SPRING: Improving the Throughput of Sharding Blockchain via Deep Reinforcement Learning Based State Placement

Pengze Li
Peking University
Beijing, China
lipengze@pku.edu.cn

Mingxuan Song
Peking University
Beijing, China
songmingxuan@stu.pku.edu.cn

Mingzhe Xing
Peking University
Beijing, China
mzxing@stu.pku.edu.cn

Zhen Xiao*
Peking University
Beijing, China
xiaozhen@pku.edu.cn

Qiuyu Ding
Peking University
Beijing, China
dingqiuyu@stu.pku.edu.cn

Shengjie Guan
Peking University
Beijing, China
guanshengjie@stu.pku.edu.cn

Jieyi Long
Theta Labs, Inc.
San Jose, USA
jieyi@thetalabs.org

ABSTRACT

Sharding provides an opportunity to overcome the inherent scalability challenges of the blockchain, which is the infrastructure for the next generation of the Web. In a sharding blockchain, the state is partitioned into smaller groups known as "shards." Since the states are placed on different shards, cross-shard transactions are inevitable, which is detrimental to the performance of the sharding blockchain. Existing solutions place states based on heuristic algorithms or redistribute states via graph-partitioning-based methods, which are either less effective or costly. In this paper, we present *SPRING*, the first deep-reinforcement-learning(DRL)-based sharding framework for state placement. SPRING formulates the state placement as a Markov Decision Process, which considers the cross-shard transaction ratio and workload balancing and employs DRL to learn the effective state placement policy. Experimental results based on real Ethereum transaction data demonstrate the superiority of SPRING compared to other state placement solutions. In particular, it decreases the cross-shard transaction ratio by up to 26.63% and boosts throughput by up to 36.03%, all without unduly sacrificing the workload balance among shards. Moreover, updating the training model and making decisions takes only 0.1s and 0.002s, respectively, which shows the overhead is acceptable.

CCS CONCEPTS

• **Computer systems organization** → *Peer-to-peer architectures.*

*Corresponding author.

WWW '24, May 13–17, 2024, Singapore, Singapore.

ACM ISBN 979-8-4007-0171-9/24/05…$15.00
https://doi.org/10.1145/3589334.3645386

KEYWORDS

blockchain; sharding; scalability; deep reinforcement learning

ACM Reference Format:
Pengze Li, Mingxuan Song, Mingzhe Xing, Zhen Xiao, Qiuyu Ding, Shengjie Guan, and Jieyi Long. 2024. SPRING: Improving the Throughput of Sharding Blockchain via Deep Reinforcement Learning Based State Placement. In *Proceedings of the ACM Web Conference 2024 (WWW '24), May 13–17, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3589334.3645386

1 INTRODUCTION

Blockchain is the technological backbone of Web 3.0, enabling a decentralized, secure, and user-empowered internet. However, the blockchain cannot scale well because of the scalability trilemma [4]. Breaking the scalability trilemma poses a significant challenge in blockchain technology as it requires improving throughput while maintaining a balance among safety, decentralization, and scalability. Among various scaling solutions, sharding [12, 13, 19, 27, 41, 50] stands out as a promising approach for addressing the scalability trilemma. Sharding adopts a "divide and conquer" approach by partitioning the blockchain into smaller segments called *shards*. Each shard stores a portion of the blockchain's state and can process transactions (*TX*) independently and concurrently, thereby enhancing the performance of the blockchain. Practically, the state in the blockchain can be represented by an *address*, a unique identifier for blockchain users. A corresponding address will be created whenever a new state is stored within the blockchain. This paper concentrates on the account-balance data model, wherein the address signifies either a user or a smart contract [43].

Although enhanced by sharding, the throughput of sharding blockchains cannot scale linearly with increasing shards, owing to the time-consuming cross-shard transactions (*CST*). The CST is a special TX that is used to handle the scenario where the states of TX participants reside on different shards [23, 41, 47]. CSTs are common in sharding blockchains. When there are more than 16 shards, over 95% of TXs are CSTs [23, 41]. Therefore, reducing CST is crucial to improve the sharding blockchains' throughput further.

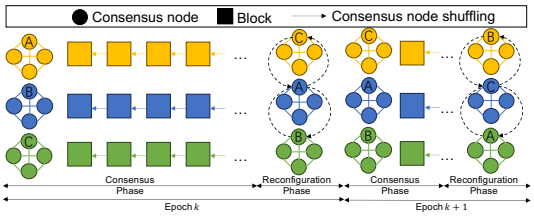


Figure 1: The two phases in each epoch of a sharding blockchain. In the consensus phase, TXs in the upcoming blocks are processed. In the reconfiguration phase, the consensus nodes are redistributed for security reasons.

The key to reducing CST lies in smartly placing states into different shards. However, there is a trade-off between minimizing the number of CST and maintaining a balanced workload distribution among shards. For instance, placing all states in the same shard could eliminate CST but regress to a standalone blockchain scenario. In this paper, we refer to the challenge of reducing CST while maintaining a balanced workload as the *state placement* problem.

The reduction of CSTs can be achieved in different operational phases of a sharding blockchain. As depicted in Fig. 1, a sharding blockchain protocol encompasses multiple consecutive epochs, and each epoch contains the *consensus phase* and the *reconfiguration phase* [13, 26]. The former phase involves processing TXs and generating blocks, utilizing *state placement* to minimize CSTs. The latter phase occurs every few blocks by shuffling consensus nodes into different shards for security reasons and implementing *state redistribution* to minimize CSTs.

In the consensus phase, the new state is created and placed for the first time. Existing state placement solutions [12, 19, 20, 41, 50] are heuristic in nature, which are simple but less effective in solving the state placement problem.

In the reconfiguration phase, the state is redistributed through the reconfiguration of consensus nodes. As TXs in blockchain reflect user activity patterns, which are typically purposeful and profit-driven, the TX pattern has rich spatial-temporal (*ST*) characteristics [6, 18, 52]. Based on this, the state redistribution method [13, 23] utilizes community detection and graph partitioning techniques applied to TX graphs to redistribute states. The ST characteristics of TXs are also valuable in solving the state placement problem as they reveal the locations of future states with TXs with that new state [6, 18, 44, 52]. However, the TX graph may contain millions of nodes (addresses) and edges (TX relationships), causing the partitioning process to be time-consuming, taking hundreds of seconds to complete [23]. Consequently, these methods are not feasible in the consensus phase.

To provide an efficient state placement framework in the consensus phase, we propose a new **S**tate **P**lacement framework based on deep **R**einforcement learn**ING** [25] (named **SPRING**) that utilizes the ST characteristics of TXs. As the block can be viewed as a sequence of TXs containing many new states, state placement can be viewed as a sequential decision problem suitable for reinforcement learning (RL) [39] approaches. Additionally, since the change of characteristics does not have an explicit pattern as the blockchain evolves [6, 18, 52], RL can adapt dynamically to capture changes that heuristics may fail to grasp. Experimental results based on real Ethereum TX data show that SPRING can solve the state placement problem effectively in different periods of real-world TXs.

