IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 10, 2020, accepted May 6, 2020, date of publication May 14, 2020, date of current version May 27, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2994328

A Fast Q-Learning Based Data Storage Optimization for Low Latency in Data Center Networks

ZHUOFAN LIAO, (Member, IEEE), JINGSHENG PENG, YUANTAO CHEN, JINGYU ZHANG, (Member, IEEE), AND JIN WANG, (Senior Member, IEEE)
School of Computer and Communication Engineering, Changsha University of Science and Technology, Changsha 410114, China

Corresponding author: Jin Wang (jinwang@csust.edu.cn)

This work was supported in part by the Graduate Innovation Project of Changsha University of Science and Technology under Grant CX2018SS16, in part by the Degree and Postgraduate Education Reform Project of Hunan Province under Grant 2019JGZD057, and in part by the National Science Foundation of China under Grant 61872387 and Grant 61972055.

ABSTRACT Data storage optimizations (DS, e.g. low latency for data access) in data center networks(DCN) are difficult online-making problems. Previously, they are done with heuristics under static network models which highly rely on designers' understanding of the environment. Encouraged by recent successes in deep reinforcement learning techniques to solve intricate online assignment problems, we propose to use the Q-learning (QL) technique to train and learn from historical DS decisions, which can significantly reduce the data access delay. However, QL faces two challenges to be widely used in data centers. They are massive input data and the blindness on parameter settings which severely hamper the convergence of the learning process. To solve these two key problems, we develop an evolutionary QL scheme, named as LFDS (Low latency and Fast convergence Data Storage). In the initial stage of the LFDS, the input matrix of QL is sparse to shrink the dimensionality of the massive input data while retaining its information as much as possible. In the following training phase, a specialized neural network is adopted to achieves a quick approximation. To overcome the blindness during QL training, the two key parameters, learning rate, and discount rate are carefully tested with real data input and network architecture. The preferred range of learning rate and discount rate are recommended for the use of QL in data centers, which brings high training rewards and fast convergence. Extensive simulations with real-world data show that the data access latency is decreased by 23.5% and the convergence rate is increased by 15%.

INDEX TERMS Data center networks, data access, latency, reinforcement learning, q-learning.

I. INTRODUCTION

With the increased importance of data analysis in the cloud data center networks, more and more service providers in the world rely on data service as part of their core business that affects the performance of that system, such as Amason, Google and Microsoft [1]. But they have to battle daily with data latency: slow data access rates can reduce their ability to deliver new digital products and services, and thus harm the profitability, customer relationships, and any operational efficiency.

Selecting the right data storage configuration is critical for both performance and cost [2]. The methodology of most existing researches is reducing latency by setting up models and designing optimal algorithms [3]. However, factors that cause the delay are diverse and dynamic, such as network latency, disk latency and other types of latency(RAM, CPU, etc.) [4]. Static models can neither describe the multiple causes of delay nor be adapt to dynamics. How to take full account of the dynamic factors of data centers to optimize data storage is still an open challenge.

Since the data storage problem can be formulated as a Markov decision process (MDP) [5], and MDP problem can find an optimal action-selection policy by model-free Q-learning [6], we choose Q-learning as the basic scheme to decide the best data locations for lower latency. However, the Q-learning technique faces two challenges to be widely used in data issues for two reasons:(1) massive input data and (2)

The associate editor coordinating the review of this manuscript and approving it for publication was Joanna Kołodziej.

blindness on parameter settings, which severely hamper the convergence of the learning process. To divide and conquer these two problems, a Low latency and Fast convergence Data Storage scheme, named as LFDS is designed. The LFDS firstly sparse the input matrix of Q-learning to reduce the dimensionality of the input while retaining its information as much as possible. Then come to the training phase, a specialized neural network is adopted for Q-learning to achieves a quick approximation. To overcome the blindness on the training parameter setting, the relationship of the two key parameters, learning rate and discount rate, are carefully studied and tested with real data input and network architecture. The preferred range of learning rate and discount rate are finally carefully analyzed and recommended for the data center scenario, which brings high training rewards and fast convergence.

The proposed LFDS acts as an agent interacting with the data center environment and continuously makes actions of choosing the storage location for each data item. By collecting feedback from the environment, such as the current state of request patterns, network conditions, and the resultant end-to-end performance metrics (e.g., the read/write latency) due to these actions, the LFDS will improve the next data access location. Through this process, the agent can learn how to make a better choice for data access.

The main contributions of this paper are summarized as follows:

1) The data storage optimization problem is analyzed in a data center environment, aiming at reducing data access latency. A Q-Learning (QL) based scheme, named as LFDS, is proposed combining with neural network techniques.
2) LFDS is designed to shrink the dimensionality of the input matrix of QL under the premise that the integrity of the input information is maintained.
3) The preferred setting of the two key parameters in QL, learning rate and discount rate, are advised for the first time on the big data benchmark, which plays a decisive role in convergence.
4) Based on real data set, extensive simulations results show that LFDS can reduce the average write and read latency by 23.4% while the convergence time is improved by 15%.

The remainder of this paper is outlined as follows. The related works were concluded in Section II. Section III presents the system architecture and problem formulation. The LFDS scheme was proposed in Section IV. We evaluate the scheme in Section V. Section VI gives the conclusion.

II. RELATED WORK

Since the accessibility of Big Data is on the top priority of the knowledge discovery process, many efforts were done on improving data centers' efficiency. Two key concerns that existed are low latency and energy consumption. Researches have pointed out that well-designed data placement in data centers can highly improve the above issues by reducing data migrations, improving memory accesses to releases the network bandwidth and disk latency.

