Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

MDLB: a metadata dynamic load balancing mechanism based on reinforcement learning*

Zhao-qi WU[1], Jin WEI[2,3], Fan ZHANG[†‡1], Wei GUO[1], Guang-wei XIE[2,3]

[1]*National Digital Switching System Engineering & Technological R&D Center, Zhengzhou 450002, China*

[2]*School of Computer Science, Fudan University, Shanghai 200433, China*

[3]*Data Arena Institute, Fudan University, Shanghai 200433, China*

[†]E-mail: 17034203@qq.com

Received Mar. 1, 2019; Revision accepted Nov. 14, 2019; Crosschecked June 2, 2020

Abstract: With the growing amount of information and data, object-oriented storage systems have been widely used in many applications, including the Google File System, Amazon S3, Hadoop Distributed File System, and Ceph, in which load balancing of metadata plays an important role in improving the input/output performance of the entire system. Unbalanced load on the metadata server leads to a serious bottleneck problem for system performance. However, most existing metadata load balancing strategies, which are based on subtree segmentation or hashing, lack good dynamics and adaptability. In this study, we propose a metadata dynamic load balancing (MDLB) mechanism based on reinforcement learning (RL). We learn that the Q_learning algorithm and our RL-based strategy consist of three modules, i.e., the policy selection network, load balancing network, and parameter update network. Experimental results show that the proposed MDLB algorithm can adjust the load dynamically according to the performance of the metadata servers, and that it has good adaptability in the case of sudden change of data volume.

Key words: Object-oriented storage system; Metadata; Dynamic load balancing; Reinforcement learning; Q_learning
https://doi.org/10.1631/FITEE.1900121 **CLC number:** TP338.8

1 Introduction

The increasing amount of data provides huge challenge to data storage systems, especially with the advent of big data era in the fields of engineering and information service. With the introduction of a series of storage systems such as the Google File System (GFS) (Ghemawat et al., 2003), Amazon S3 (Palankar et al., 2008), Hadoop Distributed File System (HDFS) (Shvachko et al., 2010), and Ceph (Wang FY et al., 2013), distributed storage technology has developed rapidly and become a supporting technology of storage at the stage of cloud computing and big data. In the current distributed file system, metadata of files is separated from data access. Metadata is stored on the metadata server (MDS), and data is stored on an object-based storage device (OSD) (Chen et al., 2013). Users can read data by communicating with the MDS on a network (Fig. 1). However, imbalance of the MDSs may cause a bottleneck problem for system performance. Existing load balancing algorithms focus mainly on two methods, i.e., subtree segmentation (Patgiri et al., 2017) and hashing (Wang JD et al., 2018).

The subtree segmentation method (Patgiri et al., 2017) assigns the directory hierarchy of a file to different server nodes statically or dynamically. Thus, the method is divided mainly into static segmentation (Patgiri et al., 2017) and dynamic segmentation (Xiong et al., 2005). Systems such as the network file system (NFS) (Hitz et al., 1994), Andrew File System

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (Nos. 61572520 and 61521003)

ORCID: Zhao-qi WU, https://orcid.org/0000-0001-7857-2875; Fan ZHANG, https://orcid.org/0000-0001-7456-82377

(AFS) (Howard et al., 1987), Coda (Satyanarayanan et al., 1990), and Sprite (Zhong et al., 2003) all use this method to manage metadata.

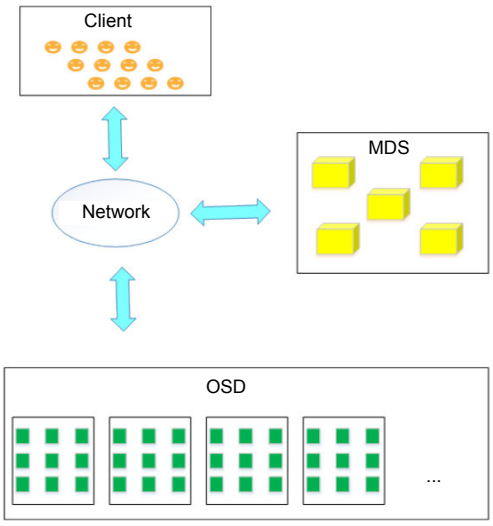

Fig. 1 Architecture of the metadata server (MDS)
Metadata is stored on the MDS and data is stored on the object-based storage device (OSD). Users can communicate with MDS and OSD over the network to read and write data

The hashing method allocates metadata among servers based on hash values. The method is divided into static hashing (Lewis and Cook, 1988), lazy hybrid (Manindra and Thiagarajan, 2004), directory path hashing (Liu and Zhou, 2007), and dynamic hashing (Li et al., 2006). Examples of algorithms are the basic load balancing algorithm (BBLA) (Schindelhauer and Schomaker, 2005) and adaptable distributed load balancing of metadata (ADMLB) (Chen et al., 2013).

However, these existing methods have disadvantages. For example, the subtree segmentation method retains the file hierarchy, but cannot effectively handle the request-intensive directories, and it does not consider the dynamic changes of metadata load. Static hashing does not support server size changes or distribute load according to server performance. The lazy hybrid method has been improved based on static hashing. However, it does not support the heterogeneity of servers, which means that the change of the server size will cause the change of the hash value. The directory path hashing method divides the full file path name into two parts, i.e., directory path and file name. The directory path servers manage the metadata, and the MDSs manage the file metadata. The single directory path server is likely to cause a bottleneck problem for performance, and the hash calculation of the file name cannot guarantee dynamic load balancing. The dynamic hashing method uses the hash value of the full file path name to allocate metadata and elastically supports the addition, deletion, and replacement of the server. However, it cannot dynamically allocate the metadata load according to the performance of the MDS. So, it is a challenging problem due to the following reasons:

1. Incompleteness. Since the existing algorithms use only metadata access delay to represent the MDS load, it is a challenge to consider other delay attributes of the MDS itself.

2. Lack of dynamics. The current algorithms based on dynamic hashing consider the load change with time, but the training parameters in the distance function are fixed, which makes the algorithms unable to dynamically adjust the server load and causes the algorithms to lack dynamics.

Because of the problems of load balancing algorithms, we propose a metadata dynamic load balancing (MDLB) method based on reinforcement learning (RL), and improve the BBLA and ADMLB algorithms. Based on the logarithmic method, the metadata is assigned to the nearest MDS according to the distance function between the metadata and MDSs. The distance function takes full consideration of the central processing unit (CPU) computing performance, memory performance, input/output (I/O) performance, disk size, and other indicators. Furthermore, when training the distance function parameters, we introduce the RL framework innovatively. The framework is divided into three modules, i.e., the strategy selection network, metadata load balancing network, and parameter update network. With the help of the RL method, we can choose the most appropriate metadata distribution. Since the training parameters in the parameter update network change as the MDS load changes, the metadata can be optimally allocated to the most reasonable MDS. Our experimental environment is based on the original HDFS Federation structure framework by adding an Nginx proxy module.