This paper makes the following contributions:

- To the best of our knowledge, SPRING is the first to solve the state placement problem with RL. We formulate the Markov Decision Process for state placement in sharding blockchains, converting the state placement into a sequential decision-making task. Moreover, we propose a sharding blockchain system model and a decentralized agent deployment and training protocol to apply SPRING into sharding blockchains.
- Compared to current DRL-based sharding blockchains, SPRING is the first to use the ST characteristics of real Ethereum TX for model training and use emulation to demonstrate the performance improvement of DRL.
- We evaluate the effectiveness of SPRING on a sharding blockchain testbed deployed on the Alibaba Cloud where the test data set consists of millions of real Ethereum TXs from the years 2015, 2019, and 2023. SPRING outperforms four baselines in all three time periods. SPRING achieves a throughput improvement of 36.09% and a reduction in the CST ratio of 26.63%. SPRING also strikes a good balance between reducing the CST ratio and maintaining workload balance.

2 BACKGROUND AND RELATED WORK

2.1 Deep Reinforcement Learning

RL [39] is a prominent method for addressing sequential decision problems [2]. RL relies on an *agent* to take *actions* continually in an *environment*, receive *rewards* and the new state from the environment based on the outcomes of the actions, and adjust its *policy* to optimize long-term cumulative rewards. Deep Reinforcement Learning (DRL) [25] combines the strengths of deep learning [22] and RL and has been widely applied in sequential decision problems [28, 29, 45, 46].

State in RL and blockchains. The *state* has different meanings in RL and blockchains. In RL, it is the agent's current view of the environment. In blockchains, state refers to the latest, agreed-upon version of the system's data, like account balances, stored in consensus nodes. To avoid ambiguity, the address is used interchangeably to represent the state in blockchain in this paper.

2.2 Sharding Blockchain and Cross-Shard Transactions

Several sharding blockchain like [7, 13, 19, 27, 50] have been designed. The consensus phase in sharding blockchains includes intra-shard and cross-shard consensus. The former is relatively simple, while the latter requires interoperability [15] to handle cross-shard TXs, which is more complex. Common methods for cross-shard TX processing include the lock-mint two-phase commit mechanism [19], relay-based approaches [23, 41], and appointing one special shard to handle all cross-shard TXs [12]. To maintain the safety of the entire system, shard reconfiguration is needed to shuffle consensus nodes across all shards. However, shuffling nodes introduces challenges in efficiently transferring and synchronizing the required state across shards, which is orthogonal to SPRING.

2.3 State Redistribution

BrokerChain [13] is a cross-shard protocol that aims to reduce cross-shard TXs through Metis [17]-based fine-grained state-graph partitioning and state segmentation mechanisms. To reduce the computation time in BrokerChain, the Constrained Label Propagation Algorithm (CLPA) [23] is proposed. The CLPA is a community detection mechanism that requires less time than Metis.

Overall, state redistribution solutions are time-consuming and storage-space-consuming. However, since these solutions and SPRING work in different phases, they are orthogonal and can be combined to reduce cross-shared TXs better. We leave this to future work as we focus on the state placement in the consensus phase.

2.4 RL in Sharding Blockchains

SkyChain [51] is the first to apply DRL in sharding blockchains. SkyChain proposes a DRL-based sharding blockchain protocol that enables a sharding blockchain to dynamically change its number of shards, block size, and shard reconfiguration interval at run-time. As other DRL-based sharding blockchains [47–49] share similar designs with SkyChain, we only reproduce SkyChain for comparison.

Common limitations: Currently, RL-based sharding blockchain is still in its early stages. There are three main limitations of current DRL-based sharding blockchains: 1) The state space design does not consider the state of each shard. SkyChain only considers the total number of consensus nodes and pending TXs, failing to reflect the situation at the shard level. 2) The action space design of the agent is inappropriate. In current DRL-based solutions, the agent's actions include adjusting the reconfiguration interval and the block size. In a blockchain, these parameters are highly related to the consensus algorithm and cannot be arbitrarily modified. For instance, increasing the block size leads to longer TX propagation time, and may expose the blockchain to security attacks such as double-spending attacks [16]. 3) The simulated training dataset does not accurately reflect the ST characteristics of real TXs. Overall, the problem they solve is more like a general scheduling task for a cloud computing platform: when the load (TXs) increases and there are more resources (consensus nodes), the number of machines (shards) is increased, which does not truly reflect the characteristics of a sharding blockchain. Consequently, current DRL-based solutions cannot solve the state placement problem. Therefore, we propose SPRING to address the limitations of previous methods.

3 SHARDING BLOCKCHAIN DESIGN

This section provides an overview of *SPRINGChain*, the sharding blockchain where SPRING can be applied, to show the feasibility of integrating SPRING in sharding blockchains.

3.1 Basic Design and Assumptions

Similar to prior work [27, 50], we use *epochs* as shown in Fig. 1 to represent the term of the consensus nodes. Each epoch spans several consensus rounds for block production. In the reconfiguration phase, a verifiable random function [30] is applied to generate unpredictable and bias-resistant randomness, called *Epoch Randomness* (ER). Furthermore, we assume that all nodes have equal computational power.

As blockchain is decentralized, there are *malicious* nodes that stage attacks to the blockchain. We assume that the adversarial parties cannot control more than $\frac{1}{3}$ of the consensus nodes and cannot forge signatures. In practice, this is attainable through various Sybil attack prevention mechanisms, such as those employed by well-established blockchains like Bitcoin [31] and Ethereum [43], including proof-of-work (PoW) [31] and proof-of-stake [43]. In SPRINGChain, we use the PoW to prevent the Sybil attack. PoW requires nodes seeking to join the blockchain to solve a puzzle, and the last few bits of the solution string indicate which shard the node belongs to. All nodes in SPRINGChain are connected by a partially synchronous [10] peer-to-peer network, where the network may partition, but it will heal after an unknown amount of time.

3.2 System Model

Fig. 2 provides an overview of SPRINGChain. As many sharding blockchains [8, 12, 13, 23] have established, SPRINGChain also consists of two types of shards: *A-Shard* and *T-Shard*. SPRINGChain requires one A-Shard and k T-Shards.

- **T-Shard**: The T-Shard refers to the *TX shard*, which concurrently verifies and processes TXs.
- **A-Shard**: The A-Shard refers to the *agent shard*, which receives users' TX requests and decides which T-Shard the TX should be sent to, thus completing state placement. The DRL training occurs in the consensus nodes of A-Shard.

Similar to the state-of-the-art sharding blockchains [19, 27, 50, 51], both the A-Shard and T-Shard in SPRING adopt a Byzantine fault tolerant (BFT)-based consensus protocol, PBFT [5] protocol, as the intra-shard consensus protocol. In addition, a relay-based [23, 41] CST processing model is adopted. In this model, the TX is first processed on the source blockchain, and then the result is relayed to the target blockchain to finalize the CST. The workflow of the sharding protocol shown in Fig. 2 is described as follows:

(1) In each consensus round, the leader in the A-Shard selects n_t TXs from its TX mempool to produce the state placement result in the form of k TX batches. After reaching a consensus on the placement result, the batches are sent to the corresponding T-Shards.
(2) T-Shards verify and execute the TXs, and only valid TXs are included in the block. Additionally, the new addresses corresponding to the new states and the number of CSTs are recorded in the new block.
(3) Finally, by observing the new blocks of T-Shards, consensus nodes in A-Shard record the location of new states, the number of CSTs, and the total number of TXs in each T-Shard for further agent training.

