Available online at https://www.cnki.net

October 2021, 28(5): 91 – 101

The Journal of China Universities of Posts and Telecommunications

https://jcupt.bupt.edu.cn

Dynamic load balancing algorithm for distributed system

Cui Yansong, Bai Chunyu (✉)

School of Electronic Engineering, Beijing University of Posts and Telecommunications, Beijing 100876, China

Abstract

In distributed systems, it is important to adjust load distribution dynamically based on server performance and load information. Meanwhile, gray release and rapid expansion are the basic requirements to ensure reliability and stability for systems with short version iteration cycles. The traditional Hash algorithm performs poorly in gray release, rapid expansion, and load distribution. To solve these problems, a novel Hash-based dynamic mapping (HDM) load balancing algorithm was proposed. On the one hand, this algorithm can adjust the load distribution dynamically based on server performance and load information. On the other hand, it implements gray release by controlling the ratio of requests assigned to the changed nodes. Additionally, HDM has a higher expansion efficiency. Experiments show that the HDM distributes the load more reasonably, provides a more stable gray release ratio, and has a higher expansion efficiency.

Keywords load balancing, gray release, rapid expansion, dynamic mapping, Hash-based dynamic mapping (HDM)

1 Introduction

With the rapid development of the Internet, the number of Internet users has exploded. In order to meet the increasing access demand, the distributed system progressively became the mainstream system. How to do load balancing well in distributed systems became a research hotspot. In a distributed system that needs to meet the access of massive users, the reliability of the system is significant [1]. Reliability requires that the system service units be capable of redundant deployment. If there is a single server failure, the entire system can still provide standard services to users. Redundant deployment of equipment and separation of business traffic are basic strategies.

In distributed systems, the reasonable load distribution of the system has a significant impact on system reliability. However, in a cluster, which contains servers with different performance, the system load distribution may become unreasonable due to differences in server performance [2]. Therefore, it is necessary to consider how to allocate user traffic among redundant nodes reasonably based on server performance to ensure system reliability. Besides, for systems with short version iteration cycles, gray release and rapid expansion are the basic requirements to ensure system reliability and stability. Gray release smooths the transition between black and white. It means that some users use the application with the old version while the others use the application with the new version. The users' feedback on the new version can be collected to decide to expand or narrow the scope of the release of the new version. When a new version is released, it is expected that only a small

Received date: 03-11-2020

Corresponding author: Bai Chunyu, E-mail: 1812928598@qq.com

DOI: 10.19682/j.cnki.1005-8885.2021.0025

number of requests to access the new version to test its correctness and stability, and then gradually expand the release scope based on the test results. Even if there are issues with the new version, the affected scope can be completely controlled. At the same time, requests should be assigned consistently to ensure that the same requests still experience the same version. Rapid expansion requires that when new nodes are added to the server cluster, the impact on existing data should be as low as possible.

For the above situation, it is meaningful to propose an efficient and reasonable load balancing algorithm to solve the existing problems. HDM load balancing algorithm was proposed, which supports the gray release of versions and rapid expansion of clusters and can distribute load reasonably in a heterogeneous cluster. The HDM load balancing algorithm calculates each request's Hash value according to its source Internet protocol (IP) address, source port number, destination IP address, and destination port number. Then it involves these Hash values to M grids, which are assigned to N servers. The M grids are divided into two groups, the first group is mapped to the changed nodes, and the second group is mapped to the original nodes. The number of grids mapped to the changed nodes is determined by the ratio of gray release, and dynamic load adjustment is performed within the group. The HDM load balancing algorithm quantifies the performance of each server and allocates the corresponding number of grids according to the ratio of the quantified values to distribute different loads to different performance servers. Moreover, it can adjust the ratio of load distribution dynamically according to server performance and load information. Therefore, the load distribution will be more reasonable. Additionally, HDM load balancing algorithm can be extended to ensure data security through data backup. If one of the data copies is lost or an error occurs, the data can be recovered through other copies. Experiments show that HDM load balancing algorithm can better balance the cluster load, provide a more stable gray release ratio and have a higher expansion efficiency.

The rest of this paper is organized as follows. Sect. 2 introduces several existing load balancing algorithms and related research. Sect. 3 describes the HDM load balancing algorithm. Sect. 4 compares HDM with other load balancing algorithms through experiments, and analyzes the experiment results. Sect. 5 summarizes this paper.

2 Related work

Currently, load balancing algorithms can be divided into static algorithms and dynamic algorithms according to load balancing strategies [3]. Static load balancing algorithms are widely studied, which are easy to implement by distributing the load according to the fixed distribution mode [4]. However, the static load balancing algorithms lack flexibility. They cannot obtain the status of servers and global load information. Therefore, when the load on the node no longer fits the strategy, the performance of the cluster may decrease. Dynamic load balancing algorithms have better flexibility and adaptability [5]. They can dynamically adjust the load distribution based on the load information on the servers, making the load distribution of the cluster more even, thus resolving the load distribution problem between servers.