As shown in Fig. 2, this module can allocate metadata to the MDS reasonably by calling the

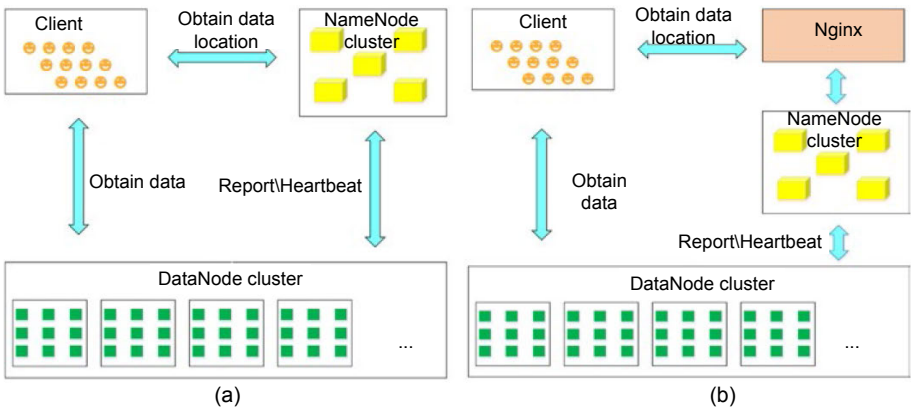

Fig. 2 Architectures of the HDFS Federation strategy (a) and our improved algorithm (b)

proposed MDLB algorithm. We select BBLA and ADMLB algorithms, which represent the methods of static and dynamic hashing, respectively. In this way, we achieve an efficient dynamic and adaptive load balancing algorithm. Experimental results show that the algorithm can fully consider the state of the MDS in the case of metadata load changes. Compared with the existing load balancing algorithms, our method can dynamically implement load distribution when the load on the MDS changes. The adjustment is reasonable, and we obtain a short response time. The goal of dynamic balancing of the MDS load is finally achieved.

The main contributions of this study are as follows:

1. We fully consider metrics of MDSs, such as CPU computing performance, memory performance, I/O performance, and disk size. These metrics make us fully consider the multiple indicators in the distance function, making the results comprehensive and accurate.

2. We reduce the deficiencies of existing hashing-based load balancing algorithms, so that the improved distributed storage cluster can dynamically adjust load based on the current MDS state.

3. When dealing with a surge of access to a certain MDS node, the entire system can respond quickly, re-allocate the metadata reasonably, and reach the load balancing state again by the MDLB algorithm.

4. We introduce the RL method into load balancing strategy design. Using the indicator of reward, the selection of parameters in the framework is reasonable, and the metadata is justifiably distributed to the MDS.

2 Related works

In the Hadoop 1.0 version, there is only one NameNode per cluster, which will cause cluster failure and node scalability. Therefore, in the Hadoop 2.0 version, HDFS high availability (HA) and HDFS Federation (https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/Federation.html) are introduced. Strategies are used to solve the problem of single point failure and NameNode expansion. However, the HDFS Federation strategy (Fig. 2a) has brought a bottleneck problem for the system performance and does not solve the load balancing problem. Azzedin (2013) proposed the NameNode clustered using chord (NCUC) strategy, which provides fast hashing calculation using a chord protocol (Stoica, et al., 2001) in a distributed NameNode cluster without the need for manual participation in signing metadata. Searching for the target node is fast, but the NUCC strategy does not consider server heterogeneity or dynamic load balancing in metadata allocation.

The subtree segmentation method assigns the directory hierarchy of a file to different server nodes statically or dynamically, and the metadata of the same directory subtree is managed by the same node. It is divided into static segmentation or dynamic segmentation. Wu JJ et al. (2010) proposed a dynamic programming method and a binary tree search method to solve the problem of metadata segmentation, and divided a certain subset of metadata into an MDS cluster. However, this method does not consider the changes of metadata load. The dynamic dir-grain method (Xiong et al., 2005) dynamically divides the namespace into hierarchical units with an adjustable

size, thus maintaining the locality of the namespace and distributing the load evenly. However, the method does not consider the case of increasing or deleting servers. Hua et al. (2008) and Zhu et al. (2008) used the bloom filter technology to efficiently locate MDSs without considering metadata allocation or load balancing issues. The hashing method distributes metadata among servers based on hash values. It is divided into static hashing, lazy hybrid (Manindra and Thiagarajan, 2004), directory path hashing (Liu and Zhou, 2007), and dynamic hashing (Li et al., 2006). The dynamic load balancing NameNode algorithm was proposed by Zhang LJ et al. (2017). This method qualitatively considers the migration of hotspot metadata under high and low loads, but has no quantitative analysis and lacks dynamic and adaptability.

BBLA (Schindelhauer and Schomaker, 2005) is based on a logarithmic approach that distributes metadata moment, and lacks dynamic response and adaptability. ADMLB (Chen et al., 2013) uses the BBLA algorithm to distribute load, uses the incremental load balancing algorithm (IBLA) (Zhang HY and Wang, 2018) to re-load balance, and uses fixed parameters when introducing access delay and the control function. The load distribution cannot reach the best level.

Q_learning is a classic RL algorithm that uses Q functions to find the optimal action-selection strategy. It evaluates which action should be selected based on the action value function, which determines the reward expectation value in a particular state, and in which state a particular action is taken. Based on the idea of Q_learning algorithm, many improved algorithms have been proposed, such as the deep Q-network algorithm (Hausknecht and Stone, 2015) and double-deep Q-network (DQN) (van Hasselt et al., 2016). The dueling network architecture (dueling DQN) (Wang ZY et al., 2016) helps generalize actions by changing the network structure. Noisy DQN (Fortunato et al., 2017) adds noise to increase the load of heterogeneous MDS by introducing a distance function between the MDSs and metadata. The metadata is distributed to the closest server, and the load is reasonably distributed at the initial time. However, the method changes with time. When the load of the MDS changes, the server load is allocated according to the server computing power at the beginning of the model. Rainbow (Hessel et al., 2018) is used to examine six extensions to the DQN algorithm, empirically studying their combination and achieving good experimental results in terms of data efficiency and final performance.