SPRING assigns the TX containing the new address to a specific T-Shard to place the new state. When an address appears on the blockchain for the first time, it indicates that a new state containing the metadata, like the balance of the address, is created in the blockchain. For simplicity, assigning the TX that contains a new address in this paper actually means placing a new state.

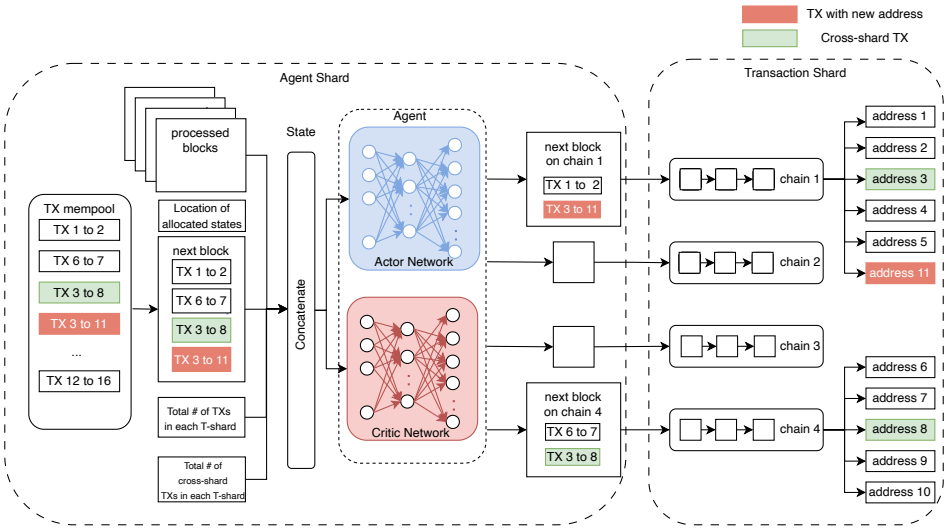


Figure 2: SPRINGChain workflow overview. The client submits TXs to the A-Shard, then the leader in the A-Shard selects TXs and places new states to T-Shards.

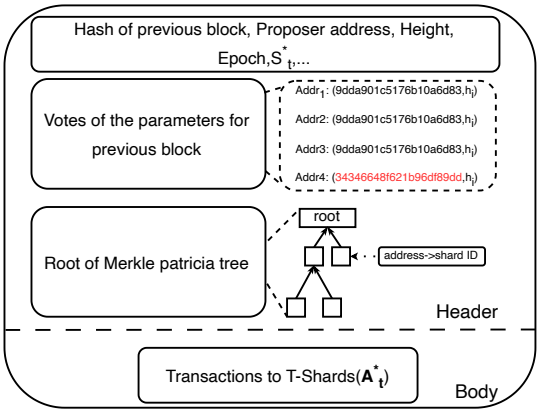


Figure 3: The data structure of the block in A-Shard at height h_{i+1}. The red text indicates the vote is from a malicious node. The S_t^* and A_t^* are the state of T-Shard and state placement results, respectively, which are used in agent training.

3.3 Decentralized Agent Deployment and Training in A-Shard

To avoid centralization issues, we describe a decentralized agent deployment and training protocol. Each consensus node in A-Shard has a copy of the agent with identical initial parameters. Moreover, A-Shard's genesis block's hash is used as the randomness seed. The agent update and the state placement actions are determined via consensus. In the consensus phase, one node is selected as the proposer/leader based on ER and proposes a new block.

The data structure of a block in A-Shard is shown in Fig. 3, which contains the *header* and the *body*. The block header includes metadata, peers' votes, and the Merkle Patricia tree (MPT) root [43]. The leaf of the MPT stores the mapping from the address to the ID of T-Shards. With the MPT, the existence and location of the address can be queried efficiently with $O(\log n)$ complexity, where n is the number of addresses.

As all the validity of the content in A-Shard block can be verified via TXs, the basic safety and liveness property of the underlying consensus protocol is not compromised. Consequently, we use PBFT to illustrate how the proposer in A-Shard proposes a block and helps all consensus nodes update the agent consistently. The details of the security analysis of the protocol are in Appendix A.

1) Pre-Prepare. At the start of a consensus round, the leader broadcasts a *pre-prepare* message that contains the proposed block to its peers. As demonstrated in Fig. 3, the proposed block includes the state S_t^* of T-Shards that the leader observes in the block header, and the placement results A_t^* based on S_t^* in the block body. S_t^* and A_t^* are RL-related components which will be introduced in Section 4. Upon receiving the *pre-prepare* message, the consensus nodes will broadcast a *prepare* message containing the block's hash to guarantee that all peers received the same block.

2) Prepare. Upon receiving *prepare* of the same block from more than 2/3 of peers, the consensus node will verify A_t^* from the leader by creating their own $A_t^{'}$ with their local views of state $S_t^{'}$. A predefined threshold ϕ_a is set to tolerate the inconsistency of placement results caused by the nature of the distributed system. Since the placement result is k TX batches, the differences between $A_t^{'}$ and A_t^* can be evaluated by the Jaccard index [42]. If the difference is within ϕ_a, the consensus node broadcasts the *commit* message to confirm the proposed block's validity.

3) Commit. All nodes accumulate *commit* messages for the proposed block. If more than 2/3 of the consensus nodes send *commit* message for the proposed block, all consensus nodes will commit the block and update the local model. Besides, the leader will send the TX to the corresponding T-Shard with proof, which indicates the placement result is reached via a consensus round.

4 DRL DESIGN

We formulate the state placement as a Markov Decision Process (MDP) [35] , which is a powerful mathematical framework that captures the essence of sequential decision-making under uncertainty. The MDP can be represented as a tuple (S, A, R, P), where:

State, S: Existing modeling of sharding blockchain has limitations [13, 23, 47, 51]. Firstly, they do not consider the state of each shard. The state space in SPRING is designed to consider the current situation in each shard. Secondly, the state space should not use too much information as graph-partitioning-based solutions do since the size of the TX graph will continue to grow and increase the overhead of the blockchain node. Thirdly, part of the information in the TX graph will be outdated due to TXs' changing ST characteristics.

As shown in Fig. 2, when a new block arrives, the agent can observe the information of that block, such as the location of senders for each receiver, and the situations in all shards, such as the workload distribution. Besides, the type of the new address, whether it corresponds to a normal account or a smart contract, is also a factor worth considering, as their TX characteristics are different [6, 18, 52]. Overall, the state s is a $11k + 1$-dimensional vector represented as follows:

$$\begin{aligned} s = [&num\_tx_1^1, \ldots, num\_tx_k^5, \\ &cross\_tx_1^1, \ldots, cross\_tx_k^5, \\ &sender\_pos_1, \ldots, sender\_pos_k, flag\ F], \end{aligned}$$

where k represents the number of T-Shards. $num\_tx_i^w$ and $cross\_tx_i^w$ incorporate the concept of a dynamic sliding window. They refer to the total number of TXs and CSTs in the previous five blocks for the i-th T-Shard so that $w \in [1, 5]$ indicates the block index. This sliding window provides the temporal information for the agent because the order of elements in $num\_tx_i^w$ and $cross\_tx_i^w$ is set according to the newness of the block. Additionally, $sender\_pos_i$ signifies the position of all senders linked to new addresses within the current block, which reflects the spatial characteristics. The flag $F \in \{0, 1\}$ is a one-hot indicator indicating whether the address is a smart contract or an externally owned account.