2.1 Static load balancing algorithms

Round-robin (RR) algorithm and weighted round-robin (WRR) algorithm are typical examples of static algorithms. RR distributes the requests to the server in sequence [6], while WRR can set different weights for servers with different performance and assigns requests based on weights and rounds [7]. The biggest advantage of static load balancing algorithms is simple and fast, which does not need to maintain complex routing rules and routing information and does not need to deploy a set of hardware-level load balancing equipment. But the shortcomings of these algorithms are obvious. First of all, they are only suitable for stateless service scenarios. For cluster nodes with stateful services such as cache service modules, they will seriously affect service efficiency. For the cache, selecting nodes alternately results in frequent elimination of cached data and a low hit rate, which seriously affects the user experience. Additionally, static load balancing algorithms are detrimental to version changes. For example, for the changed version, it is hoped that only 10% of the requests will experience the new version at the beginning and then gradually scale up when the new version is stable. However, the RR results in an equal probability of receiving requests from all nodes in the back end, unable to control the number of requests from specific nodes, and lacking flexibility. WRR improves RR, which can set weights for servers but it cannot control the ratio of gray release flexibly.

The Hash algorithm is also a standard static load balancing algorithm. At first, it obtains the Hash value of the requested key and then takes the remainder according to the number of servers to select a server that provides service. This approach is simple as well, for the data is dispersed evenly and the same request key will fall on the same server. However, it has a fatal drawback of needing to refactor data when the number of servers changes, such as cluster expansion or node removal. Similarly, the Hash algorithm will also bring enormous migration costs. For example, certain data are hashed according to the key and assigned to node 1 for physical storage. At this time, the cluster needs to expand, and the number of servers increases. At the next time the corresponding data will not be accessed through the same key and the same request will not be assigned to node 1. Therefore, it is necessary to migrate a large amount of data when storage expansion occurs. Besides, The Hash algorithm cannot solve the problem of gray release. The consistent Hash algorithm is a significant improvement on the Hash algorithm, which solves the problem of excessive data migration costs caused by changes in the number of nodes in the Hash algorithm [8]. Firstly, the consistent Hash algorithm calculates the Hash value of the server (node) and maps it to the circle of $0-2^{32}$. It uses the same method to obtain the Hash value of the request and maps it to the circle. Then it starts the clockwise search from the location where the request maps, and assigns the request to the first server found, as shown in Fig. 1. If it does not find the server after more than 2^{32}, the first server will accept the request. However, one of the shortcomings of this method is that it still cannot implement gray release. That is because the number of requests on the specific nodes cannot be controlled.

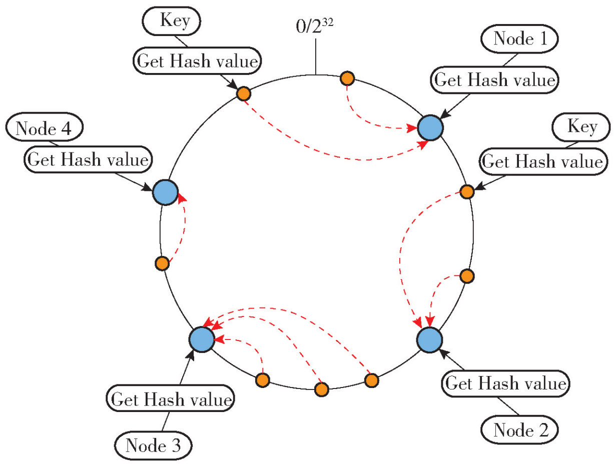


Fig. 1 Schematic diagram of consistent Hash algorithm

2.2 Dynamic load balancing algorithms

The current standard dynamic load balancing algorithms include least connections (LC) and weighted least connections (WLC). When a request arrives, LC will get the number of active connections of all servers in the cluster and assign the request to the server with the least number of current active connections [9]. WLC can set different weights for servers of different performance in the cluster and assign requests according to the weight and the number of connections [10]. Similar to static load balancing algorithms, they have some problems. On the one hand, the load generated by different requests is likely to be unequal, so using the number of server connections to measure the system load lacks objectivity. On the other hand, they cannot control the gray release of the versions well.

Tu [11] proposed a load balancing algorithm for Linux cluster systems. The load balancer collects the load information of each server and assigns requests to the server with the lowest load weight. Although the algorithm considers the load information of each server in the cluster, it cannot adjust the distribution of load in real-time. Therefore, the algorithm cannot achieve reasonable distribution of load in the case of processing massive requests.

Kulkarni et al. [12] put forward a dynamic load balancing algorithm. The load balancing server assigns requests to the server with the lowest load each time and adjusts the load weight of the server according to load increment feedback. Although this algorithm dynamically adjusts the load weight of each server, it still has limitations in terms of gray release and rapid expansion.

Sharma [13] proposed a load balancing framework based on the response time. It assigns requests among servers based on response time and pending jobs. Although the framework reduces the response time of requests, it does not consider the performance of the server. Therefore, when the server performance is different, it cannot achieve load balancing.

Srinivasan et al. [14] designed an adaptive load balancing scheme. The scheme assigns virtual nodes to servers based on their performance and moves data among servers to optimize performance. It adapts to changing workload in a heterogeneous environment. Although the scheme considers the heterogeneity of systems, it still does not solve the problem of gray release.