Fan *et al.* [7] tackled the problem of green data placement in data centers to strike a tradeoff among access latency, the energy consumption of data centers and network transport. The problem is proved to be NP-completeness and a 3-proximation algorithm is prosed. To meet the latency requirements of the applications and clients, Xiang *et al.* [8] provided an insightful upper bound on the average service delay of erasure-coded storage with arbitrary service time distribution and consisting of multiple heterogeneous files. Oh, *et al.* [9] presented a lightweight system called Trips to model and solve the data placement problem using mixed-integer linear programming to determine data placement. In addition, to adapt quickly to dynamics, they introduced the notion of Target Locale List, a pro-active approach to avoid expensive re-evaluation of the optimal placement. Ren *et al.* [10] modeled the joint problem of data purchasing and data placement within a cloud data market as a facility location problem which is NP-hard, and gave a divide and conquer design to get near-optimal results. Li *et al.* [11] analyzed the complexity and compared algorithms for superposed data uploading problem in networks with smart devices. Based on this, Li *et al.* [12] designed a multi-model framework for indoor localization via mobile edge computing technology. Chen *et al.* [13] proposed visual object tracking algorithm research based on adaptive combination kernel. To guarantee QoS, Chen *et al.* [14] proposed a single-image super-resolution algorithm based on structural self-similarity and deformation data block features.

However, static models can neither describe the multiple causes of delay nor be adapt to dynamics in data center networks. Because the machine learning method can approximate the optimal solution by iteratively learning the feedback from historical decisions, it is thought to be one of the best tools to solve optimal problems under environments with multiple dynamics factors. Wu *et al.* [15] gave a reinforcement learning-based data storage scheme for vehicular ad hoc networks, which can dynamically consider throughput, vehicle mobility, and bandwidth efficiency by employing a fuzzy logic algorithm. Xu *et al.* [16] proposed a reinforcement learning-based job scheduling algorithm combining with neural networks to reduce data centers' cost. To enhance the training speed, random pool sampling is proposed to retrain the neural networks via accumulated training data, and a unidirectional bridge network architecture is designed for further by using historical knowledge. Liu *et al.* [5] presented DataBot, a reinforcement learning-based adaptive model to learn the optimal data placement policies facing dynamic network conditions and time-varying request patterns. Liao *et al.* [17] considered a practical data center networks with Fat-Tree topology, and utilized a deep learning technology k-means to store most related data blocks, where k is the number of cores in the Fat-Tree. Klimovic *et al.* [2] presented a tool Selecta to recommend near-optimal

configurations of cloud compute and storage resources for data analytic workloads. An online incremental and decremental learning algorithm based on a variable support vector machine was proposed in [18].

In most of the above learning application researches, convergence is rarely considered and users are relying on default parameter settings provided by the training system. A different parameter setting, however, might yield a much higher-quality convergence. He *et al.* [19] studied parameters compressing in deep learning. Deblasio and Kececioglu *et al.* [20] have considered biological benchmarks for the first time the problem of learning the optimal set of parameter choices for a parameter advisor, who proved that learning an optimal set for an advisor is NP-complete. They implemented an approximation algorithm to find sets for advisors that are close to optimal. Considering the real-time requirements of data analysis services, convergence should require more attention and improvement. In this work, aiming at improving the learning convergence, we design a Low latency and Fast convergence Data Storage scheme by (1)shrinking the dimensionality of the input matrix of QL under the premise that the integrity of the input information is maintained, and (2)advising the two key parameters of QL, learning rate and discount rate, via analysis and tests.

III. SYSTEM ARCHITECTURE AND PROBLEM STATEMENT

A. THE DATA CENTER NETWORKS

Data center networks are comprised of three different three entities, they are master nodes, data nodes, and clients. Taking one storage system for example, it usually includes one master node who is managing the data of data (metadata), oversees the following key operations that comprise the system. Data nodes are responsible for storage and running parallel computations on that data. Clients are the applications or the load data into the cluster, submit jobs describing how that data should be processed, and then retrieves or views the results of the job when processing is finished.

The DCN has a distributed storage system consists of one master node and a set of $\mathcal{N}$ data nodes. Similar to the Hadoop Distributed File System (HDFS) [21], when a file comes, it is split into a set of data blocks $\mathcal{D}$ and these blocks are supposed to be stored in a set of data nodes. The master node executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to data nodes. The data nodes are responsible for serving read and write requests from the file system's clients. The data nodes also perform block creation, deletion, and replication upon instruction from the master node.

To WRITE in a file, the client, master and data nodes will interact with each other as the following steps:

1) The client actively requests to upload a file by communicating with the master node, and the master node checks whether the target file already exists and whether the parent directory for the target file exists.
2) The master node responds to the request of the client and returns whether it can be uploaded.
3) After receiving the response from the master node, the client will split the file into blocks and start requesting the storage location for the first block.
4) The master node recommends a list of data nodes to the client. Meanwhile, the starting time $T^W_{i,start}$ is recorded.
5) The client selects one data node from the list and requests to write in the first block. After the first data node receives the request, it will continue to call the second data node on the list, and then the second data node calls the third data node, completes the entire data node pipeline, and returns to the client step by step.
6) The client starts uploading the first block to the first data node. The first data node receives one and passes it to the second, and the second passes to the third. Usually, each block has three replicas. The third data node will send a feedback to the master node to record the complete time for the data block, such as $T^W_{i,finish}$.
7) When a block transfer is completed, the client repeat Step 1 until the target file is completely written in.

Denoting the latency of writing in the ith data block is L^W_i, then

$$L^W_i = T^W_{i,finish} - T^W_{i,start} \tag{1}$$

To READ a file from the system, a client needs to interact with the master (who stores all the metadata i.e. data about the data). Now the master checks for required privileges, if the client has sufficient privileges then the master provides the address of the data node where a file is stored. Then the client will interact directly with the respective servers to read the data blocks. The master records the time duration from receiving a request to complete reading and as L^R_i. For the data analytical function, distributed applications run on multiple data nodes and may require the transmission of data blocks among them. Because the data analytical latency is mainly related to the computation workload of data nodes and influenced by the request task priority, the optimization of analytical latency is beyond the scope of this paper.

The storage system is built on fat-tree [22], a typical network topology found in data centers cite. Generally, a fat-tree topology is typically referred to in terms of the number of pods that are numbered left to right from Pod-0 to Pod-$(k-1)$. The topology consists of k pods with three layers of switches: edge switches, aggregation switches, and core switches. Fig. 1 illustrates a distributed storage system and the Fat-Tree topology of the DCN.