At the start of each training step, $num\_tx_i^w$ and $cross\_tx_i^w$ are initialized based on the recent five processed blocks. Moreover, as the senders' address has been placed in earlier blocks, $sender\_pos_i$ is initialized based on the placement result of existing addresses. The *flag F* is also initialized based on the new address type.

Considering the feasibility and decentralization, the state should come from the publicly available content of blockchains, which limits the range of options. Although subject to this limitation, experimental results show that SPRING's state design can represent the situations in each shard and each new address to an acceptable extent without introducing a significant storage burden.

Action, A: Existing RL-based studies select adjusting the block size and reconfiguration interval as the agent's action, which is not applicable in the real sharding blockchain. To overcome this drawback, the agent in SPRING opts to directly determine which shard the new address will belong to, which is a more feasible action since it is independent of the parameters of the consensus process. Specifically, the action is an k-dimensional one-hot vector, represented as:

$$action = [a_1, a_2, \ldots, a_k], \tag{1}$$

where the value of a_i is either 0 or 1, indicating whether to assign a new address to the i-th T-Shard or not. The action space in SPRING is concise and directly places the address.

Reward, R: The reward is the beacon that guides the agent to achieve our design goal. In reward design, two indices, i.e., r_{cstr} and r_{wlb}, are defined to represent the ratio of CST and the workload balance situation across all shards in the *current block* after taking the action, respectively. The reward function r is defined as follows:

$$r_t = \lambda \cdot r_{cstr} + (1 - \lambda) \cdot r_{wlb} \tag{2}$$

$$r_{cstr} = \frac{\sum_{i=1}^{k} num\_tx_i}{\sum_{i=1}^{k} cross\_tx_i + \epsilon} \tag{3}$$

$$avg\_tx = \frac{\sum_{i=1}^{k} num\_tx_i}{k} \tag{4}$$

$$abs\_diff = \sum_{i=1}^{k} |num\_tx_i - avg\_tx| \tag{5}$$

$$r_{wlb} = \exp(-\beta \cdot abs\_diff), \tag{6}$$

where r_t represents the reward at the t-th step and λ is a weight parameter that balances the importance between the cross-shard TX ratio and workload balancing. The optimal λ settings are explored in Subsection 5.2. r_{cstr} is the inverse of the ratio of CST as our goal is to achieve a low ratio of CST, and ϵ is a small value like 10^{-6} to avoid division by zero. As for r_{wlb}, an exponential function of the absolute difference *abs_diff* represents the workload balance situations. To ensure the balance of the workload and easier training, we use the Laplace–Stieltjes transform [3] in r_{wlb}. Finally, β in r_{wlb} is to control the decay rate.

The overall objective R is to maximize the accumulated reward and can be defined as follows:

$$R = \mathbb{E}_{\tau \sim \pi_\theta} \Big[\sum_{t=0}^{T} \gamma^t r_t \Big],$$

where τ is the trajectory that represents an episode of state placement procedure, π_θ is the address policy parameterized by θ, and T is the total number of steps in the trajectory. Each step corresponds to a block. The γ is the discount factor. This paper applies Proximal Policy Optimization (PPO) [37] to optimize the agent. The details of PPO are in Appendix B.

Transition, P: The state transition function P defines the probability of reaching the new state s_{t+1} after taking action a_t under the given state s_t.

P works as follows: For each block, the total number of TXs $num\_tx_i^w$ and the number of cross-shard TXs $cross\_tx_i^w$ on each T-Shard is only changed at the start of a step according to the situation in T-Shards. The sender position $sender\_pos_i$ is changed according to each agent's actions within a block. After placing an address, $sender\_pos_i$ related to this address is updated. The flag is updated based on the type of the current address to be placed.

5 EVALUATION

5.1 Experimental Settings

For the evaluation, the A-Shard is implemented via simulation in Python 3.9 and the T-Shard is implemented via a blockchain testbed. The agent is trained on a machine with an Intel(R) Core(TM) i7-9750H CPU @ 2.60 GHz, 64 GB RAM, and an NVIDIA GeForce RTX 2080. Additionally, we verify the trained result with an open-sourced sharding blockchain testbed, BlockEmulator [14]. BlockEmulator uses PBFT as its intra-shard consensus protocol and a relay-based approach for CSTs, which is consistent with our design described in Section 3. The T-Shard is deployed on 64 ecs.c7.large instances from the same zone in Alibaba Cloud with 16 shards and four consensus nodes in each shard. To demonstrate the adaptability of SPRING, we use real Ethereum TX data from 2015, 2019, and 2023 [53], encompassing up to six million TXs.

Baselines. We choose the following four baselines to show the effectiveness of SPRING. The detailed introduction to these baselines are in Appendix C.

(1) **Random.** Random solutions in previous work [19, 50] randomly select a shard for the new state.
(2) **Shard Scheduler** [20]. Shard Scheduler assigns a new state to the shard with the least number of states.
(3) **Monoxide** [41]. Monoxide allocates the new state based on the last few hexadecimal digits of the corresponding address. For example, for a new address ending with "e" in a sharding chain system with eight shards, it will be placed in the 7th shard (e % 8 = 6).
(4) **SkyChain** [51]. We reproduce SkyChain by using its state and reward for state placement. SkyChain uses the total number of consensus nodes and pending TXs as its state and the throughput as its reward.

5.2 Hyper-parameters Settings

Table 1: Hyperparameters and their values

Hyperparameter	Value
DRL training batch, $batch$	2048
Learning rate, lr	3×10^{-4}
Clip value in PPO, $clip$	0.2
Discount factor, γ	0.99
Decay rate weight factor, β	0.1
Weight factor, λ	0.5
number of neurons in each layer, n_{neuron}	64

The hyperparameter settings used in SPRING are summarized in Table 1. We set the batch size for model training to $batch = 2048$. After comparing surrogate objectives, we find that the best learning effect on the environment policy is achieved when the clip value is set to 0.2. Moreover, we set the initial learning rate to $lr = 3 \times 10^{-4}$.

Choice of λ. λ represents the trade-off between reducing cross-shard transactions and achieving load balancing. When λ approaches 1, SPRING focuses on reducing cross-shard transactions without considering load balance. This approach can result in a new state being almost exclusively allocated to a single shard. While this may decrease the ratio of cross-shard transactions, it can reduce the throughput to the level of a regular blockchain without sharding. Conversely, when λ approaches 0, SPRING minimally reduces the ratio of cross-shard transactions, which can decrease the system's throughput to that of the Shard Scheduler [20].

Besides, the best λ setting would differ for different sharding blockchains. This is because which one of load balancing and CSTR has a big impact depends on the implementation and the assumption of the blockchain system. For instance, LB-Chain [24] assumes that the two sub-transactions of a CST are the same as a normal transaction, so it comes to the conclusion that load balancing impacts the performance of a sharding system most. However, other research [13, 32] views the CSTR as the most important factor.

The effectiveness of workload balance is contingent on the processing capacity of individual shards, and the influence of CSTR hinges on a blockchain system's method of managing cross-shard transactions. Given that sharding blockchain systems vary in their design and configuration for these factors, there is no one-size-fits-all optimal setting for λ. Consequently, the adjustment of λ must be tailored to the specific characteristics of the system implementing SPRING. We explored λ in 0.2, 0.5, 0.8 settings and found that the system's throughput improvement is optimal when λ is set at 0.5 for BlockEmulator.