3 Problem definition

The proposed algorithm is based on the cluster architecture (Fig. 2b). We improve the existing HDFS Federation architecture, which consists of four modules, i.e., the client (client module), NameNode cluster (MDS cluster module), Nginx agent (MDS management module), and DataNode cluster. The Nginx agents, which are introduced to handle the request from the client, send the request to the NameNode cluster (MDS cluster), allocate the metadata to each MDS by calling the load balancing algorithm, and finally achieve the purpose of load balancing.

We give the following formal definitions of the metadata load balancing problem to be solved in this study:

Definition 1 (MDS set) (Chen et al., 2013) The MDS set in the storage system is $F=\{f_1, f_2, \ldots, f_n\}$, where n is the number of servers and f_i (i=1, 2, …, n) represents the i^{th} MDS server.

Definition 2 (Computing capacity of the server) (Chen et al., 2013) Let $c_i(t)$ be the computing capacity value at time t. Thus, the computing capacity set of all servers at time t is $C(t)=\{c_1(t), c_2(t), \ldots, c_n(t)\}$. $C(0)$ (i.e., t=0) represents the initial computing capacity set of all servers.

Definition 3 (Metadata set) (Chen et al., 2013) The metadata set in the storage system is $X=\{x_1, x_2, \ldots, x_m\}$, where m represents the number of pieces of metadata and x_i (i=1, 2, …, m) represents the i^{th} piece of metadata.

Definition 4 (MDS load) (Chen et al., 2013) The MDS load in the storage system changes as the number of user accesses changes. The load size of server $f_i \in F$ is load_i, and its relative load size is also load_i compared with the loads of all MDSs.

Definition 5 (Function of load layout) (Chen et al., 2013) Hotspot files appear in the storage system and then hotspot metadata appears, resulting in unbalanced MDS cluster load. The dynamic data layout function g: $X \rightarrow F$ is used to map metadata set X to

MDS set F according to its load size. For a certain time t, the following inequation is satisfied:

$$\forall f_i, f_j \in F, \xi > 0,$$
$$\left| \frac{\sum_{x_k \in X \wedge g(x_k)=f_i} \text{load}_i(t)}{c_i(t)} - \frac{\sum_{x_k \in X \wedge g(x_k)=f_j} \text{load}_j(t)}{c_j(t)} \right| < \xi, \quad (1)$$

meaning that after all the metadata is mapped to the corresponding server, the ratio of the MDS load to the computing power of the MDS differs within a small range of values. Then the local load function g is fair. So, the goal of metadata load balancing is to minimize

$$\sum_{f_i \in F} \left| c_i(t) - \frac{\sum_{x_k \in X \wedge g(x_k)=f_j} \text{load}_j(t)}{\sum_{f_j \in F} \text{load}_j(t)} \right|, \quad (2)$$

indicating that for any server in the server pool, the relative load of this server matches its computing power as much as possible.

Definition 6 (Latency of MDS) The set of metadata requests processed by MDS f_i (i=1, 2, …, n) in the latest period of time t is Q_i=$\{q_1, q_2, \ldots, q_y\}$, and its delay time in the system is defined as $\beta_i(t) = \frac{1}{y} \sum_{q_i \in Q_i} (w_i + s_i)$, where w_i and s_i represent the waiting time and service time of metadata requests $q_i \in Q_i$ in the system, respectively.

Definition 7 (Average delay of the MDS system) For all MDS servers $f_i \in F$ (i=1, 2, …, n), there is a delay time $\beta_i(t)$ for time t, and then the average delay time of the system at time t is $\beta_{\text{ave}}(t) = \frac{1}{n} \sum_{f_i \in F} \beta_i(t)$.

Definition 8 (Load balancing state of MDSs) When performing load distribution, if there is a moment and for a long period after this moment, the delay time $\beta_i(t)$ (i=1, 2, …, n) for all MDSs and the average delay time of the system at that time satisfy $|\beta_i(t)-\beta_{\text{ave}}(t)| \leq 0.05\beta_{\text{ave}}(t)$, then we deem that the system has reached the load balancing state of the MDS.

Definition 9 (Adjustment time of MDSs) When the system reaches the load balancing state for the first time, for a certain MDS server f_i (i=1, 2, …, n), the current time is recorded as the adjustment time of MDS.

Definition 10 (Delay overshoot of MDSs) When performing metadata load allocation at time t, for all MDS servers $f_i \in F$ (i=1, 2, …, n), a server delay time reaches the maximum; this delay time is represented as $\beta_{\max}(t)$ and the time is recorded as $t_{\max}$. The metadata system load eventually reaches the equilibrium, and the server delay time is β_{fin}. The delay overshoot of the MDSs is defined as $p=(\beta_{\max}-\beta_{\text{fin}})/\beta_{\text{fin}}$.

4 The proposed method

We propose a new dynamic metadata load balancing mechanism based on the basic deep Q-network algorithm (Mnih et al., 2013).

4.1 Reinforcement learning and the Q_learning algorithm

In this subsection, we review the RL model and the Q_learning algorithm.

RL is an important branch of machine learning (Sutton and Barto, 2018). Its essence is to solve the problem of automatic decision making and make continuous decisions.

As shown in Fig. 3a, the RL framework consists of four main elements, i.e., agent, environmental status, action, and reward. The goal of RL is to obtain more rewards. The agent operates the environment element by taking actions and moves from one state to

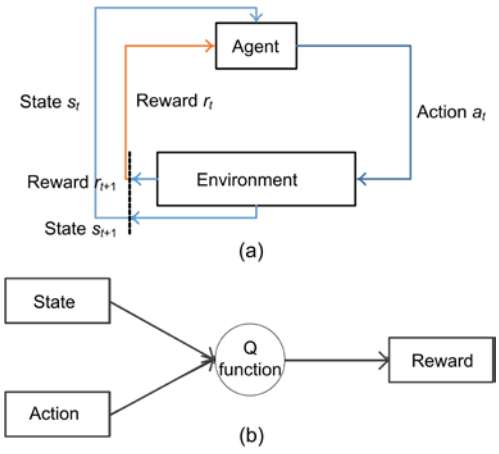

Fig. 3 Sketch maps of the reinforcement learning (RL) (a) and Q_learning (b) algorithms

The RL framework consists of four main elements, i.e., agent, environmental status, action, and reward

another. After completing the action, the agent is rewarded. Agents maximize their rewards by changing their actions constantly.

The Q_learning algorithm (Fig. 3b) is used to learn the action value function, which has two inputs (i.e., status and action) and outputs the future expected value of this action.