B. PROBLEM STATEMENT

In the big data storage system as mentioned before, different data streams accessed by analytic workloads have distinct characteristics. Selecting the right to compute and storage data node for data analytic applications is difficult as the space of available options is large and the interactions between options are complex [2]. How to decide the optimal

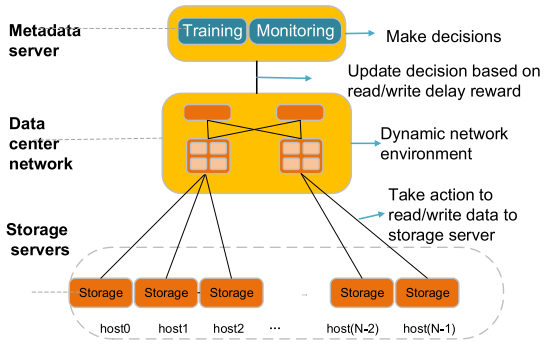


FIGURE 1. The architecture of the data storage system.

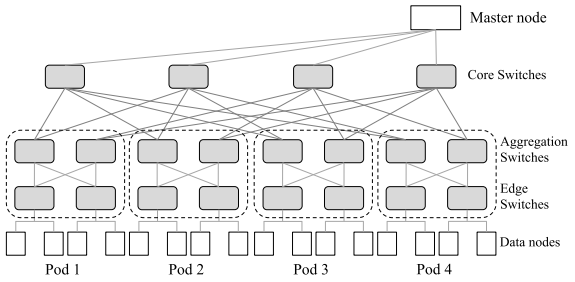


FIGURE 2. Data storage system with the Fat-Tree topology: The master node, switches and data nodes.

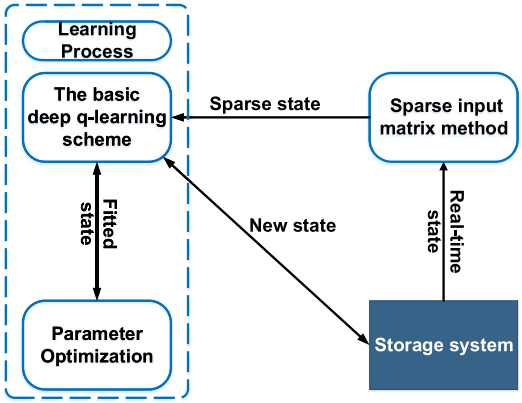


FIGURE 3. The overview of the LFDS scheme.

data access location among all available data nodes to reduce latency is critical for both performance and cost.

Since the DCN environment is complex and dynamic, the traditional static model is no more suitable for the optimization problem of DCN. The data access can be formulated as a finite Markov decision process (FMDP) as described in one of our previous works [5], for (1) the amount of candidate data access locations(data nodes) are finite, and (2) each data access decision depends only upon the present state of the DCN, not on the sequence of events that preceded it, which is called as the Markov property. Because for any given FMDP, given infinite exploration time and a partly-random policy, Q-learning can identify an optimal action-selection policy [23] and it is a model-free reinforcement learning algorithm, **Q-learning** is chosen as the solution to learn dynamically from the historical DCN data access and apply improved data access decision.

Although DQL can provide us with the optimal solution through the finite Markov decision process, that is, when the request arrives, it can determine the optimal storage location for us. However, imagine that in the era of big data, with the exponential growth of data, facing the dynamic allocation of massive data and the demand of low latency, if all these data are used as input of neural network learning, it will result in huge input and training space and a low convergence of the training process. In another word, the input of massive data hinders the advantage of Q-learning. How to reduce the training input while retaining information becomes the key issue.

IV. DESIGN OF THE LOW LATENCY AND FAST CONVERGENCE DATA ACCESS SCHEME (LFDS)

LFDS is composed of two parts: (1) the basic Deep Q-Learning scheme (DQL) for dealing with the dynamic environment and data access patterns, (2) the Sparse input matrix method to further reduce the input state scale of DQL. Fig. 3 give the overview of the LFDS scheme, followed by design details of each part.

A. THE BASIC DEEP Q-LEARNING SCHEME

Q-learning is used on the master node acting as an agent interacting with the data storage system. This agent continuously makes actions of choosing the write/read location for each data block and collects the feedback from the environment, including the current state of request patterns and network conditions, and the resultant end-to-end performance metrics (e.g. the read/write latency) due to these actions.

1) DESIGN OF Q-LEARNING

Similar to [5], the fundamental design of Q-learning consists of three sets: states $\mathcal{S}$, actions $\mathcal{A}$ and reward Q-function. Each **State** of the storage system will changed according to each **Action**, and brings different **Reward**, which can be expressed as: $\mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$.

States: According to the characteristics of the big data center, we divide the state into three categories, in which the state information comes from the data read/write request log.

- Network conditions include the average latency of a read request from the source node i to the destination node j which is denoted as L_{ij}^{R}, and the average latency of write requests which is L_{ij}^{W}, where $i, j \in N$. The average delay is measured in a real-time network state information log.
- Request frequency include [24]: 1) Read rate or frequency of data block m from source server i, denoted by $F_{i,m}^{[R]}$; 2) Write rate of data block m to source server i, denoted by $F_{i,m}^{[W]}$; 3) Read rate of all data block from source server i, denoted by $\hat{F}_{i}^{[R]}$; 4) Write rate of all data block to source server i, denoted by $\hat{F}_{i}^{[W]}$.

Action: We use a to represent the destination node (storage node) of data item writing. N is the number of storage servers,

where $a \in N$. We use an array to represent the action set. When the action is taken (i.e. after the target node is successfully stored), set the index value of the corresponding array to 1 (the index of the array is the number of the current storage node), and set all other indexes to 0. In the dynamic data center network environment, the actions taken by the master node agent at every moment may affect the load balance of the data center.