5.3 Overhead of SPRING

For storage overheads, the size of an RL model in SPRING is about 30KB. The computational overhead of SPRING is twofold: 1) the cost of assigning an address (AA), including the time taken by the agent to take the action of deciding which shard to assign the address to, and 2) the cost of updating the training model (UTM), which only happens once per block. Based on the machine used in this experiment, AA costs about 0.002 seconds, and UTM costs about 0.1 seconds, which is much lower than the computational overhead of graph-partitioning-based solutions.

Moreover, the computational speed can still be improved. AA and UTM are vector computations, which are related to the computational power of the hardware [21]. Therefore, powerful hardware such as a graphics processing unit can accelerate both UTM and AA. Additionally, some of the addresses are already placed in previous blocks, and according to our evaluations, a block contains between 20 to 200 new addresses, which means that not all addresses need to be assigned and less computation time is required.

Overhead of A-Shard. Although all TXs are first sent to A-Shard, A-Shard will not become a bottleneck. For computing overhead, unlike previous work [7], A-Shard does not actually process TXs, which is the time-consuming part [33]. Transactions are stored in the mempool of the A-Shard node, an in-memory data structure with $O(\log n)$, even constant [9], insertion/deletion time, where n is the number of TXs. For storage overhead, unlike previous work [13, 23] which designates one shard to store the full state of the whole blockchain, A-Shard only stores $address \Rightarrow shard$ relation to place new states. Overall, even with normal computing resources, the computational cost of both network structures of SPRING is acceptable.

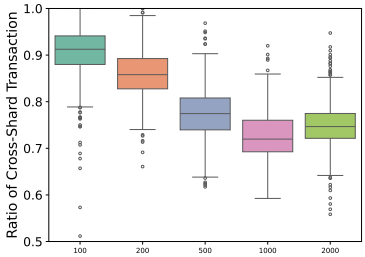


Figure 4: Box plot of CSTR for different block sizes with TX data in 2023

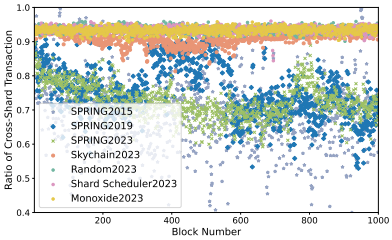


Figure 5: Scatter plot of CSTR for different algorithms

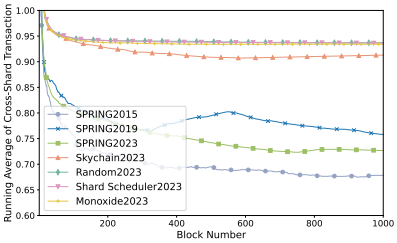
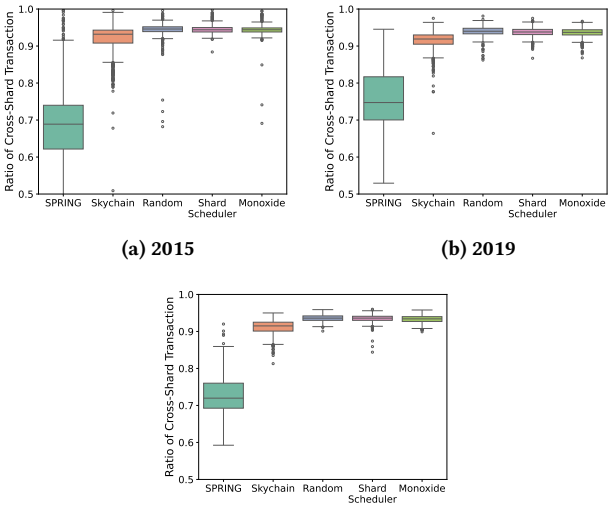


Figure 6: Running average of CSTR for different algorithms



(a) 2015 **(b) 2019**

(c) 2023

Figure 7: Box plot of CSTR for different algorithms with TX data from 2015, 2019 and 2023

5.4 Exploring Block Size for Improved Performance

Unlike DRL-based solutions mentioned in Subsection 2.4, the block size in SPRING is not adjustable to avoid potential issues. However, the block size is not set arbitrarily. In this experiment, we investigate the impact of block size on the cross-shard transaction ratio (*CSTR*). The block size n_t used in this experiment includes 100, 200, 500, 1000, and 2000. In each experiment, 1000 blocks are used to ensure the same number of training steps. Fig. 4 only shows the CSTR with different n_t trained on data from 2023, as results trained with data from other periods are similar. Among them, when $n_t = 1000$, the median value is the lowest. Additionally, the distribution of outliers for $n_t = 1000$ indicates better stability than other n_t settings.

Considering that the ST characteristics of TX data can change during runtime [6, 52], n_t can be seen as the *timing window* that represents the ST characteristics within n_t TXs. If n_t is too large, like 2000, the ST characteristics might have changed significantly within n_t blocks. Conversely, a too-small n_t, like 100 to 500, may not contain enough ST information for effective state placement. Consequently, based on the experimental observations from Fig. 4, we set n_t to 1000 in the rest of the experiments.

5.5 Performance Comparison

The performance experiments are conducted using a total of one million TXs in each period which corresponds to 1000 blocks as the training data, and various metrics such as CSTR and workload distribution are collected during the training process. These experiments aim to demonstrate the effectiveness of SPRING compared to the chosen baselines.

5.5.1 Cross-shard Transaction Ratio. Fig. 7a to Fig. 7c illustrates the CSTR using different state placement algorithms. For all algorithms, except SPRING and SkyChain, the CSTR is at around 94% in all periods, showing the effectiveness of DRL. Moreover, SPRING significantly outperforms all other baselines in all periods. As SPRING takes advantage of the ST characteristics of the TX data, it can make more judicious state placement decisions. Overall, SPRING achieves up to 26.63% reduction in CSTR compared to other baselines.

Fig. 5 and Fig. 6 take a deeper view into the training process. As the baselines have similar performance, we only show baseline results with data from 2023. Fig. 5 shows that in most of the training steps SPRING outperforms all the baselines in all periods. Besides, Fig. 6 demonstrates that SPRING can learn the ST characteristics to reduce CSTR efficiently, since we train the agent from scratch with data in all three periods. It is worth noting that, in the middle of the training with 2019 data, the CSTR rises. This could be the case of the change in the ST characteristics. For example, most TXs are issued from or to several addresses, with the counterpart addresses spread out across different shards. After this period, the CSTR decreases, showing SPRING possesses good adaptability.