3 HDM load balancing algorithm

3.1 HDM load balancing algorithm

Fig. 2 outlines the design of the HDM load balancing algorithm. When the request arrives, HDM calculates each request's Hash value according to its source Internet protocol (IP) address, source port number, destination IP address, and destination port number, then divides the Hash value by the number of grids. The remainder of the division points to one of the grids that determines which server the request should be assigned to. Suppose there are M grids and N servers, some of which are the changed nodes and the others are the original nodes. HDM divides M grids into two groups, it assigns the grids of the first group to the changed nodes and assigns the grids of the second group to the original nodes. The number of grids in the first group is determined by the ratio of gray release. For example, if there are 1 000 grids and the desired ratio of gray release is 10%, which means 10% of requests to experience the new version, the number of grids in the first group can be set to 100. With the new version running steadily, HDM can increase the ratio of gray release by increasing the number of grids in the first group. At the same time, HDM considers the different performance of the server in the cluster. On the one hand, when HDM is initializing, a corresponding number of grids will map to each server according to the performance of the servers in each group. On the other hand, HDM distributes load reasonably during the operation of the cluster. HDM collects load information of each server regularly to find the two servers with the heaviest load and the lightest load in each group. It adjusts their grid allocation proportion dynamically within the group according to the server performance and load information.

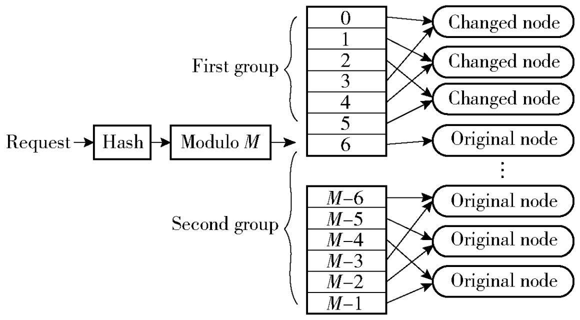


Fig. 2 The design of HDM

1) Initial grid allocation

After the number of grids in each group is determined based on the ratio of gray release, the initial grid allocation within the group can be performed based on the performance of the server. The better the performance of the server, the more grids it will be mapped and the more requests it will be processed. Performance indicators are the most direct indication of the condition of a server [15]. There are many performance indicators of the server, such as central processing unit (CPU) frequency, memory size, network bandwidth rate, disk input output (IO) rate, network card rate, etc. In order to better describe the performance of the server, HDM defines performance characteristics according to the performance indicators of the server, as shown in Eq. (1), where p_i

represents the ith performance characteristic of the server, P_i represents the ith performance indicator of the server, and $P_{i,\text{ave}}$ represents the average value of the ith performance indicator of the server cluster.

$$p_i = \frac{P_i}{P_{i,\text{ave}}} \tag{1}$$

HDM defines performance vector according to multiple performance characteristics of the server, which describes the performance of the server, as shown in Eq. (2), where I represents the number of metrics used to describe the overall performance of the server.

$$\boldsymbol{P} = (p_1, p_2, \ldots, p_i, \ldots, p_I); \quad 1 \leqslant i \leqslant I \tag{2}$$

The universally unique identifiers (UUIDs) of all servers in the cluster form a vector $\boldsymbol{U} = (u_1, u_2, \ldots, u_n)^{\mathrm{T}}$, where n represents the number of servers in the cluster. UUIDs of the entire cluster and the corresponding performance vectors can be built into a performance matrix similar to the key/value mode, as shown in Eq. (3):

$$\boldsymbol{Q} = [\boldsymbol{U} \quad \boldsymbol{P}'] = \begin{bmatrix} u_1 & p'_{11} & p'_{12} & \cdots & p'_{1I} \\ \vdots & \vdots & \vdots & & \vdots \\ u_n & p'_{n1} & p'_{n2} & \cdots & p'_{nI} \end{bmatrix} \tag{3}$$

Among them, each row in the matrix represents the performance vector of a server, and p'_{vi} represents the ith performance characteristic of the server u_v. At the same time, HDM establishes the performance weight vector $\boldsymbol{K} = (k_1, k_2, \ldots, k_i, \ldots, k_I)$, where k_i is the weight of the ith performance characteristic, $\sum_{i=1}^{I} k_i = 1$, $0 < k_i < 1$. Besides, it calculates the quantified performance of each server according to the performance matrix and weight vector, as shown in Eq. (4), where s_v is the quantified performance of server u_v.

$$\boldsymbol{S} = \boldsymbol{Q}(k_1, k_2, \ldots, k_i, \ldots, k_I)^{\mathrm{T}} = (s_1, s_2, \ldots, s_v, \ldots, s_n)^{\mathrm{T}} \tag{4}$$

The count of initial grids of each server is obtained according to the calculation results of Eq. (5), where x_v, $1 \leqslant v \leqslant n$, is the number of initial grids of server u_v.

$$\boldsymbol{X} = \boldsymbol{S}\frac{M}{\sum_{v=1}^{n} s_v} = (x_1, x_2, \ldots, x_v, \ldots, x_n)^{\mathrm{T}} \tag{5}$$

2) Dynamic grid allocation

During the operation of the cluster, HDM can implement more reasonable load balancing. HDM collects load information of each server in each cycle T to find the two servers with the heaviest load and the lightest load in each group. Besides, it adjusts grid allocation proportion dynamically within the group based on the server performance and load information. HDM uses multiple resource utilization of the server to calculate the server load. The initial load of the cluster is relatively low, so the load for the first m cycles of the cluster is not taken into account. After m cycles, the load information of the server will be obtained at the beginning of each cycle. Firstly, the load vector of the server is defined according to resource utilization, as shown in Eq. (6), where r_j, $1 \leqslant j \leqslant J$, represents the utilization of the jth resource of the server, and J represents the number of metrics used to describe the overall load of the server.