Reward: The big data center system needs to evaluate every data location update (action), that is, the reward in reinforcement learning. The goal of data center network optimization is to obtain low latency data deployment by maximizing rewards. The reward is defined as the reciprocal of the weighted sum of read/write delays for data item movement at time $[t, t')$, l_w refers to the write latency, l_k^R is the read latency of the k th round of training [24]. t is the time when the data item m to be written, and t' is the time of next write operation to m. In this period, although the data is only written once, there may be multiple read operations, it is necessary to calculate the delay of all read operations during this period. Then average delay is seen as a reward for this time. The calculation method of reward is shown in Formula 2.

$$Reward = \frac{1}{l^w} + \frac{1}{|k|}\sum_{k\in K}\frac{1}{l_k^R} \tag{2}$$

2) OPTIMAL VALUE FUNCTION

The Markov decision process (MDP) indicates that the next time state of the system is only the current time state, which is independent of the historical state:

$$\mathrm{P}\left(S_{t+1}|S_t, S_{t-1}, \cdots, S_1, S_0\right) \tag{3}$$

We use (S, A, P) to represent MDP, that is, S to represent the state, A to action and P to the probability of state transition which means the probability of state S_t transferred to S_{t+1} by action A_t at time t. We define the state set $S = \{s_1, s_2, \cdots, s_n\}$, Action set $A = \{node_1, node_2, \cdots, node_n\}$. Our goal is to obtain a low latency data deployment strategy by using a reinforcement learning algorithm. The strategy here represents the mapping from the state to the action, which is given by the conditional probability distribution π, that is, the distribution of the action set in the known state s:

$$\pi(a|s) = P\left[A_t = a|S_t = s\right] \tag{4}$$

In Formula 4, strategy π specifies an action probability in each state s. When strategy π has been solved, we can use strategy π to figure out what action to take in any state s. When we adopt the strategy π according to the current state s and action a, the system will interact with the environment according to the current strategy and get rewards. Reinforcement learning ultimately seeks the optimal strategy, which is measured by the cumulative return from environmental feedback. The greater the cumulative return, the closer to the optimal solution. When the master node agent adopts the policy π, we can calculate the cumulative return. Cumulative return is defined as:

$$\mathrm{G}_t = R_{t+1} + \gamma R_{t+2} + \gamma R_{t+3} \cdots = \sum_{k=0}^{\infty}\gamma^k R_{t+k+1} \tag{5}$$

where R is the reward, k is the round of training, γ is discount factor, which is generally less than 1. Usually, the present reward is more important.

When the master node agent uses the policy π, the expected value of the cumulative reward under the state s is defined as the state-value function:

$$V_\pi(s) = E_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t = s\right] \tag{6}$$

Accordingly, the state-behavior value function is:

$$Q_{t+1}(s,a) = (1-\alpha_t(s,a))Q_t(s,a) + \alpha_t(s,a)(R_M(s,a) + \gamma \max_{b\in U(s'}Q_t(s',b)) \tag{7}$$

Finding the optimal strategy is equivalent to solve the optimal value function:

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) \tag{8}$$

The updating formula of value function is as follows:

$$\begin{aligned}&\forall s, a : Q_0(s,a) = C\\ &\forall s, a : Q_{t+1}(s,a) = (1-\alpha_t^\omega(s,a))Q_t(s,a)\\ &\qquad + \alpha_t^\omega(s,a)(R_M(s,a) + \gamma \max_{b\in U(\bar{s})}Q_t(\bar{s},b))\end{aligned} \tag{9}$$

3) POLICY DESIGN

The master node agent explores the dynamic network environment through the ε-greedy strategy. In other words, the probability of ε is used to select random actions, and the probability of 1-ε is used to calculate and make an optimal action according to the current Q value, Where ε is a number greater than 0 and less than 1. The mathematical expression of ε-greedy strategy is:

$$\pi(\mathrm{a|s}) \leftarrow \begin{cases} 1-\varepsilon+\dfrac{\varepsilon}{|A(\varepsilon)|} & \text{if } a = \arg\max_a Q(s,a) \\ \dfrac{\varepsilon}{|A(s)|} & \text{if } a \neq \arg\max_a Q(s,a) \end{cases} \tag{10}$$

4) DEEP Q-NETWORK

Q-Learning can solve the Markov decision problem in low dimensional state space, but in the real data center network, every microsecond has to deal with huge data. It is not rational to use the Q-Table to store state-action pairs. In this paper, deep neural uses formula 6 to fit the state-value function [5]. Every time a certain number of samples are collected and the Q-function is updated, that is to say, the parameters of the neural network are constantly updated, to learn the optimal action. Fig. 4 shows the deep neural network structure.

The network delay matrix x_1 is the average delay of the end i sending read operation request to the destination j. if we use N to represent the number of servers in the data center,

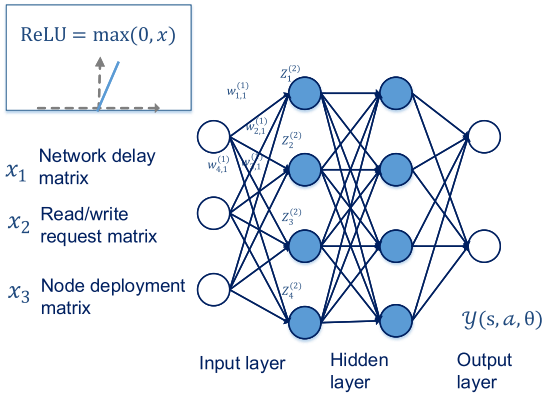


FIGURE 4. Deep neural network structure.

then the network delay matrix size is $N \times N$. The read-write request matrix x_2 is the frequency of each data item requesting operation from the original end. If we use M to represent the number of data items, the size of the read/write request matrix is $M \times N$. Because there is only one source side of the current data block request and there are N servers in total, the size of Node deployment matrix is $1 \times N$.