For the performance of each algorithm, since SkyChain uses throughput as a reward, which is related to CSTR, it can also find a way to reduce CSTR to some extent. However, it does not consider the state within each shard and the location of the sender address associated with the receiver, so its actions are taken based on the less informative state compared to SPRING, resulting in a limited

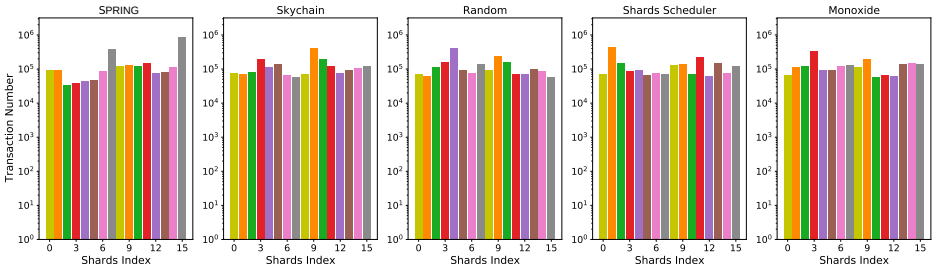


Figure 8: TX distribution of all shards in different algorithms with TX data in 2023

reduction in CSTR. The heuristic algorithm also has the same issue of not being able to allocate the state to an appropriate shard.

Overall, SPRING has good adaptability and successfully utilizes a DRL-based state placement approach that leverages the ST characteristics of TX data to reduce the CSTR.

5.5.2 Load Balancing Exploration. In this part, the load-balancing situation in five state placement strategies is analyzed. Fig. 8 shows the accumulated TX distribution in different state placement strategies for the 1000 blocks. The y-axis of Fig. 8 is in log-scale. Overall, there is a trade-off between the cross-shard ratio and workload balance. Previous work such as Monoxide does not guarantee optimization of both factors [13]. However, SPRING reduces the cross-shard ratio without excessively sacrificing workload balance.

It can be observed from Fig. 8 that all strategies suffer from some level of workload imbalance. The imbalance in all algorithms is attributed to the power law distribution of TXs, which has been observed in Ethereum TX data analysis [6, 52]. This indicates that a small fraction of addresses are responsible for a large number of TXs. Taking the situation in 2015 as an example, 9 out of 33,899 addresses contribute to 73.7% of TXs, following the power law distribution. These popular addresses are usually smart contracts or mine pools [6, 52]. These popular addresses are involved in many TXs across every shard. Therefore, if these TXs are in the shard containing the popular address, it would significantly reduce CSTs, but at the cost of disrupting load balance. The theoretical baseline for achievable reductions in CSTs, is contingent on the proportion of TXs associated with popular addresses. Moreover, since Fig. 8 represents the cumulative result, the difference between shards is smaller when a block contains only 1000 TXs, meaning that the influence of the imbalance could be amortized across all shards.

Furthermore, balance in address distribution does not guarantee balance in TX distribution. For instance, although the Shard Scheduler evenly places addresses across shards, the distribution of TXs is not even as shown in Fig. 8. This holds true for SPRING as well. For instance, although there are about 25% more addresses in T-Shard 7 (index starts from 0) compared to T-Shard 6, the difference in the number of TXs between them is only 11%. Additionally, the total number of addresses in 2019 and 2023 is greater than in 2015. This is because, as users become more concerned about privacy, mixing services [38] have been widely used since 2019, which generates many one-off addresses. The placement result of SPRING is consistent with the findings from Ethereum TX data analysis [6, 52], indicating that SPRING successfully captures the characteristics of the TX data.

Table 2: TPS of different algorithms

Algorithm	TPS	Algorithm	TPS
SPRING2015	486.8532	Monoxide2023	325.542
SPRING2019	426.424	ShardScheduler2023	321.957
SPRING2023	437.961	Random2023	326.691
		SkyChain2023	342.35

5.5.3 Throughput Exploration. The throughput is the number of TXs the sharding blockchain can process per second (TPS). We collect the TPS using BlockEmulator [14] and modify the state placement module of the BlockEmulator to verify the effectiveness of SPRING. As shown in Table 2, SPRING consistently outperforms the other algorithms. The performance of baselines is similar in all periods. Thus, we only present the result in 2023. Moreover, the TPS is improved up to 36.03% with 2023 TX data, which is higher than the reduction in CSTR, which is 20.99%. This indicates that CST's impact on the TPS is significant. Since the cross-shard needs an extra processing mechanism, it is more time-consuming. Consequently, reducing the CSTR does improve throughput, and SPRING outperforms other baselines.

6 CONCLUSION

In this paper, we present SPRING, a deep reinforcement learning (DRL)-based state placement method that first models the Markov Decision Process for state placement. Our solution minimizes the proportion of cross-shard transactions without unduly compromising the workload balance between different shards. In addition, our approach exploits the spatial-temporal properties of transaction data, resulting in improved overall system throughput. Compared to traditional graph partitioning methods, SPRING has lower computational and storage overheads. In future work, SPRING can be combined with a reconfiguration mechanism to improve the performance of sharding blockchains.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their comments. This work was supported by the Beijing Natural Science Foundation under Funding No. IS23055. The contact author is Zhen Xiao.

REFERENCES

[1] Abien Fred Agarap. 2018. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375* (2018).

[2] Andrew Gehret Barto, Richard S Sutton, and CJCH Watkins. 1989. *Learning and sequential decision making*. University of Massachusetts Amherst, MA.

[3] CJK Batty. 1990. Tauberian theorems for the Laplace-Stieltjes transform. *Trans. Amer. Math. Soc.* 322, 2 (1990), 783–804.

[4] Vitalik Buterin. 2023. *Blockchain Scalability Trilemma*. https://ethereum.org/en/upgrades/vision/

[5] Miguel Castro and Barbara Liskov. 2002. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)* 20, 4 (2002), 398–461.

[6] Ting Chen, Zihao Li, Yuxiao Zhu, Jiachi Chen, Xiapu Luo, John Chi-Shing Lui, Xiaodong Lin, and Xiaosong Zhang. 2020. Understanding ethereum via graph analysis. *ACM Transactions on Internet Technology (TOIT)* 20, 2 (2020), 1–32.

[7] Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, Ee-Chien Chang, Qian Lin, and Beng Chin Ooi. 2019. Towards scaling blockchain systems via sharding. In *Proceedings of the 2019 international conference on management of data*. 123–140.

[8] Sourav Das, Vinith Krishnan, and Ling Ren. 2020. Efficient cross-shard transaction execution in sharded blockchains. *arXiv preprint arXiv:2007.14521* (2020).

[9] Saulo Dos Santos, Chukwuka Chukwuocha, Shahin Kamali, and Ruppa K Thulasiram. 2019. An efficient miner strategy for selecting cryptocurrency transactions. In *2019 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 116–123.

[10] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 1988. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)* 35, 2 (1988), 288–323.

[11] Abdelatif Hafid, Abdelhakim Senhaji Hafid, and Mustapha Samih. 2020. A novel methodology-based joint hypergeometric distribution to analyze the security of sharded blockchains. *IEEE Access* 8 (2020), 179389–179399.

[12] Zicong Hong, Song Guo, Peng Li, and Wuhui Chen. 2021. Pyramid: A layered sharding blockchain system. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 1–10.

[13] Huawei Huang, Xiaowen Peng, Jianzhou Zhan, Shenyang Zhang, Yue Lin, Zibin Zheng, and Song Guo. 2022. BrokerChain: A Cross-Shard Blockchain Protocol for Account/Balance-based State Sharding. In *IEEE INFOCOM*.

[14] Huawei Huang, Guang Ye, Qinde Chen, Zhaokang Yin, Xiaofei Luo, Jianru Lin, Taotao Li, Qinglin Yang, and Zibin Zheng. 2023. BlockEmulator: An Emulator Enabling to Test Blockchain Sharding Protocols. *arXiv preprint arXiv:2311.03612* (2023).

