu, X. Wang, Z. Li et al., "Mitigating conflicting transactions in hyperledger fabric-permissioned blockchain for delay-sensitive IoT applications," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10596–10607, 2021.

[36] I. Surjandari, H. Yusuf, E. Laoh, and R. Maulida, "Designing a permissioned Blockchain network for the halal industry using Hyperledger fabric with multiple channels and the raft consensus mechanism," *Journal of Big Data*, vol. 8, no. 1, pp. 1–16, 2021.

[37] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX Annual Technical Conference ({USENIX}{ATC} 14)*, pp. 305–319, Philadelphia, PA, USA, 2014.

[38] Y. Xiao, N. Zhang, W. Lou, and Y. Thomas Hou, "A survey of distributed consensus protocols for blockchain networks," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 1432–1465, 2020.

[39] C. Zhong, M. Cenk Gursoy, and S. Velipasalar, "A deep reinforcement learning-based framework for content caching," in *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*, pp. 1–6, Princeton, NJ, USA, 2018.

[40] C. Zhong, M. Cenk Gursoy, and S. Velipasalar, "Deep reinforcement learning-based edge caching in wireless networks," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 1, pp. 48–61, 2020.

[41] C. Song, X. Wenxiang, W. Tingting, Y. Shimao, P. Zeng, and N. Zhang, "QoE-driven edge caching in vehicle networks based on deep reinforcement learning," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 6, pp. 5286–5295, 2021.

[42] H. Zhu, Y. Cao, X. Wei, W. Wang, T. Jiang, and S. Jin, "Caching transient data for internet of things: a deep reinforcement learning approach," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2074–2083, 2019.

[43] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, Montreal, Quebec, Canada, 2009.

[44] Y. Hu, W. Wang, H. Jia et al., "Learning to utilize shaping rewards: a new approach of reward shaping," *Advances in Neural Information Processing Systems*, vol. 33, pp. 15931–15941, 2020.

[45] Y. Wu and Y. Tian, "Training agent for first-person shooter game with actor-critic curriculum learning," in *International Conference on Learning Representations*, Toulon, France, 2017.

[46] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[47] A. Demers, D. Greene, C. Hauser et al., "Epidemic algorithms for replicated database maintenance," in *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pp. 1–12, Vancouver, British Columbia, Canada, 1987.

[48] S. K. Ezzat, Y. N. M. Saleh, and A. A. Abdel-Hamid, "Blockchain oracles: state-of-the-art and research directions," *IEEE Access*, vol. 10, pp. 67551–67572, 2022.

[49] P. Liu, C. Yao, C. Li, S. Zhang, and X. Li, *A Caching-Enabled Permissioned Blockchain Framework for Industrial Internet of Things based on Deep Reinforcement Learning*, Research Square Preprint, 2022.