$(w_{i,j}, b)$ is used to represent parameters in neural networks. $w_{i,j}^{(l)}$ represents the weight of the connection between i unit of l layer and j unit of $l+1$ layer. $b_i^{(l)}$ is the deviation parameter. $z_i^{(l)}$ is the weighted sum of i unit input in l layer, which can be calculated by Formula 11.

$$z_i^l = \sum_{j=l}^{nl} w_{ij}^{(l)} x_j + b_i^{(l)} \tag{11}$$

When training neural network, there is a functional relationship between the output of the L layer node and the input of the $L+1$ layer node, such as Formula 12, which is called activation function.

$$a_i^{(l)} = f\left(z_i^{(l)}\right) \tag{12}$$

In this paper, We use the Rectified Linear Unit (Relu) as the excitation function, which is a piecewise linear function. When the input parameter p is greater than zero, the output is equal to the input and when p is less than zero, the output is zero. Compared with sigmoid, Relu tends to converge more easily in multi-layer deep neural network training.

After training, the weights in the neural network will be updated. It uses back propagation (BP) to update the weights in the network. Gradient descent is a very common method to find the local minimum value. Constructing square loss function for for a single sample $(x^{(i)}, y^{(i)})$ in a training set $T = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})\}$:

$$J\left(W, b, x^{(i)}, y^{(i)}\right) = \frac{1}{2}\left\|h_{w,b}\left(x^{(i)}\right) - y^{(i)}\right\|^2 \tag{13}$$

According to the loss function, the gradient descent method is used to update $W_{i,j}^{(l)}$ and $b_i^{(l)}$:

$$W_{t,j}^{(l)} = W_{i,j}^{(l)} - \alpha \frac{\partial}{\partial W_i^{(l)}} J(w, b) \tag{14}$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(w, b) \tag{15}$$

Every iteration of gradient descent calculates all samples, which will affect the speed of convergence. So in this paper, we use stochastic gradient descent (SGD) to randomly select a group of samples from each iteration. In the case of large sample data, we can get an acceptable loss value without training all the samples.

When the system is running, clients continuously send read/write requests to the master node agent. By using the state matrix s of the current time t as the input of the neural network, the agent obtains the action a of the time t through the neural network or the greedy search strategy. Then the agent takes the action a in the data center network environment of the time $t+1$ to obtain the data center network state and the reward of the time $t+1$. We can see that the whole data center environment is dynamic. After continuous attempts, the master node will select 128[1] sample data from the sample pool as training data, and constantly update the parameters in the neural network. In the end, the master node can take the optimal node deployment location according to the current load status of the data center network.

B. SPARSE INPUT MATRIX METHOD

By analyzing the features of real data, we observed that only a few data appeared frequently and Zipf's law [25] is satisfied. As shown in Fig. 5, In other words, we do not need to consider all the read/write request and only a small part of them covers the read/write request law. Sparsity refers to retraining the main feature information in the input as much as possible and eliminating the secondary feature parameters. In the training process, as many feature parameters as possible are expected to be zero, so that the practical and effective information can be concentrated in a low-dimensional space.

In our system, we assume that there are N data nodes and M data blocks. $f_{m,n}$ denote the request frequency of data block m by the node n. The Data-node matrix is recorded as:

$$A = \begin{pmatrix} f_{1,1} & \cdots & f_{1,n} \\ \vdots & \ddots & \vdots \\ f_{m,1} & \cdots & f_{m,n} \end{pmatrix} \tag{16}$$

The maximum frequency value of row i of A denoted as f_{max}. If the variance between $f_{i,j}$ and f_{max} is less than a certain value d, then we call this data block i as **Active Data** in nodes j. Otherwise, we call it **Inactive Data** and set $f_{(i,j)} = 0$.

$$\text{Active Data} \leftarrow \left(f_{max} - f_{i,j}\right)^2 < d, \quad (d > 0) \tag{17}$$

$$\text{Inactive Data} \leftarrow \left(f_{max} - f_{i,j}\right)^2 > d, \quad (d > 0) \tag{18}$$

[1]Generally, the selection range is between (0, 2^{32}], and then 128 is the most used value for small batches in the q-learning algorithm

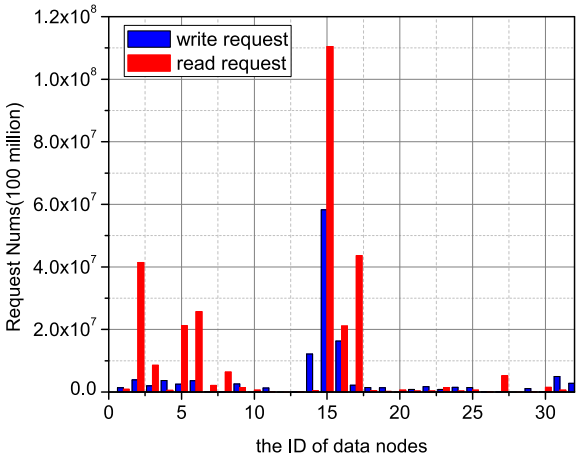


FIGURE 5. The distribution of requests among 30 data nodes.

We use the data blocks with high request frequency as the input of our neural network, so neural network does not need to train all the data. By preprocessing the data frequency request in advance and reducing the deep Q-learning status input, the following matrix B is obtained. Matrix B is formed by sparseness of matrix A, where $k \ll m$.

$$B = \begin{pmatrix} f_{1,1} & \cdots & f_{1,n} \\ \vdots & \ddots & \vdots \\ f_{k,1} & \cdots & f_{k,n} \end{pmatrix} \tag{19}$$

V. PERFORMANCE EVALUATION

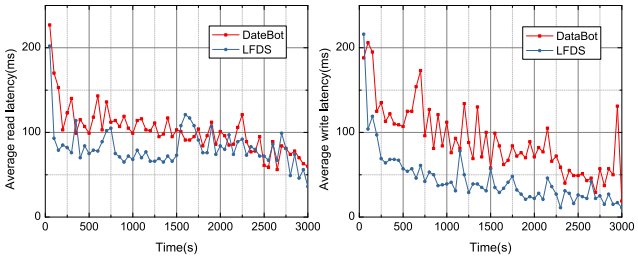
Based on the real trace data, Microsoft Research Cambridge Trace [26], a series of simulations are carried out in this section. The experimental environment of this paper is based on the Ubuntu 16.04 operating system. The hardware configuration is equipped with Intel @ Xeon (R) CPU e5-2697 V2 processor, 8G memory and 512GB hard disk. The required software includes mininet v2.2.0, openflow 1.3, floodlight v1.2 and memcached v1.59.