$$\boldsymbol{R} = (r_1, r_2, \ldots, r_j, \ldots, r_J); \quad 1 \leqslant j \leqslant J \tag{6}$$

The UUIDs of the entire cluster and the corresponding load vectors can form a load matrix

$$\boldsymbol{V} = [\boldsymbol{U} \quad \boldsymbol{R}'] = \begin{bmatrix} u_1 & r'_{11} & r'_{12} & \cdots & r'_{1J} \\ \vdots & \vdots & \vdots & & \vdots \\ u_n & r'_{n1} & r'_{n2} & \cdots & r'_{nJ} \end{bmatrix} \tag{7}$$

Among them, each row in the matrix represents the load vector of a server, and r'_{vj} represents the jth resource utilization of the server u_v. At the same time, HDM establishes a load weight vector $\boldsymbol{W} = (w_1, w_2, \ldots, w_j, \ldots, w_J)$, where w_j is the weight of the jth load characteristic, $\sum_{j=1}^{J} w_j = 1$, $0 < w_j < 1$. Secondly, HDM calculates the quantified load value of each server based on the load matrix and load weight vector, as shown in Eq. (8), where l_v is the quantified load value of server u_v.

$$\boldsymbol{L} = \boldsymbol{V}(w_1, w_2, \ldots, w_j, \ldots, w_J)^{\mathrm{T}} = (l_1, l_2, \ldots, l_v, \ldots, l_n)^{\mathrm{T}} \tag{8}$$

The weight of the load on each server can be calculated according to the quantified load value and quantified performance, as shown in Eq. (9), where q_v is the weight of the load on server u_v.

$$q_v = \frac{s_v}{l_v}; \quad 1 \leqslant v \leqslant n \tag{9}$$

Finally, HDM calculates the standard deviation of the load weights of all servers

$$\sigma = \sqrt{\frac{1}{n}\sum_{v=1}^{n}(q_v - \bar{q})^2} \tag{10}$$

where $\bar{q}$ is the average load weight. In each cycle, the standard deviation σ of this cycle is compared with the minimum standard deviation $\sigma_{\min}$, if $\sigma \leqslant \sigma_{\min}$, HDM updates the value of the minimum standard deviation: $\sigma_{\min} = \sigma$. If $\sigma > \sigma_{\min}$, HDM finds the two servers with the largest load weight and the smallest load weight, and the number of grids they occupy is redistributed according to the ratio of their load weights. For example, suppose that the 1st server has the largest load weight and the 2nd server has the smallest load weight. The load weight of the 1st server is set to $q_{\max}$, the number of grids it has in the previous cycle is set to N_1, the load weight of the 2nd server is set to $q_{\min}$, the number of grids it has in the previous cycle is set to N_2. The grid numbers of the 1st server and the 2nd server in this cycle are set to N_1' and N_2' respectively:

$$N_1' = (N_1 + N_2)\frac{q_{\max}}{q_{\max} + q_{\min}} \tag{11}$$

$$N_2' = (N_1 + N_2)\frac{q_{\min}}{q_{\max} + q_{\min}} \tag{12}$$

3) Flow chart of HDM

The flow chart of HDM load balancing algorithm is shown in Fig. 3.

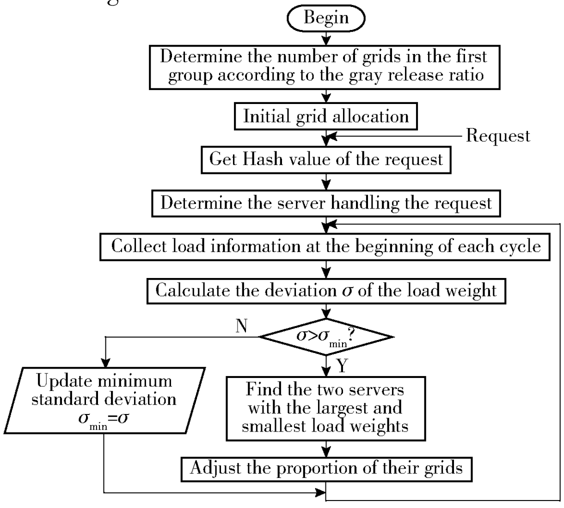

Begin
Determine the number of grids in the first group according to the gray release ratio
Initial grid allocation
Request
Get Hash value of the request
Determine the server handling the request
Collect load information at the beginning of each cycle
Calculate the deviation σ of the load weight
σ>σ_min?
N
Y
Update minimum standard deviation σ_min=σ
Find the two servers with the largest and smallest load weights
Adjust the proportion of their grids

Fig. 3 Flow chart of HDM load balancing algorithm

3.2 Rapid expansion and disaster tolerance

A good load balancing algorithm can provide high data migration efficiency when expanding or removing nodes. In this respect, HDM is superior to other load balancing algorithms. As shown in Fig. 4, assuming there are 12 grids, and node 3 is to be added to the two-node cluster, only adjust the grids. The original requests for grids 4, 5, 10, and 11 are taken over by node 3. At this time, it is only needed to migrate part of the data from node 1 and node 2 to node 3 after the expansion. It can be found from Fig. 4 that data migration only involves the grids that are replaced by new nodes, so the cost of data migration of HDM is low. When new nodes are added, rapid expansion is possible as only part of the data needs to be migrated.