[15] Hai Jin, Xiaohai Dai, and Jiang Xiao. 2018. Towards a novel architecture for enabling interoperability amongst multiple blockchains. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1203–1211.

[16] Ghassan O Karame, Elli Androulaki, and Srdjan Capkun. 2012. Double-spending fast payments in bitcoin. In *Proceedings of the 2012 ACM conference on Computer and communications security*. 906–917.

[17] George Karypis and Vipin Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing* 20, 1 (1998), 359–392.

[18] Arijit Khan. 2022. Graph Analysis of the Ethereum Blockchain Data: A Survey of Datasets, Methods, and Future Work. In *2022 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 250–257.

[19] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. 2018. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 583–598.

[20] Michał Król, Onur Ascigil, Sergi Rene, Alberto Sonnino, Mustafa Al-Bassam, and Etienne Rivière. 2021. Shard scheduler: object placement and migration in sharded account-based blockchains. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*. 43–56.

[21] Griffin Lacey, Graham W Taylor, and Shawki Areibi. 2016. Deep learning on fpgas: Past, present, and future. *arXiv preprint arXiv:1602.04283* (2016).

[22] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[23] Canlin Li, Huawei Huang, Yetong Zhao, Xiaowen Peng, Ruijie Yang, Zibin Zheng, and Song Guo. 2022. Achieving Scalability and Load Balance across Blockchain Shards for State Sharding. In *2022 41st International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 284–294.

[24] Mingzhe Li, Wei Wang, and Jin Zhang. 2023. LB-Chain: Load-Balanced and Low-Latency Blockchain Sharding via Account Migration. *IEEE Transactions on Parallel and Distributed Systems* (2023).

[25] Yuxi Li. 2017. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274* (2017).

[26] Yizhong Liu, Jianwei Liu, Marcos Antonio Vaz Salles, Zongyang Zhang, Tong Li, Bin Hu, Fritz Henglein, and Rongxing Lu. 2022. Building blocks of sharding blockchain systems: Concepts, approaches, and open problems. *Computer Science Review* 46 (2022), 100513.

[27] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. 2016. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 17–30.

[28] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM workshop on hot topics in networks*. 50–56.

[29] Hangyu Mao, Zhengchao Zhang, Zhen Xiao, Zhibo Gong, and Yan Ni. 2020. Learning agent communication under limited bandwidth by message pruning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 5142–5149.

[30] Silvio Micali, Michael Rabin, and Salil Vadhan. 1999. Verifiable random functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*. IEEE, 120–130.

[31] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review* (2008), 21260.

[32] Lan N Nguyen, Truc DT Nguyen, Thang N Dinh, and My T Thai. 2019. Optchain: optimal transactions placement for scalable blockchain sharding. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 525–535.

[33] Michael Pacheco, Gustavo Oliva, Gopi Krishnan Rajbahadur, and Ahmed Hassan. 2023. Is my transaction done yet? an empirical study of transaction processing times in the ethereum blockchain platform. *ACM Transactions on Software Engineering and Methodology* 32, 3 (2023), 1–46.

[34] Malay K Pakhira. 2014. A linear time-complexity k-means algorithm using cluster shifting. In *2014 international conference on computational intelligence and communication networks*. IEEE, 1047–1051.

[35] Martin L Puterman. 1990. Markov decision processes. *Handbooks in operations research and management science* 2 (1990), 331–434.

[36] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438* (2015).

[37] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[38] István András Seres, Dániel A Nagy, Chris Buckland, and Péter Burcsi. 2019. Mixeth: efficient, trustless coin mixing service for ethereum. *Cryptology ePrint Archive* (2019).

[39] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[40] Gerald Tesauro et al. 1995. Temporal difference learning and TD-Gammon. *Commun. ACM* 38, 3 (1995), 58–68.

[41] Jiaping Wang and Hao Wang. 2019. Monoxide: Scale out blockchains with asynchronous consensus zones. In *16th USENIX symposium on networked systems design and implementation (NSDI 19)*. 95–112.

[42] Wikipedia. 2023. *Jaccard Index*. https://en.wikipedia.org/wiki/Jaccard_index

[43] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151, 2014 (2014), 1–32.

[44] Mingzhe Xing, Shuqing Bian, Wayne Xin Zhao, Zhen Xiao, Xinji Luo, Cunxiang Yin, Jing Cai, and Yancheng He. 2021. Learning Reliable User Representations from Volatile and Sparse Data to Accurately Predict Customer Lifetime Value. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 3806–3816.

[45] Mingzhe Xing, Hangyu Mao, and Zhen Xiao. 2022. Fast and Fine-grained Autoscaler for Streaming Jobs with Reinforcement Learning. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (Vienna, Austria, 23-29 July 2022)(IJCAI 2022). ijcai. org, USA*. 564–570.

[46] Mingzhe Xing, Hangyu Mao, Shenglin Yin, Lichen Pan, Zhengchao Zhang, Zhen Xiao, and Jieyi Long. 2023. A Dual-Agent Scheduler for Distributed Deep Learning Jobs on Public Cloud via Reinforcement Learning. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2776–2788.

[47] Zhaoxin Yang, Ruizhe Yang, F Richard Yu, Meng Li, Yanhua Zhang, and Yinglei Teng. 2022. Sharded blockchain for collaborative computing in the Internet of Things: Combined of dynamic clustering and deep reinforcement learning approach. *IEEE Internet of Things Journal* 9, 17 (2022), 16494–16509.

[48] Shijing Yuan, Jie Li, Jinghao Liang, Yuxuan Zhu, Xiang Yu, Jianping Chen, and Chentao Wu. 2021. Sharding for blockchain based mobile edge computing system: A deep reinforcement learning approach. In *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 1–6.

[49] Jusik Yun, Yunyeong Goh, and Jong-Moon Chung. 2020. DQN-based optimization framework for secure sharded blockchain systems. *IEEE Internet of Things Journal* 8, 2 (2020), 708–722.

[50] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. 2018. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*. 931–948.

[51] Jianting Zhang, Zicong Hong, Xiaoyu Qiu, Yufeng Zhan, Song Guo, and Wuhui Chen. 2020. Skychain: A deep reinforcement learning-empowered dynamic blockchain sharding system. In *Proceedings of the 49th International Conference on Parallel Processing*. 1–11.

[52] Lin Zhao, Sourav Sen Gupta, Arijit Khan, and Robby Luo. 2021. Temporal analysis of the entire ethereum blockchain network. In *Proceedings of the Web Conference 2021*. 2258–2269.

[53] Peilin Zheng, Zibin Zheng, Jiajing Wu, and Hong-Ning Dai. 2020. XBlock-ETH: Extracting and exploring blockchain data from Ethereum. *IEEE Open J. Comput. Soc.* 1 (May 2020), 95–106. https://doi.org/10.1109/OJCS.2020.2990458

A SECURITY ANALYSIS

Shard member selection. In the reconfiguration phase, all shards are reconfigured to rotate the consensus nodes within them, ensuring that the failure probability of all shards remains lower than the safety threshold. Since an adversary can accumulate in one shard to gain control over it, the failure probability of A-Shard or T-Shard in SPRING is calculated using the hypergeometric distribution [11]:

$$Pr[X \geq \lceil fn_s \rceil] = \sum_{x=\lceil fn_s \rceil}^{n_s} \frac{\binom{fn}{x}\binom{n-fn_s}{n_s-x}}{\binom{n}{n_s}},$$

where n denotes the total number of consensus nodes, n_s is the number of consensus nodes in a shard, and f is the fraction of malicious nodes, which is $\frac{1}{3}$ in this paper as A-Shard and T-Shard adopt PBFT. With a sufficient number of consensus nodes, the value of $Pr[X \geq \lceil fn_s \rceil]$ can be lowered to as low as 10^{-6}.