Through simulations, the performance of the proposed LFDS in improving read/write latency is compared with related benchmarks, and the convergence of each algorithm is compared. Finally, the impact of the learning rate and discount rate setting on the performance of the algorithm is tested. Benchmark works used in this paper include:

1) DateBot [24]: the basic Q-learning scheme for data storage using the original state space matrix.
2) CommonIP [27]: metaserver selects the node closest to the current copy to place data.
3) DSBK [17]: data deployment strategy based on k-means which is used to cluster the data.

Mininet [28] is used to simulate the data center network in this paper. Mininet has the advantages of fast start-up speed, large scalability, multiple bandwidths, easy installation and use, which is very suitable for our simulation environment construction. We have built a three-layer switch network topology of Fat-Tree, and we set the direct link bandwidth

FIGURE 6. The simulated topology of the fattree.



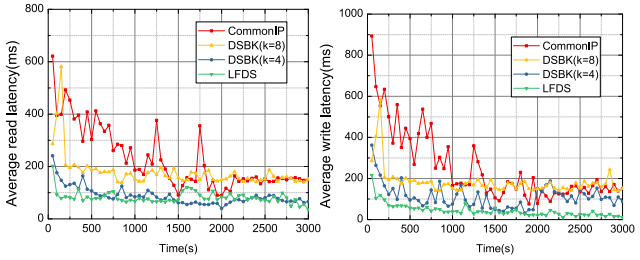
(a) Read (b) Write

FIGURE 7. The impact of sparse input matrix method on read/write latency.

of each switch to 1Gbps [7]. A Fat-Tree, as shown in Fig. 6, is set to 4 pods, so there are 32 hosts in the network. Each host is equipped with Memcache. Each storage host has a client as the request source and a memcached process as the request-target. Memcache is a memory-based cache system, which supports the storage of key-value pairs. It has excellent data reading and writing performance and distributed expansion capability. Similar to HDFS, we adopt a 3-copy data backup strategy.

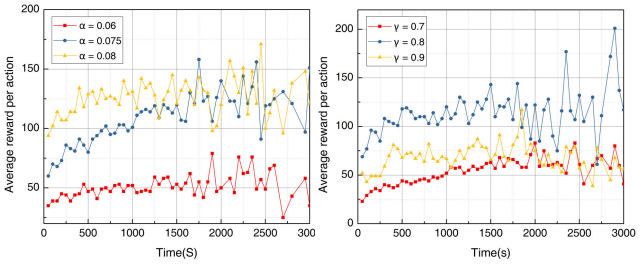
A. READ/WRITE LATENCY PERFORMANCE

We explored the impact of the sparse input matrix method on DQL. We have carried out two groups of experiments between the LFDS and the Datebot. From Fig. 7(a), we can see that during the first 1500s of system operation, when the request operation sent by the client is read, the average read/write latency is lower than the Datebot. After the 1500s, the convergence of the two algorithms is the same. In general, the sparse input matrix method reduces the average latency of reading operation by 39.3 ms. On the other hand, from Fig. 7(b), the sparse input matrix method reduces the average latency of write operation by 45.67ms. We can explain that for the data center, the sparse DQL state-space matrix is helpful to improve network utilization and reduce the internal



(a) Read (b) Write

FIGURE 8. Latency of all deployment strategy.



(a) Learning rate: α (b) Discount rate: γ

FIGURE 9. The impact of learning rate and discount rate on reward.

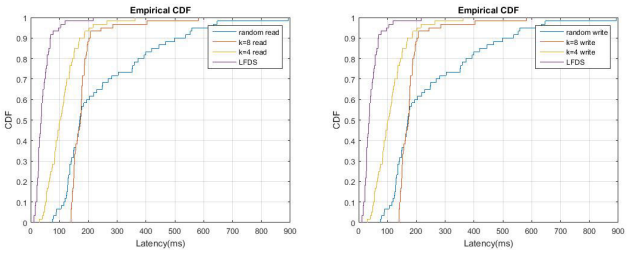
backbone network load. It is more helpful for its master node to adopt the optimal server node deployment to achieve lower read-write latency.

Finally, we compared the read/write latency performance of all deployment strategy. It can be seen from Fig. 8 that during the 3000s running of our system, the average read-write delay of DQL is 81.7ms and 43.98ms respectively, which is 2.4% and 28.4% lower than DSBK (k = 4), and 63.9% and 82.4% lower than CommonIP respectively. We can conclude that in the same experimental environment, DQL and DSBK have lower average read/write latency than CommonIP. In the aspect of data write operation, DQL reduces the latency by 28.4% compared with DSBK.

B. IMPACT OF LEARNING RATE AND DISCOUNT RATE ON CONVERGENCE

We test the impact of learning rate α and discount rate γ on reward in Q-learning. The learning rate refers to how much difference between each iteration will be learned, and the discount rate is the attenuation value of the future reward. To control the variables, we preset the learning rate and a discount rate as 0.075 and 0.7 respectively.

As shown in Fig. 9(a) when the learning rate is 0.06, the reward value obtained by each action (storing data) taken by the master node agent fluctuates between 40 and 80, and the change is not very large over time but rebounds around the 2600s. When the learning rate is 0.08 and 0.075 within the 1500s of system operation, there is a significant difference in reward: $ARA^{\alpha}_{0.08} > ARA^{\alpha}_{0.075} > ARA^{\alpha}_{0.06}$, and AVA is reward per action. When the system runs for the 1500s, $ARA^{\alpha}_{0.08}$ and $ARA^{\alpha}_{0.075}$ are the same. While they all grow faster than $ARA^{\alpha}_{0.06}$.



(a) CDF of read latency (b) CDF of write latency

FIGURE 10. The Cumulative Distribution Function of read/write latency.