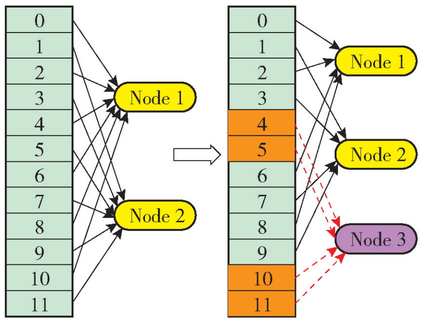
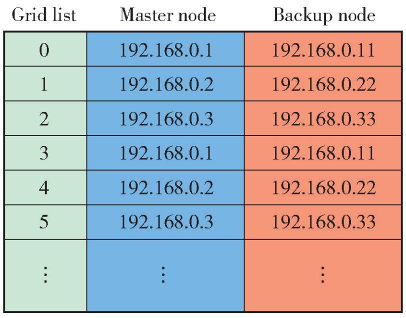


Fig. 4 Data migration of HDM load balancing algorithm

As the cornerstone of Internet applications, mass storage systems provide the most basic storage services. Data security is particularly important. Usually, the system stores several copies of a piece of data to ensure that one of the copies is lost or an error occurs, the data can be recovered through other copies [16]. The distribution of copies can be supported to implement data security by extending HDM, shown in Fig. 5. It can be found from Fig. 5 that the table contains multiple columns. Each column represents a copy of data (except the first column). For example, if there are two copies, the table will contain three columns, and the next two columns are backup nodes, which can be located in different computer rooms or Internet data centers (IDCs). The second column can be used as the master node information and the following columns as slave node information. Each time a specific grid is located, the master node will provide services first. Once the master node fails, the slave nodes will be upgraded to the master node to achieve disaster tolerance.



Grid list	Master node	Backup node
0	192.168.0.1	192.168.0.11
1	192.168.0.2	192.168.0.22
2	192.168.0.3	192.168.0.33
3	192.168.0.1	192.168.0.11
4	192.168.0.2	192.168.0.22
5	192.168.0.3	192.168.0.33
⋮	⋮	⋮

Fig. 5 Disaster tolerant

3.3 Performance analysis

In terms of load distribution, assuming that the processing ability of the vth server is a_v, $v = 1, 2, \ldots, n$, $L_v(\Delta t)$ is load quantity of the vth server in time Δt. During the dynamic adjustment of HDM, the load on the server will reach a dynamic balance and it is proportional to the processing ability of the server, as shown in Eq. (13), where a_v is the processing ability of the vth server, $L_v(\Delta t)$ is load quantity of the vth server in time Δt.

$$L_v(\Delta t) \propto a_v \tag{13}$$

In terms of gray release, assuming that the total number of grids is M, the number of grids mapped to the changed node is M_1. The ratio of gray release is

$$P_{\text{gray}} = \frac{M_1}{M} \tag{14}$$

In terms of rapid expansion, assuming that the total data volume is D, the original number of servers is C_1, the number of servers that need to be added is C_2. In order to distribute the data among all servers evenly, the data migration rate is

$$P_{\text{migrate}} = \frac{\dfrac{D}{C_1} - \dfrac{D}{C_1 + C_2}}{D} C_1 = \frac{C_2}{C_1 + C_2} \tag{15}$$

The HDM collects load information from each server in cycle T, so the average time complexity of HDM is $O(N)$, where N is the total number of servers. The HDM needs to use extra space to store the mapping relationship between the grids and servers, so the space complexity of HDM is $O(M)$, where M is the total number of grids.

4 Performance evaluation of HDM

4.1 Experimental environment

In order to verify the feasibility of HDM, a cluster system based on Nginx was constructed and the JMeter test tool was used to simulate user requests. Experiments use CRC-32 as the Hash function, which operates on four packet header fields, which are source IP address, destination IP address, source port number and destination port number. There are six servers with different performance in the cluster, a load balancer and a client. Table 1 shows the hardware configuration of the servers in the experimental environment.

Table 1 Hardware configuration of experimental environment

Node	CPU frequency/GHz	Memory/GB	Network bandwidth/(Mbit·s^{-1})
Load balancing server	2.60	4	1 000
Server 1	1.60	2	100
Server 2	2.40	4	200
Server 3	2.60	8	400
Server 4	1.60	4	100
Server 5	2.60	8	200
Server 6	2.60	8	400

In the experiments, the CPU frequency, memory size and network bandwidth rate of the server were used as performance indicators to measure the performance of the server and the utilization rate of these three resources was used as the standard to measure the load [17 – 18]. In the experiments, Hash algorithm, WRR algorithm, WLC algorithm and HDM load balancing algorithm were used to schedule user requests. The experiments tested their effects on load distribution and gray release, respectively. Server 1, Server 2 and

Server 3 are the changed nodes, the others are the original nodes. In the experiments, the values of each parameter are shown in Table 2.

Table 2 Parameters used in the experiment

Parameter	Value
M	1 000
T/s	5
m	100
I	3
$[k_1, k_2, k_3]$	[0. 6, 0. 2, 0. 2]
$[w_1, w_2, w_3]$	[0. 50, 0. 25, 0. 25]
Expected gray release ratio/%	40

4. 2 Load distribution

This paper tested the load distribution on each server according to different scheduling algorithms. The number of server queries per second (QPS) increased from 300 to 1 500 successively, each time by 300. The load distribution ratio of different QPS was measured and the average value was taken. Figs. 6 – 9 show the performance proportions of servers and the load distribution proportions of different scheduling algorithms.