4.2 RL model for MDLB

In this subsection, we give a specific model of metadata load balancing.

As shown in Fig. 4, our model includes three modules, i.e., the policy selection network, metadata load balancing network, and parameter update network. We use a two-layer multilayer perceptron (MLP) model (Amari et al., 2018) in the policy selection network.

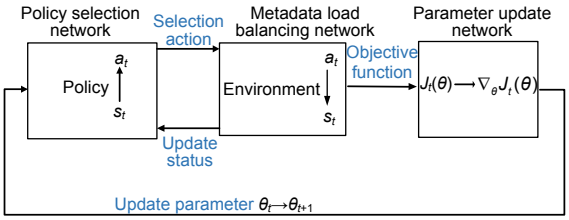

Fig. 4 Metadata dynamic load balancing mechanism model

4.2.1 Policy selection network

At time t, the policy selection network accepts the environmental state s_t of the metadata load balancing network (or called the real environment). According to the parameter θ_t of the parameter update network, the corresponding strategy $\pi(a_t|s_t, \theta_t)$ is formulated. With a comprehensive consideration of environmental state s_t and parameter θ_t, the most suitable action argmax$(\pi(a_t|s_t, \theta_t))$ at time t is selected, and a_t is passed to the metadata load balancing network.

4.2.2 Metadata load balancing network

At time t, the metadata load balancing network accepts the action a_t given by the policy selection network. By operating the environmental state s_t in the real environment, the network reaccepts the delay of each server and recalculates the computing capacity of each server. As a result, the environmental state is transformed from s_t to s_{t+1}, and s_{t+1} finally returns to the policy selection network. As the environmental state changes, the network is rewarded with r_t, obtains an objective function $J_t(\theta)$ at time t, and then passes the objective function $J_t(\theta)$ to the parameter update network.

4.2.3 Parameter update network

At time t, the parameter update network accepts $J_t(\theta)$ of the metadata load balancing network, and updates the parameter from θ_t to θ_{t+1} with the policy gradient algorithm and the RL method. Then, θ_{t+1} is passed to the policy selection network.

4.3 Metadata dynamic load balancing mechanism

In this subsection, we give a concrete model of the dynamic load balancing mechanism of metadata.

The environmental state refers to the computing capacity set of each MDS at a certain moment. At time t, the environmental state s_t can be expressed as $s_t=\{C(t)\}=\{c_1(t), c_2(t), \ldots, c_n(t)\}$.

Action refers to the action taken at a certain time according to the environmental state and strategy. Action space refers to the set of local load functions of data on the server $G=\{g_{j,i}\}$, where j=1, 2, …, m, i=1, 2, …, n, and G is the metadata load of X put on the MDS set F, i.e., $g(x_j)=f_i$.

The strategy is a rule set at a certain moment based on parameters. However, the strategy model here is fixed, while parameters in the strategy are constantly updated, so the strategy is dynamic to a certain degree. For example, at time t, the strategy $\pi(a_t|s_t, \theta_t)$ is the minimum value of $d(x_j, f_i)$, where $d(x_j, f_i)$ represents the distance between metadata x_j and server f_i.

According to BBLA, for any metadata x_j, the strategy is to map x_j to the closest metadata server f_i. h_1:$X\rightarrow$[0, 1] maps the metadata $x_i\in X$ to a point on the ring whose range is [0, 1], and parameter h_1 is the full name of the path of the file directory. h_2:$F\rightarrow$[0, 1] maps $f_i\in F$ to a point on the ring whose range is also [0, 1], and h_2 takes addr$_i$ as the parameters, expressing the Internet Protocol (IP) address of the MDS and the external port string of this server. In this study, hash functions h_1 and h_2 take the SHA1 function. $d(x_j, f_i)$ is defined as

$$d(x_j, f_i)=\frac{-\ln[1-|h_1(x_j)-h_2(f_i)|]}{c_i(t)}. \quad (3)$$

For a certain metadata, to ensure that the likelihood of selecting the nearest MDS is maximized, we use the strategy of softmax (Kanai et al., 2018). So, the strategy $\pi(a_t|s_t, \theta_t)$ can be presented as

$$\forall j,\ \pi(a_t \mid s_t, \theta_t) = \pi(g_{j,i} \mid C(t), \theta_t) = \arg\max \frac{\exp[-d(x_i, f_j)]}{\sum_{f_k \in F} \exp[-d(x_i, f_k)]}. \tag{4}$$

In the definition of distance function $d(x_j, f_i)$ in Eq. (3), $c_i(t)$ is the processing capability of the i^{th} MDS. When there is no metadata stored in the MDS, $c_i(0)$ is calculated as

$$c_i(0)=r_1\cdot \text{cpu}_i+r_2\cdot \text{mem}_i+r_3\cdot \text{io}_i+r_4\cdot \text{disk}_i, \tag{5}$$

where r_1, r_2, r_3, and r_4 represent the CPU computing performance, memory performance, I/O performance, and disk size of the i^{th} MDS, respectively, satisfying the following relationship:

$$r_1+r_2+r_3+r_4=1. \tag{6}$$

These four parameters can be calculated according to the TOPSIS algorithm (Tzeng and Huang, 2011) to obtain the optimal values. Here, we have r_1=0.116, r_2=0.368, r_3=0.258, and r_4=0.258.

The MDS load is related to the resource utilization of the MDS. The definition of resource utilization rate $\alpha_i(t)$ of MDS server f_i is expressed as

$$\alpha_i(t) = r_1 \cdot \text{cpu}(U_i(t)) + r_2 \cdot \text{mem}(U_i(t)) + r_3 \cdot \text{io}(U_i(t)) + r_4 \cdot \text{disk}(U_i(t)), \tag{7}$$

where $\text{cpu}(U_i(t))$, $\text{mem}(U_i(t))$, $\text{io}(U_i(t))$, and $\text{disk}(U_i(t))$ represent the resource utilization of the CPU, memory, I/O, and disk size of the i^{th} MDS, respectively, all measured through the program every 200 ms in the experiment.

To avoid a sudden change in the resource utilization of the MDS at a certain time, a weighted moving average is used to calculate the resource utilization of the modified MDS:

$$a_i'(t) = \mu_t \cdot \alpha_i(t) + (1-\mu_t)\cdot a_i'(t-1), \tag{8}$$

where $0<\mu_t<1$.