According to Fig. 9(b), during the operation of the system, the master node continuously deploys and stores the data, and obtains the corresponding reward from the network environment, to optimize the deployment strategy of the master node. By changing the attenuation value of the reward, we can find the optimal value for our scheme. During the system operation time, $ARA^{\gamma}_{0.8} > ARA^{\gamma}_{0.9} > ARA^{\gamma}_{0.7}$. **To sum up, when the learning rate is 0.08 and the discount factor is 0.8, the reward for the master node to take action will be greater**.

C. CONVERGENCE COMPARISON

The faster the algorithm converges, the faster the data deployment scheme can be given. By comparing the cumulative distribution function of read/write latency of each scheme, we can see the convergence of the scheme. In another word, the fast the CDF curve convergent, the better the scheme. Given a simple example, in the Fig. 10(a), it is shown that when the read latency of LFDS's is 100ms, its CDF is about 0.95, which means that the distribution probability is 95% when the read latency is less than 100ms. And the CDF value of LFDS is already 1 at about 200ms, while that of the CommonIP reaches 1 at about 800ms. This means that LFDS has a faster convergence than CommonIP and can provide a data deployment scheme faster than the CommonIP.

By analyzing the CDF value of CommonIP, DSBK (k = 8), DSBK (k = 4) and LFDS, it is found that when the CDF value of CommonIP is 0.6, the read-write delay is 200ms. When the CDF of DSBK(k = 8) is 0.6, the read/write latency is 174.2ms (as shown in Fig. 10(a)) and 177.7ms (as shown in Fig. 10(b)). When the CDF of DSBK(k = 4) is 0.6, its read/write latency is 85.4ms and 112.2ms. The cumulative distribution probability of LFDS is 0.6, and the delay is 81.4ms, 42.6ms. With the sparse input matrix and the dedicated set learning/discount rate, LFDS outperforms its counterparts on convergence, which means high efficiency on data storage.

VI. CONCLUSION

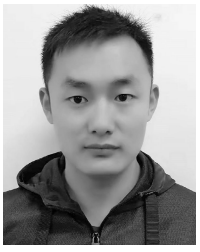
This paper studied the data center storage method for reducing read/write latency in the data center. An evolutionary Q-learning scheme, named as LFDS (Low latency and Fast convergence Data Storage), was proposed. Reinforcement learning was used to obtain the optimal assignment and the neural network to fit the input data. Furthermore, the input

matrix of Q-learning was sparse to shrink the dimensionality of the massive input data. The simulation-based on real data shows that LFDS can effectively reduce the read/write latency of DCN.