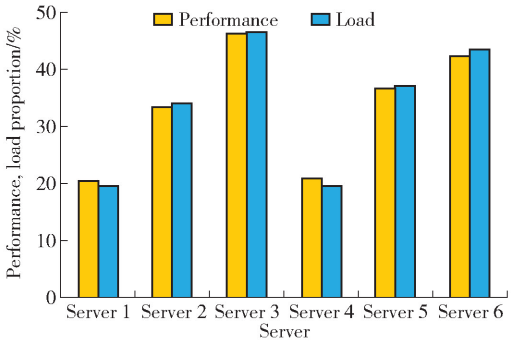


Fig. 6 Load distribution of HDM algorithm

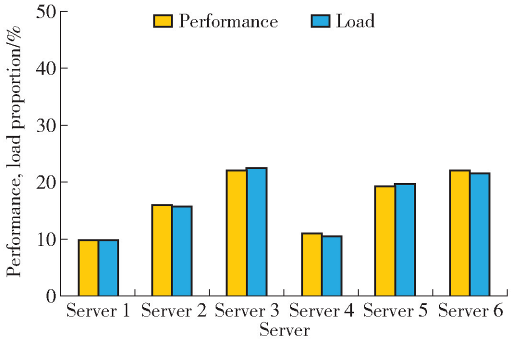


Fig. 7 Load distribution of WRR algorithm

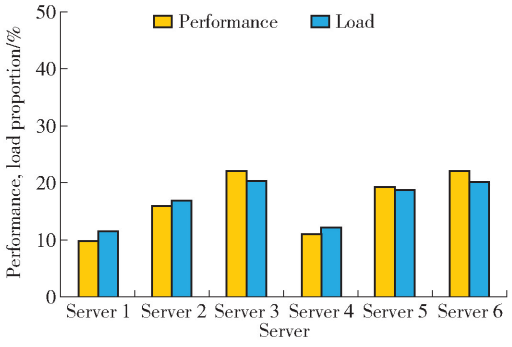


Fig. 8 Load distribution of WLC algorithm

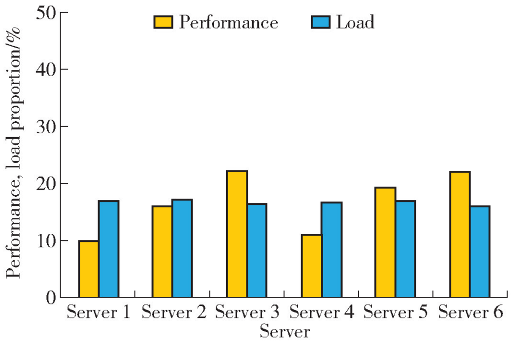


Fig. 9 Load distribution of Hash algorithm

Table 3 shows the deviations of the different scheduling algorithms. The second column of deviation in Table 3 summarizes the total difference between the load proportion and the performance proportion of all servers. As can be seen from Table 3, the WRR algorithm has the minimum deviation when distributing the load according to the performance, while the Hash algorithm does not consider the performance of the server, so it cannot distribute the load according to the performance of the server.

Table 3 Deviation of scheduling algorithm

Algorithm	Deviation/%
HDM	0. 87
WRR	0. 44
WLC	1. 35
Hash	5. 07

The load balancing effect of HDM is very close to WRR (deviation less than 0. 44%). The deviation of HDM is 0. 48% less than that of WLC, and 4. 2% less than that of Hash algorithm. As expected, the HDM can distribute the load more reasonably in the cluster according to the performance of the server.

4.3 Gray release

These experiments simulated the gray release process using different scheduling algorithms. At first, only 40% of the requests are expected to experience the new version. In the second stage, 50% of the requests are expected to experience the new version. Figs. 10 and 11 show the gray release effects of the first and second stages, respectively. In Figs. 10 – 11, QPS represents the number of requests per unit time, and the ordinate represents the ratio of gray release.

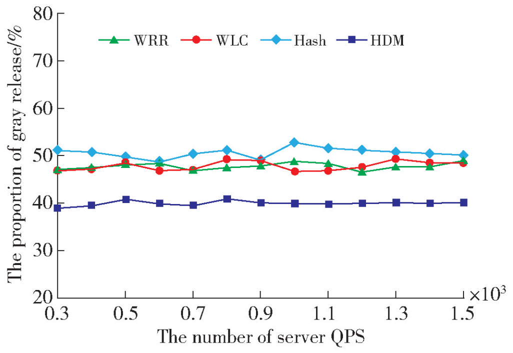


Fig. 10 The first stage of gray release

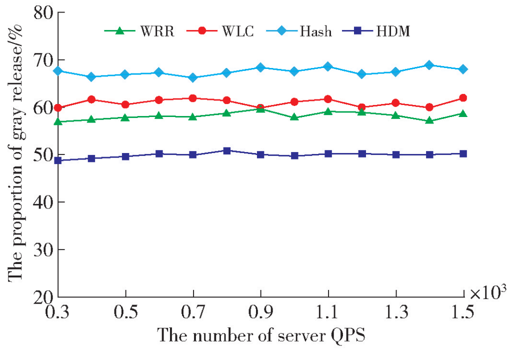


Fig. 11 The second stage of gray release

Table 4 shows the average release ratio and deviation of different scheduling algorithms. As can be seen from Table 4, the HDM meets the release expectations (error less than 0.07%) and has the smallest deviation. The HDM can make the ratio of gray release completely controllable by controlling the number of best grids in the first group, while the other three algorithms can only achieve gray release by increasing the number of changed nodes when the server performance is determined. However, their gray release ratio is not what the administrator expects, and the ratio of gray release cannot be controlled fine-grained. With the HDM, the release ratio of the new version can be controlled in a fine-grained manner.