Assume that the relative load of the MDS at $\text{load}_i(t)$ is related only to the resource utilization of the server at time t:

$$\text{load}_i(t) = a_i'(t). \tag{9}$$

Define the average load of MDS set $F=\{f_1, f_2, \ldots, f_n\}$ as

$$\text{load}(t)=\frac{1}{n}\sum_{f_i \in F} \text{load}_i(t). \tag{10}$$

Using the control function in FAST TCP, the computing capacity $c_i(t)$ for server f_i at time t can be calculated by

$$c_i(t) = (1-v_t)c_i(t-1) + v_t\left(\frac{\text{load}(t)}{\text{load}_i(t)}c_i(t-1) + c_i(0)\right). \tag{11}$$

At time t, for strategy $\pi(a_t|s_t, \theta_t)$, state s_t refers to the computing capability set, i.e., $s_t=\{C(t)\}=\{c_1(t), c_2(t), \ldots, c_n(t)\}$, parameter θ_t refers to $\{\mu_t, v_t\}$, and the action refers to $a_t = \{g_{1,i_1}, g_{2,i_2}, \cdots, g_{m,i_m}\}$, $i \in \{1, 2, \ldots, n\}$.

4.4 Reward and objective function

In terms of the intensive learning strategy of text, reward refers to the feedback value obtained after performing the best action according to the corresponding strategy in the actual environment at time t. The objective function refers to a weighted feedback value obtained based on the time-weighted index and policy-weighted probability according to the reward at that moment.

According to our optimized goal, for instance, in Eq. (2), the load of each MDS is required to match its computing power. Thus, at time t, reward r_t is calculated as

$$r_t=\sum_{k=0}^{t}\gamma^k\sum_{i=1}^{n}\left|c_i(k)-\frac{\sum_{x_k \in X \wedge g(x_k)=f_j}\text{load}_j(k)}{\sum_{f_j \in F}\text{load}_j(k)}\right|, \tag{12}$$

where γ is a time-weighted parameter.

The objective function $J_t(\theta)$ is expressed as

$$\begin{aligned} J_t(\theta) &= E_{(s_t,a_t)\sim P_\theta(s_t,a_t)} r_t = P(s_0 a_0, s_1 a_1, \cdots, s_t a_t) \\ &= P(s_0)\left[\prod_{m=0}^{t-1} \pi_\theta(a_m \mid s_m) P(s_{m+1} \mid s_m, a_m)\right] r_t \\ &= r_t \prod_{m=0}^{t-1} \pi_\theta(a_m \mid s_m). \end{aligned} \tag{13}$$

Since the initial state s_0 is fixed and the environmental state s_{m+1} is completely determined by s_m and a_m, we have $P(s_0)=1$ and $P(s_{m+1}|s_m, a_m)=1$.

From the maximum likelihood gradient method and the reinforced gradient update algorithm, parameters are updated according to the following gradient equation:

$$\nabla_\theta J_t(\theta) = r_t \sum_{m=0}^{t-1} \nabla_\theta \log[\pi_\theta(a_m \mid s_m)]. \tag{14}$$

The result of the updated parameter is

$$\theta_{t+1} \leftarrow \theta_t + \eta r_t \sum_{m=0}^{t-1} \nabla_\theta \log[\pi_\theta(a_m \mid s_m)],\ \eta \in (0,1). \tag{15}$$

4.5 Algorithm complexity analysis

The parameter gradient is updated by stochastic gradient descent, and the time complexity is $O(n)$. Although the time complexity of the algorithms such as BBLA is increased, the adaptability and dynamics of system load balancing are effectively enhanced within an acceptable range.

5 Experiments

From the cluster architecture (Fig. 2b), to test the feasibility of the dynamic load balancing algorithm based on the RL method, we build a cluster experiment environment. The specific parameters of the environment are as follows: general-purpose CPU x86, 12 cores, and 32 GB memory. Development and testing are deployed in the Linux environment with a CentOS7 system, and the Hadoop version is Hadoop 2.7.3. Eclipse 4.7.2 is used in the development environment. Table 1 describes the specific configurations of the five NameNodes. The experimental interval is 200 ms. Five heterogeneous NameNodes and 10 DataNodes are deployed for testing and comparison with the existing distributed NameNode policies, i.e., HDFS Federation and BBLA. We use the mdtest tool to test each MDS. NameNode1 performs the worst and NameNode5 the best.

BBLA is based on a logarithmic approach that distributes metadata load to heterogeneous MDS, and the ADMLB algorithm is a specific improvement of the dynamic hashing algorithm. Our algorithm is based mainly on the dynamic hashing method, which means that the system can be adjusted to achieve dynamic load balancing. So, we select BBLA and ADMLB algorithms, which represent the methods of static and dynamic hashing, respectively.

5.1 Metadata server resource utilization analysis

First, we observe the resource utilization of each NameNode in three sets of experiments mentioned above. The experiments are sampled every 10 min. Fig. 5a shows that HDFS Federation does not consider load balancing and the resource utilization of each server changes as time passes. Figs. 5b and 5c show the changes of the resource utilization of each server under the BBLA and MDLB algorithms, respectively.

As we can see from Fig. 5a, because of the randomness of access, the resource utilization of each server is substantially different under the original Hadoop mechanism. NameNode1 performs badly, so its resource utilization is high; NameNode5 is just the opposite, and the overall hardware utilization is relatively low.

Table 1 NameNode server configurations

NameNode server	CPU	Memory	Disk	Bandwidth (GB)
NameNode1	FT1500	32 GB DDR4	240 GB SSD	1
NameNode2	ARM	32 GB DDR4	240 GB SSD*2	1
NameNode3	ARM*2	32 GB DDR4*2	240 GB SSD*4	1
NameNode4	E52620V4*4	32 GB DDR4*2	240 GB SSD*4	1
NameNode5	E52620V4*4	32 GB DDR4*4	240 GB SSD*8	1

It can be seen from Figs. 5b and 5c that after using the BBLA and MDLB algorithms, the resource utilization of the entire cluster is improved and the difference is reduced. However, when MDLB is initiated, there are few differences between the resource utilization of each NameNode, and the fluctuation is reduced compared with the BBLA algorithm. The results show that, compared with the BBLA algorithm, the proposed method performs better in addressing the load balancing problem of heterogeneous servers.

5.2 Dynamic load balancing analysis

In the experiments, as data from the client is stored continuously in the cluster, the client should continuously send requests to the NameNode cluster. Since load balancing is not considered in Hadoop's original HDFS Federation mechanism, we do not consider that issue in this subsection neither. On the other hand, the loading status of the MDS can be ascertained from the server's delay time (Wu CX and Burns, 2003). Fig. 6 shows the time-varying curves of the metadata delay time measured by BBLA, ADMLB, and MDLB. Table 2 shows the comparison results of BBLA, ADMLB, and MDLB corresponding to the curves in Fig. 6.