REFERENCES

[1] M. U. Bokhari, Q. Shallal, and Y. K. Tamandani, "Cloud computing service models: A comparative study," in *Proc. Int. Conf. Comput. Sustain. Global Develop.*, Mar. 2019, pp. 890–895.
[2] A. Klimovic, H. Litz, and C. Kozyrakis, "Selecta: Heterogeneous cloud storage configuration for data analytics," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2018, pp. 759–773. [Online]. Available: https://www.semanticscholar.org/paper/Selecta%3A-Heterogeneous-Cloud-Storage-Configuration-Klimovic-Litz/12482296ddb3ec88cd91f3b03ad74fd60e438bcb and https://www.usenix.org/conference/atc18/presentation/klimovic-selecta
[3] Y. Hirashima, Y. Iiguni, A. Inoue, and S. Masuda, "Q-learning algorithm using an adaptive-sized Q-table," in *Proc. 38th IEEE Conf. Decis. Control*, vol. 2, Dec. 1999, pp. 1599–1604. [Online]. Available: http://ieeexplore.ieee.org/document/830250/
[4] M. Alizadeh, A. Kabbani, T. Edsall, A. Vahdat, and M. Yasuda, and B. Prabhakar, "Less is more: Trading a little bandwidth for ultra-low latency in the data center," in *Proc. 9th USENIX Conf. Netw. Syst. Design Implement.*, 2012, pp. 253–256.
[5] K. Liu, J. Peng, J. Wang, B. Yu, Z. Liao, Z. Huang, and J. Pan, "A learning-based data placement framework for low latency in data center networks," *IEEE Trans. Cloud Comput.*, early access, Sep. 12, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8834843/, doi: 10.1109/TCC.2019.2940953.
[6] A. Al-Tamimi, F. L. Lewis, and M. Abu-Khalaf, "Model-free Q-learning designs for linear discrete-time zero-sum games with application to H-infinity control," *Automatica*, vol. 43, no. 3, pp. 473–481, Mar. 2007.
[7] Y. Fan, H. Ding, L. Wang, and X. Yuan, "Green latency-aware data placement in data centers," *Comput. Netw.*, vol. 110, pp. 46–57, Dec. 2016, doi: 10.1016/j.comnet.2016.09.015.
[8] Y. Xiang, T. Lan, V. Aggarwal, and Y.-F.-R. Chen, "Joint latency and cost optimization for erasure-coded data center storage," *IEEE/ACM Trans. Netw.*, vol. 24, no. 4, pp. 2443–2457, Aug. 2016.
[9] K. Oh, A. Chandra, and J. Weissman, "TripS: Automated multi-tiered data placement in a geo-distributed cloud environment," in *Proc. 10th ACM Int. Syst. Storage Conf.*, May 2017, pp. 1–11. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3078468.3078485
[10] X. Ren, P. London, J. Ziani, and A. Wierman, "Datum: Managing data purchasing and data placement in a geo-distributed data market," *IEEE/ACM Trans. Netw.*, vol. 26, no. 2, pp. 893–905, Apr. 2018.
[11] W. Li, H. Liu, J. Wang, L. Xiang, and Y. Yang, "An improved linear kernel for complementary maximal strip recovery: Simpler and smaller," *Theor. Comput. Sci.*, vol. 786, pp. 55–66, Sep. 2019.
[12] W. Li, Z. Chen, X. Gao, W. Liu, and J. Wang, "Multimodel framework for indoor localization under mobile edge computing environment," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4844–4853, Jun. 2019.
[13] Y. Chen, J. Wang, R. Xia, Q. Zhang, Z. Cao, and K. Yang, "The visual object tracking algorithm research based on adaptive combination kernel," *J. Ambient Intell. Humanized Comput.*, vol. 10, no. 12, pp. 4855–4867, Dec. 2019.
[14] Y. Chen, J. Wang, X. Chen, M. Zhu, K. Yang, Z. Wang, and R. Xia, "Single-image super-resolution algorithm based on structural self-similarity and deformation block features," *IEEE Access*, vol. 7, pp. 58791–58801, 2019.
[15] C. Wu, T. Yoshinaga, Y. Ji, T. Murase, and Y. Zhang, "A reinforcement learning-based data storage scheme for vehicular ad hoc networks," *IEEE Trans. Veh. Technol.*, vol. 66, no. 7, pp. 6336–6348, Jul. 2017.
[16] C. Xu, K. Wang, P. Li, R. Xia, S. Guo, and M. Guo, "Renewable energy-aware big data analytics in geo-distributed data centers with reinforcement learning," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 1, pp. 205–215, Jan. 2020.
[17] Z. Liao, R. Zhang, S. He, D. Zeng, J. Wang, and H.-J. Kim, "Deep learning-based data storage for low latency in data center networks," *IEEE Access*, vol. 7, pp. 26411–26417, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8653268/
[18] Y. Chen, J. Xiong, W. Xu, and J. Zuo, "A novel online incremental and decremental learning algorithm based on variable support vector machine," *Cluster Comput.*, vol. 22, no. S3, pp. 7435–7445, May 2019.
[19] S. He, Z. Li, Y. Tang, Z. Liao, F. Li, and S.-J. Lim, "Parameters compressing in deep learning," *Comput., Mater. Continua*, vol. 62, no. 1, pp. 321–336, 2020.
[20] D. DeBlasio and J. Kececioglu, "Learning parameter-advising sets for multiple sequence alignment," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 14, no. 5, pp. 1028–1041, Sep. 2017. [Online]. Available: https://ieeexplore.ieee.org/document/7102700/
[21] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol. (MSST)*, May 2010, pp. 1–10.
[22] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE Trans. Comput.*, vol. C-34, no. 10, pp. 892–901, Oct. 1985.
[23] N. Kantasewi, S. Marukatat, S. Thainimit, and O. Manabu, "Multi Q-table Q-learning," in *Proc. 10th Int. Conf. Inf. Commun. Technol. Embedded Syst. (IC-ICTES)*, Mar. 2019, pp. 1–7. [Online]. Available: https://ieeexplore.ieee.org/document/8695963/
[24] K. Liu, J. Wang, Z. Liao, B. Yu, and J. Pan, "Learning-based adaptive data placement for low latency in data center networks," in *Proc. IEEE 43rd Conf. Local Comput. Netw. (LCN)*, Oct. 2018, pp. 142–149. [Online]. Available: https://ieeexplore.ieee.org/document/8638050/
[25] W. Li, "Random texts exhibit Zipf's-law-like word frequency distribution," *IEEE Trans. Inf. Theory*, vol. 38, no. 6, pp. 1842–1845, Nov. 1992.
[26] SNIA Lotta Respository. *Q-Learning Algorithm Using an Adaptive-Sized Q-Tablemsr Cambridge Traces*. [Online]. Available: http://iotta.cs.hmc.edu/traces
[27] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan, "Volley: Automated data placement for geo-distributed cloud services," in *Proc. 7th USENIX Conf. Netw. Syst. Design Implement.* Berkeley, CA, USA: USENIX Association, 2010, pp. 1–16.
[28] R. C. Chen, P. G. Jessel, and R. B. Patterson, "MININET: A microprocessor-controlled 'Mininetwork,"' *Proc. IEEE*, vol. 64, no. 6, pp. 988–993, Jun. 1976.

ZHUOFAN LIAO (Member, IEEE) received the Ph.D. degree in computer science from Central South University, China, in 2012. From 2017 to 2018, she worked as a Visiting Scholar with The University of Victoria, Canada, supported by the China Scholarship Council. She is currently an Assistant Professor with the School of Computer and Communication Engineering, Changsha University of Science and Technology, China. Her research interests include wireless networks optimization, big data, and edge computing for 5G. She has published papers in leading transactions and conferences as the first author in the above areas. She has served as a Reviewer for the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS and the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY.



JINGSHENG PENG is currently pursuing the master's degree with the Changsha University of Science and Technology. His research interests include mobile edge computing and big data placement optimization. He is good at C/C++ programming under Linux, and also familiar with Python.

YUANTAO CHEN received the B.S. degree in computer science and technology from the Jianghan Petroleum Institute, the M.S. degree in geodetection and information technology from Yangtze University, and the Ph.D. degree in control science and engineering from the Nanjing University of Science and Technology, in 2014. He is currently an Associate Professor with the Changsha University of Science and Technology. His research interests include pattern recognition and image processing.

JINGYU ZHANG (Member, IEEE) received the B.E. degree in communication engineering from Hunan Normal University, in 2008, the M.E. degree in computer applications from Chongqing Jiaotong University, in 2010, and the Ph.D. degree in computer science and technology from Shanghai Jiao Tong University, in 2017. He was a Visiting Ph.D. Student with Ohio State University, from 2014 to 2016. He is currently an Assistant Professor with the School of Computer and Communication Engineering, Changsha University of Science and Technology, China. His main research interests include high-performance architecture and big data performance optimization, blockchain performance, and consensus mechanism optimization. He is a member of the China Computer Federation.

JIN WANG (Senior Member, IEEE) received the M.S. degree from the Nanjing University of Posts and Telecommunications, China, in 2005, and the Ph.D. degree from Kyung Hee University, South Korea, in 2010. He is currently a Professor with the Changsha University of Science and technology. He has published more than 300 international journals and conference papers. His research interests mainly include wireless ad hoc and sensor networks, network performance analysis, and optimization. He is a member of ACM.