Table 4 Average and deviation of the proportion of gray release

Algorithm	Average/%		Deviation/%	
	1st stage	2nd stage	1st stage	2nd stage
HDM	39.96	49.93	0.50	0.51
WRR	47.78	58.18	0.71	0.77
WLC	47.85	60.92	0.95	0.80
Hash	50.62	67.46	1.03	0.81

4.4 Expansion efficiency

In order to simulate data expansion, this experiment applied the load balancing algorithms to the data layer and used the database servers with the same performance. There are only two data nodes in the initial cluster, and 10 000 pieces of initial data are inserted into the cluster. Add one data node to the cluster in turn to make the data evenly distributed on each data node and observe the migration rate of data. The experimental results are shown in Fig. 12.

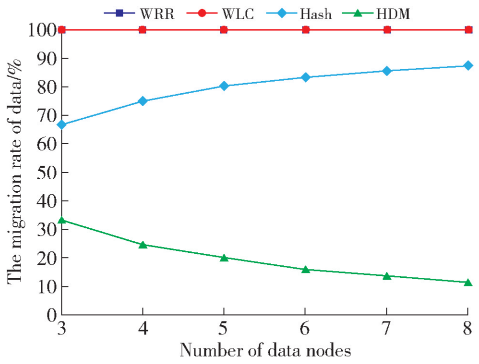


Fig. 12 The migration rate of data

The abscissa in Fig. 12 represents the number of data nodes in the cluster, and the ordinate represents the migration rate of data. As can be seen from Fig. 12, with the increase in the number of nodes in the cluster, HDM needs to migrate the least amount of data, Hash algorithm needs to migrate more and more data, while WRR and WLC need to migrate all data. It can be seen that the HDM can effectively reduce the amount of data migration during expansion and provide a higher expansion efficiency.

4.5 Analysis of experimental results

In the above experiments, the HDM load balancing algorithm was applied to web server cluster and database cluster. From the experimental results, it can be found that the HDM load balancing algorithm distributes the load according to the performance of the servers reasonably and provides an expected and stable gray release ratio. When adding new nodes to the cluster, HDM load balancing algorithm can effectively reduce the amount of data migration. This shows that in distributed systems, HDM load balancing algorithm is not only more reasonable in terms of load scheduling, but also can fine-grained control of the release of new versions. Besides, it can improve the efficiency of cluster expansion.

5 Conclusions

Distributed systems became the main trend of Internet application development. The existing load balancing algorithms cannot meet the development needs of applications. In this paper, a novel load balancing algorithm called HDM load balancing algorithm was proposed, which can adjust the load distribution dynamically according to server performance and load information. Firstly, HDM load balancing algorithm distributes the load more reasonably in the heterogeneous clusters. Secondly, HDM load balancing algorithm provides a convenient gray release mechanism for systems with short iteration cycles to make the launch of the new version fully controllable and safe. Finally, HDM load balancing algorithm reduces the amount of data migration effectively during cluster expansion and has a higher expansion efficiency. The experimental results show that the HDM load balancing algorithm distributes the load more reasonably, provides the expected and stable gray release ratio, and reduces the amount of data migration effectively. It has some prospects for practical implementation.