Safety and liveness. As T-Shard only processes transactions like a traditional blockchain with PBFT as its consensus protocol, we mainly focus on the *safety* and *liveness* of the protocol mentioned in A-Shard. The safety property guarantees malicious nodes will not compromise the blockchain, and the liveness property indicates all consensus nodes will finally reach a consensus on the proposed block. We use $\frac{1}{3}$ as the safety threshold since we assume that the number of malicious nodes is less than $\frac{1}{3}$ of the total consensus nodes in Section.3.

Theorem 1. *A-Shard achieves safety if there are less than $\frac{1}{3}$ of the nodes in the A-Shard are malicious.*

Proof. Assuming the leader is malicious and sends arbitrary state placement result A_t^* and its view of state S_t^* to other nodes. According to the *Prepare* phase, all nodes will verify the validity independently. Consequently, a malicious leader cannot compromise the A-Shard. Moreover, Assuming some consensus nodes are also malicious, they can only vote for the malicious leader, as shown in the red font in Fig.3, or refuse to vote for the honest leader. However, as the malicious nodes are less than $\frac{1}{3}$, and the block is valid only when more than $\frac{2}{3}$ of the consensus nodes vote for it, malicious nodes cannot make an invalid block valid. Overall, safety can be ensured in A-Shard.

Theorem 2. *SPRING achieves liveness if there are less than $\frac{1}{3}$ of the nodes in the A-Shard are malicious.*

Proof. Assuming a partially synchronous network, the proposed block will eventually reach the honest nodes. If a valid block is not produced due to malicious nodes crashing or misbehaving during the consensus round, the consensus round will time out and switch to the next one. The leader will be evicted from the peer connection table of other consensus nodes. Overall, liveness can be ensured in A-Shard.

B AGENT NETWORK STRUCTURE AND UPDATE ALGORITHM

We choose a feed-forward neural network with three linear layers and two ReLU [1] activation functions in the agent neural network design. The input layer contains in_D neurons. The two hidden linear layers both have a n_{neuron}-dimensional feature space, and the output layer maps the n_{neuron} neurons to out_D neurons. ReLU activation function is used between each linear layer.

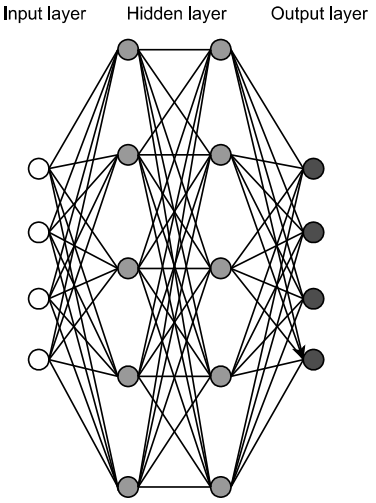


Figure 9: Schematic diagrams of agent network structures

(1) Input layer: This layer has a dimensionality of in_D, which is the state observation of the agent.
(2) Hidden layer 1: A fully connected layer, followed by a ReLU activation function.
(3) Hidden layer 2: Another fully connected layer, also followed by a ReLU activation function.
(4) Output layer: A fully connected layer with a dimensionality equal to the action space dimension (out_D) for the actor-network, or a single output neuron for the critic network.

As for deep reinforcement learning, a *policy* is a mapping from states to actions, denoted by $\pi(a|s)$, which represents the probability of taking action a in state s. It can be optimized to maximize the expected cumulative reward:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\Big[\sum_{t=0}^{T} \gamma^t R(s_t, a_t, s_{t+1}) \Big| \pi\Big].$$

Here, we adopt the PPO algorithm to optimize the address assignment policy. PPO consists of an *actor network* and a *critic network*, which are built using a neural network and are responsible for making action decisions and estimating the value of the action, respectively.

During an episode, the agent interacts with the environment using the existing policy π_θ (actor-network) to collect a batch of data. Once a complete batch of data is obtained, the actor-network and critic-network learn from the sampled data, following the algorithm

Algorithm 1 PPO Overview

Initialize policy network parameters θ and critic network parameters w.
Collect a set of trajectories $D = \tau$ by running the current policy π_θ in the environment.
for each state-action pair (s_t, a_t) in D **do**
    Calculate the target value function $V_{target}(s_t)$
    Compute the advantage function A_t
end for
for K iterations **do**
    Perform optimization on the policy network using Adam with mini-batches sampled from D, computing the gradient $\hat{g}$ by PPO surrogate objective $L(\theta)$.
    Optimize the critic network B times by minimizing the squared TD error L_{critic}. Update critic network parameters w using Adam, and compute the gradient $\hat{g}_{critic}$.
end for

1. Specifically, the actor network is optimized using surrogated Policy Gradient as follows:

$$L(\theta) = \mathbb{E}_t[\min(ratio_t A_t, clip(ratio_t, 1-\epsilon, 1+\epsilon)A_t)],$$
$$\text{ratio} = \mathbb{E}_t\left[\exp\left(\log(\pi_\theta) - \log(\pi_{\theta_{old}})\right)\right],$$
$$A_t = \delta_t + (\gamma \cdot \lambda) \cdot \delta_{t+1} + \cdots + (\gamma \cdot \lambda)^{T-t} \cdot \delta_T,$$
$$\delta_t = r_t + \gamma * V(s_{t+1}) - V(s_t),$$

where A_t is the Generalized Advantage Estimation [36], which measures the relative advantage of taking action a_t in state s_t compared to the average situation. The critic network can be optimized with Temporal Difference [40] (TD):

$$L_{critic} = \mathbb{E}t[(V(s_t) - V_{target}(s_t))^2],$$
$$V(s_t) = \mathbb{E}_{a_t}\left[r_t + \gamma * \mathbb{E}_{s_{t+1}}[V(s_{t+1})]\right].$$

By utilizing the above objective functions, the parameters of the actor network and critic network can be continuously updated over multiple episodes, leading to improved performance of the agent in the environment.

C INTRODUCTION OF THE BASELINES

Shard Scheduler [20] is a study that explores strategies for the distribution and movement of states across different shards in account-based blockchain architectures to improve scalability and performance.

Monoxide [41] introduces a novel blockchain architecture designed to enhance scalability by partitioning the network into multiple consensus zones, which operate asynchronously to achieve high throughput and reduced latency. This framework aims to address the scalability challenges of traditional blockchains by allowing zones to process transactions independently, thereby significantly increasing the system's overall capacity and efficiency.

Clustering-based dynamic sharding (CBDS) [47] focuses on blockchains in the Internet of Things (IoT). CBDS clusters IoT devices using K-means [34] for user grouping and consensus node assignment. CBDS then builds a graph representing TXs between IoT devices and dynamically adjusts sharding blockchains based on this graph as the state for RL.