It can be seen from Fig. 6 that BBLA can distribute only the metadata to the MDS according to the performance of the MDS at the initial moment. It cannot be dynamically adjusted according to the change of the root data when the system is running. Fig. 6b shows that the adjustment time of ADMLB is 9 s, and Fig. 6c shows that under the MDLB algorithm, the system reaches load balancing in 8 s. Although ADMLB can be run in the system during the adjustment of the metadata load because its

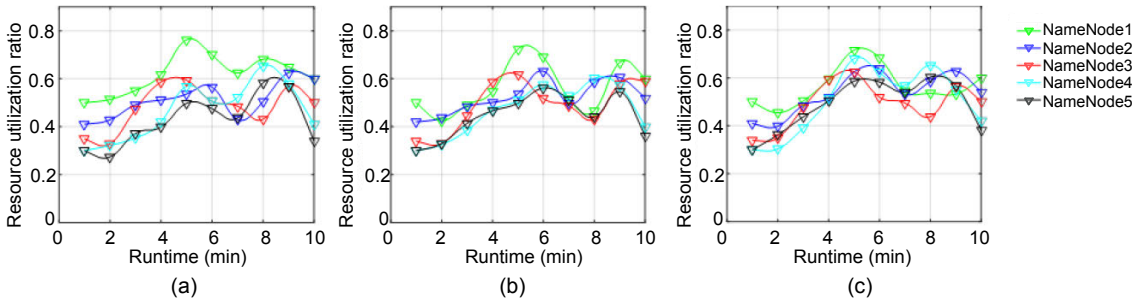

Fig. 5 Resource utilization ratios of each NameNode under different algorithms: (a) HDFS Federation; (b) BBLA; (c) MDLB

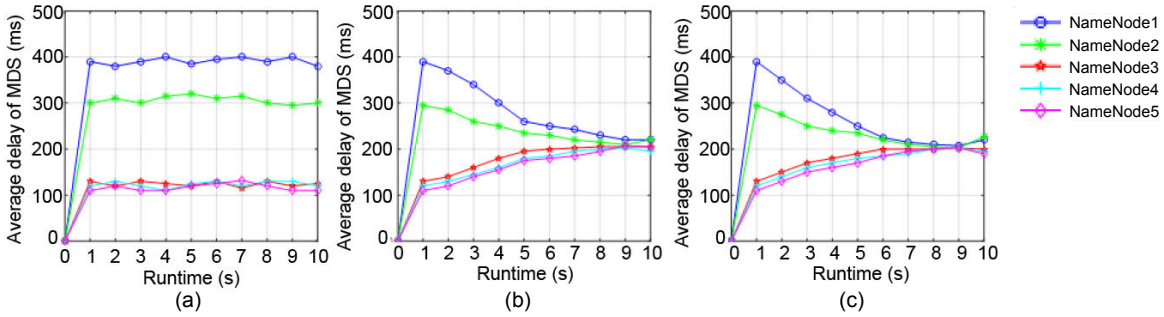

Fig. 6 Time-varying curves of the metadata delay time: (a) BBLA; (b) ADMLB; (c) MDLB

Table 2 Comparison results of the BBLA, ADMLB, and MDLB algorithms

Algorithm	Average delay of MDS (ms)					Variance of average delay	Adjustment time
	NameNode1	NameNode2	NameNode3	NameNode4	NameNode5		
BBLA	390	310	125	120	110	16 930.0	–
ADMLB	220	215	210	207	203	47.0	9 s
MDLB	215	215	205	210	210	17.5	8 s

– means that the BBLA algorithm does not converge in terms of the adjustment time

adjustment parameters are set to fixed values at the initial time, it is impossible to select the most appropriate adjustment strategy according to the actual situation of the system. The MDLB algorithm combined with the Q_learning algorithm, performing the dynamic calculation of parameters, can ensure that the system adjusts the load of the MDS in an optimal way at all times, thus achieving a good dynamic load balancing effect.

As can be seen from Fig. 7, at point *A*, the system reaches the equilibrium for the first time. ADMLB and MDLB have similar adjustment effect, but the adjustment time of ADMLB is slightly larger than that of MDLB. At point *B*, there is a large amount of data access to NameNode1, so the metadata delay of NameNode1 will rapidly increase and the load balancing mechanism starts to adjust. Because of the fixed parameter selected by the ADMLB algorithm, this algorithm cannot adjust the system to a good load balancing state, and only the load of several MDSs is approximated. Under the MDLB algorithm, the system reaches load balancing again in 7 s. As we can see, the MDLB algorithm combines the advantages of RL through the dynamic calculation of the adjustment parameter, and the system can achieve ideal dynamic load balancing and reaches a new load balancing point *C*.

Table 2 shows that, from the variance of average delay, the load of all NameNodes is the most balanced when using MDLB. MDLB performs the best in load balancing. Because BBLA cannot allocate the load according to MDS's dynamic load, it does not converge in terms of the adjustment time. Although ADMLB can adjust the load distribution of the metadata, its equalization ability is slightly lower than that of MDLB, and its adjustment time is 12.5% more than that of MDLB.

5.3 Metadata server delay overshoot and adjustment time analysis

Fig. 8 shows the relationship between the delay overshoot of the MDSs (defined in Definition 10) and the adjustment time under the requirement of a large amount of data in different cases.

We conduct experiments in different situations on sudden visits of a large amount of data, and calculate the overshoot and corresponding adjustment time for different experiments. As can be seen from Fig. 8, when the system is visited suddenly by a large amount of data, the overshoot and adjustment time of the system will increase accordingly. When the system is visited by a small amount of data, the overshoot and adjustment time will be reduced. However, under different overshoot conditions, the difference of each adjustment time of the system is not significant, indicating that under the proposed algorithm, the system can adjust the unbalanced load dynamically and quickly.

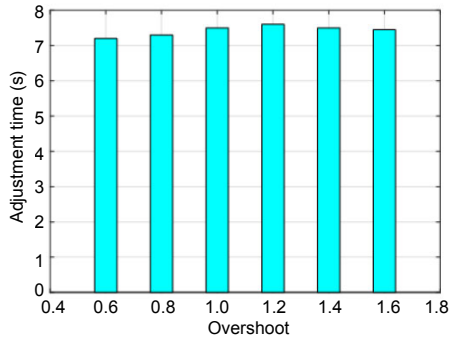

Fig. 8 Relationship between metadata server delay overshoot and adjustment time

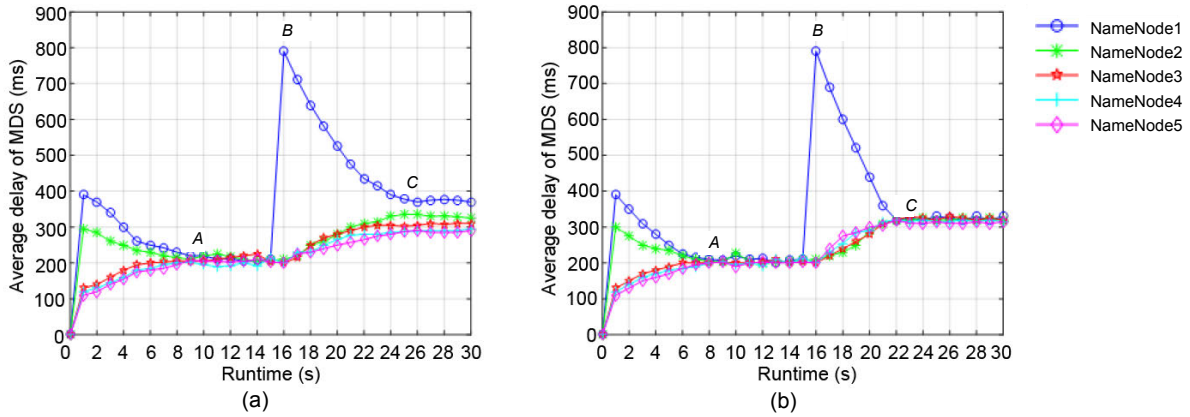

Fig. 7 Time-varying curves of the metadata delay time in systems with a large amount of data: (a) ADMLB; (b) MDLB

5.4 Metadata distribution analysis

We conduct experiments to compare the existing HDFS Federation and BBLA algorithms on metadata distribution. At the initial moment, based on different performances of NameNodes, the numbers of files assigned to different NameNodes are different. The files used in the experiments are evenly distributed across the directory hierarchy. The metadata sever distribution results are shown in Fig. 9.

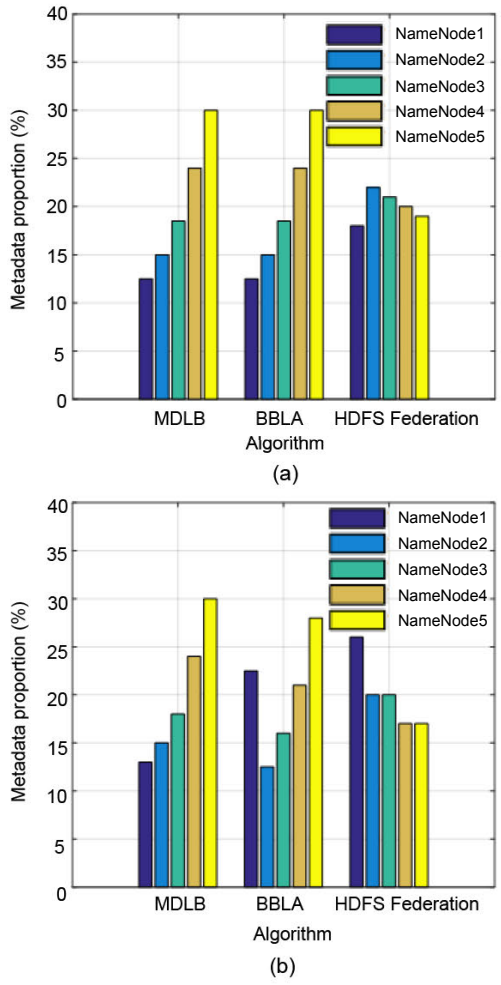

Fig. 9 Metadata server distribution results at the initial time (a) and after system re-stabilization when there is a large amount of data (b)

The results illustrate that the proposed algorithm and the basic load balancing algorithm fully consider the heterogeneous MDS, and that there is a lot of metadata stored in the MDS with strong processing capability. When a large number of file requests are suddenly received by NameNode1 at a certain moment, the proposed algorithm can finally distribute the newly arriving metadata in a balanced manner according to the computing power of the NameNode. In contrast, due to the lack of dynamic adjustment in the basic load balancing algorithm, BBLA and HDFS Federation cannot be dynamically adjusted.

5.5 Discussion

The results show that the load balancing algorithm proposed in this study has good performance in adjusting the heterogeneous server load. Compared with the BBLA and ADMLB algorithms, the MDLB method can dynamically adjust not only the server load, but also the metadata access on the server quickly and dynamically in the case of sudden data increase, which indicates the good adaptability of the proposed algorithm.

6 Conclusions

Based on the existing dynamic hashing based metadata load balancing algorithm, we have introduced the relevant adjustment parameters of RL. When applied to Hadoop's HDFS File System, by introducing the Nginx proxy, the system resource utilization and access latency of the MDS are quantified, considering the performance difference of the NameNodes. In this way, the system has dynamic load balancing capabilities, and the load of each NameNode can be automatically adjusted without manual intervention. The feasibility of the algorithm is proved by experiments. Compared with the BBLA and HDFS Federation algorithms, our algorithm has higher efficiency and better dynamics and adaptability in the load balancing of metadata.

Contributors

Zhao-qi WU designed the research. Zhao-qi WU, Jin WEI, Fan ZHANG, and Wei GUO processed the data. Zhao-qi WU drafted the manuscript. Jin WEI and Fan ZHANG helped organize the manuscript. Jin WEI, Fan ZHANG, and Guang-wei XIE revised and finalized the paper.

Compliance with ethics guidelines

Zhao-qi WU, Jin WEI, Fan ZHANG, Wei GUO, and Guang-wei XIE declare that they have no conflict of interest.

References

Amari SI, Ozeki T, Karakida R, et al., 2018. Dynamics of learning in MLP: natural gradient and singularity

revisited. *Neur Comput*, 30(1):1-33. https://doi.org/10.1162/neco_a_01029

Azzedin F, 2013. Towards a scalable HDFS architecture. Proc Int Conf on Collaboration Technologies and Systems, p.155-161. https://doi.org/10.1109/CTS.2013.6567222

Chen T, Xiao N, Liu F, 2013. Adaptive metadata load balancing for object storage systems. *J Softw*, 24(2):331-342 (in Chinese). https://doi.org/10.3724/SP.J.1001.2013.04177

Fortunato M, Azar MG, Piot B, et al., 2017. Noisy networks for exploration. https://arxiv.org/abs/1706.10295

Ghemawat S, Gobioff H, Leung ST, 2003. The Google file system. Proc 19th ACM Symp on Operating Systems Principles, p.29-43. https://doi.org/10.1145/945445.945450

Hausknecht M, Stone P, 2015. Deep recurrent Q-learning for partially observable MDPs. https://arxiv.org/abs/1507.06527

Hessel M, Modayil J, van Hasselt H, et al., 2018. Rainbow: combining improvements in deep reinforcement learning. https://arxiv.org/abs/1710.02298

Hitz D, Lau J, Malcolm M, 1994. File system design for an NFS file server appliance. Proc USENIX Winter Technical Conf, p.9-19.

Howard J, Kazar M, Menees S, et al., 1987. Scale and performance in a distributed file system. *ACM SIGOPS Oper Syst Rev*, 21(5):1-2. https://doi.org/10.1145/37499.37500

Hua Y, Zhu YF, Jiang H, et al., 2008. Scalable and adaptive metadata management in ultra large-scale file systems. Proc 28th Int Conf on Distributed Computing Systems, p.403-410. https://doi.org/10.1109/ICDCS.2008.32

Kanai S, Fujiwara Y, Yamanaka Y, et al., 2018. Sigsoftmax: reanalysis of the softmax bottleneck. Proc 32nd Int Conf on Neural Information Processing Systems, p.284-294.

Lewis TG, Cook CR, 1988. Hashing for dynamic and static internal tables. *Computer*, 21(10):45-56. https://doi.org/10.1109/2.7056

Li WJ, Xue W, Shu JW, et al., 2006. Dynamic hashing: adaptive metadata management for petabyte-scale file systems. Proc 23rd IEEE/14th NASA Goddard Conf on Mass Storage Systems and Technologies, p.159-164.

Liu Z, Zhou XM, 2007. A metadata management method based on directory path. *J Softw*, 18(2):236-245 (in Chinese).

Manindra A, Thiagarajan PS, 2004. Lazy rectangular hybrid automata. Proc Int Workshop on Hybrid Systems: Computation and Control, p.1-15. https://doi.org/10.1007/978-3-540-24743-2_1

Mnih V, Kavukcuoglu K, Silver D, et al., 2013. Playing Atari with deep reinforcement learning. https://arxiv.org/abs/1312.5602

Palankar MR, Iamnitchi A, Ripeanu M, et al., 2008. Amazon S3 for science grids: a viable solution? Proc Int Workshop on Data-Aware Distributed Computing, p.55-64. https://doi.org/10.1145/1383519.1383526

Patgiri R, Dev D, Ahmed A, 2017. dMDS: uncover the hidden issues of metadata server design. In: Sa PK, Sahoo MN, Murugappan M, et al. (Eds.), Progress in Intelligent Computing Techniques: Theory, Practice, and Applications. Springer, Singapore, p.531-541. https://doi.org/10.1007/978-981-10-3373-5_53

Satyanarayanan M, Kistler JJ, Kumar P, et al., 1990. Coda: a highly available file system for a distributed workstation environment. *IEEE Trans Comput*, 39(4):447-459. https://doi.org/10.1109/12.54838

Schindelhauer C, Schomaker G, 2005. Weighted distributed hash tables. Proc 7th Annual ACM Symp on Parallelism in Algorithms and Architectures, p.218-227. https://doi.org/10.1145/1073970.1074008

Shvachko K, Kuang HR, Radia S, et al., 2010. The Hadoop distributed file system. Proc IEEE 26th Symp on Mass Storage Systems and Technologies, p.1-10. https://doi.org/10.1109/MSST.2010.5496972

Stoica I, Morris R, Karger D, et al., 2001. Chord: a scalable peer-to-peer lookup service for Internet applications. Proc Conf on Applications, Technologies, Architectures, and Protocols for Computer Communications, p.149-160. https://doi.org/10.1145/383059.383071

Sutton RS, Barto AG, 2018. Reinforcement Learning: an Introduction (2nd Ed.). MIT Press, London, UK.

Tzeng GH, Huang JJ, 2011. Multiple Attribute Decision Making: Methods and Applications. Chapman and Hall, New York, USA.

van Hasselt H, Guez A, Silver D, 2016. Deep reinforcement learning with double Q-learning. Proc 30th AAAI Conf on Artificial Intelligence, p.2094-2100.

Wang FY, Nelson M, Oral S, et al., 2013. Performance and scalability evaluation of the Ceph parallel file system. Proc 8th Parallel Data Storage Workshop, p.14-19. https://doi.org/10.1145/2538542.2538562

Wang JD, Zhang T, Song JK, et al., 2018. A survey on learning to hash. *IEEE Trans Patt Anal Mach Intell*, 40(4):769-790. https://doi.org/10.1109/TPAMI.2017.2699960

Wang ZY, Schaul T, Hessel M, et al., 2016. Dueling network architectures for deep reinforcement learning. Proc 33rd Int Conf on Machine Learning, p.1995-2003.

Wu CX, Burns R, 2003. Handling heterogeneity in shared-disk file systems. Proc ACM/IEEE Conf on Supercomputing, Article 7. https://doi.org/10.1109/SC.2003.10045

Wu JJ, Liu PF, Chung YC, 2010. Metadata partitioning for large-scale distributed storage systems. Proc IEEE 3rd Int Conf on Cloud Computing, p.212-219. https://doi.org/10.1109/CLOUD.2010.24

Xiong J, Tang RF, Wu SN, et al., 2005. An efficient metadata distribution policy for cluster file systems. Proc IEEE Int Conf on Cluster Computing, p.1-10. https://doi.org/10.1109/CLUSTR.2005.347060

Zhang HY, Wang K, 2018. Research of dynamic load balancing based on stimulated annealing algorithm. *Int J Embed Syst*, 10(3):188-195. https://doi.org/10.1504/IJES.2018.091777

Zhang LJ, Cui Y, Luo GC, et al., 2017. Dynamic load balance algorithm for big-data distributed storage. *Comput Sci*, 44(5):178-183 (in Chinese). https://doi.org/10.11896/j.issn.1002-137X.2017.05.032

Zhong S, Chen J, Yang YR, 2003. Sprite: a simple, cheat-proof, credit-based system for mobile ad-hoc networks. Proc 22nd Annual Joint Conf of the IEEE Computer and Communications Societies, p.1987-1997. https://doi.org/10.1109/INFCOM.2003.1209220

Zhu YF, Jiang H, Wang J, et al., 2008. HBA: distributed metadata management for large cluster-based storage systems. *IEEE Trans Parall Distrib Syst*, 19(6):750-763. https://doi.org/10.1109/TPDS.2007.70788