References

1. Zhou J, Chen Y, Wang W P, et al. A highly reliable metadata service for large-scale distributed file systems. IEEE Transactions on Parallel and Distributed Systems, 2019, 31(2): 374 – 376
2. Vinutha D C, Raju G T. Dynamic load balancing algorithm for heterogeneous hadoop cluster. Proceedings of the 2019 International Conference on Advanced Technologies in Intelligent Control, Environment, Computing and Communication Engineering (ICATIECE'19), Mar 19 – 20, 2019, Bangalore, India. Piscataway, NJ, USA: IEEE, 2019: 75 – 78
3. Ivanisenko I N, RadivilovaIgor T A. Survey of major load balancing algorithms in distributed system. Proceedings of the 2015 Information Technologies in Innovation Business (ITIB'15), Oct 7 – 9, 2015, Kharkiv, Ukraine. Piscataway, NJ, USA: IEEE, 2015: 89 – 92
4. Semchedine F, Bouallouche-Medjkoune L, AÏSsani D. Task assignment policies in distributed server systems: a survey. Journal of Network and Computer Applications, 2011, 34(4): 1123 – 1130
5. Raj J S, Fiona R. Load balancing techniques in grid environment: a survey. Proceedings of the 2013 International Conference on Computer Communication and Informatics (ICCCI'13), Jan 4 – 6, 2013, Coimbatore, India. Piscataway, NJ, USA: IEEE, 2013: 1 – 4
6. Kaur S, Kumar K, Singh J, et al. Round-robin based load balancing in software defined networking. Proceedings of the 2nd International Conference on Computing for Sustainable Global Development (INDIACom'15), Mar 11 – 13, 2015, New Delhi, India. Piscataway, NJ, USA: IEEE, 2015: 2136 – 2138
7. Sharifian S, Motamedi S A, Akbari M K. An approximation-based load-balancing algorithm with admission control for cluster web servers with dynamic workloads. The Journal of Supercomputing, 2010, 53: 440 – 463
8. Lin P, Nie H M, Ding G. Load balancing framework based on consistency hashing algorithm. Proceedings of the 2014 International Conference on Mechatronics and Control (ICMC'14), Jul 3 – 5, 2014, Jinzhou, China. Piscataway, NJ, USA: IEEE, 2014: 1504 – 1507
9. Mustafa E M, Amin M I. Load balancing algorithms round-robin (RR), least-connection and least loaded efficiency. International Journal of Computer and Information Technology, 2017, 4(2): 255 – 257
10. Liu Y Y, Fang Y K. Optimizing WLC scheduling algorithm of LVS. Proceedings of the 2010 International Conference on Computer Application and System Modeling (ICCASM'20), Oct 22 – 24, 2010, Taiyuan, China. Piscataway, NJ, USA: IEEE, 2010: 585 – 588
11. Tu J Y. An improved load balancing of Linux cluster system of Linux. Microelectronics and Computer, 2012, 29(3): 106 – 109 (in Chinese)
12. Kulkarni A K, Annappa B. Load balancing strategy for optimal peak hour performance in cloud datacenters. Proceedings of the 2015 IEEE International Conference on Signal Processing, Informatics, Communication and Energy Systems (SPICES'15), Feb 19 – 21, 2015, Kozhikode, India. Piscataway, NJ, USA: IEEE, 2015: 1 – 5

13. Sharma D. Response time based balancing of load in web server clusters. Proceedings of the 7th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO'18), Aug 29 – 31, 2018, Noida, India. Piscataway, NJ, USA: IEEE, 2018: 471 – 476
14. Srinivasan L, Varma V. Adaptive load-balancing for consistent Hashing in heterogeneous clusters. Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, May 4 – 7, 2015, Shenzhen, China. Piscataway, NJ, USA: IEEE, 2015: 1135 – 1138
15. Guo Z, Sun Y D, Jiang Q, et al. Research on the performance test method of information system based on the domestic software and hardware platform. Proceedings of the 2019 International Conference on Information Technology and Computer Application (ITCA'19), Dec 20 – 22, 2019, Guangzhou, China. Piscataway, NJ, USA: IEEE, 2019: 171 – 173
16. Yu P, Hu H W, Zhou N. Design and implementation of a MySQL database backup and recovery system. Proceeding of the 11th World Congress on Intelligent Control and Automation, Jun 29 – Jul 4, 2014, Shenyang, China, Piscataway, NJ, USA: IEEE, 2014: 5410 – 5413
17. Zhu L S, Cui J M, Xiong G F. Improved dynamic load balancing algorithm based on least-connection scheduling. Proceedings of the IEEE 4th Information Technology and Mechatronics Engineering Conference (ITOEC'18), Dec 14 – 16, 2018, Chongqing, China. Piscataway, NJ, USA: IEEE, 2018: 1858 – 1861
18. Prakash S W, Deepalakshmi P. Server-based dynamic load balancing. Proceedings of the 2017 International Conference on Networks and Advances in Computational Technologies (NetACT'17), Jul 20 – 22, 2017, Thiruvanthapuram, India. Piscataway, NJ, USA: IEEE, 2017: 25 – 28

(Editor: Wang Xuying)

From p. 35

27. Teutsch J, Jain S, Saxena P. When cryptocurrencies mine their own business. Proceedings of the 2016 International Conference on Financial Cryptography and Data Security (FC'16), 2016, Feb 22 – 26, Kralendijk, Bonaire. LNCS 9603. Berlin, Germany: Springer, 2016: 499 – 514
28. Saleh F. Blockchain without waste: proof-of-stake. The Review of Financial Studies, 2021, 34(3): 1156 – 1190
29. Dong C Y, Wang Y L, Aldweesh A, et al. Betrayal, distrust, and rationality: smart counter-collusion contracts for verifiable cloud computing. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS'17), 2017, Oct 30 – Nov 3, Dallas, TX, USA. New York, NY, USA: ACM, 2017: 211 – 227
30. Asgaonkar A, Krishnamachari B. Solving the buyer and seller's dilemma: a dual-deposit escrow smart contract for provably cheat-proof delivery and payment for a digital good without a trusted mediator. Proceedings of the 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC'19), 2019, May 14 – 17, Seoul, Republic of Korea. Piscataway, NJ, USA: IEEE, 2019: 262 – 267
31. Kang J W, Xiong Z H, Niyato D, et al. Incentivizing consensus propagation in proof-of-stake based consortium blockchain networks. IEEE Wireless Communications Letters, 2018, 8(1): 157 – 160
32. Xiong Z H, Feng S H, Niyato D, et al. Optimal pricing-based edge computing resource management in mobile blockchain. Proceedings of the 2018 IEEE International Conference on Communications (ICC'18), 2018, May 20 – 24, Kansas City, MO, USA. Piscataway, NJ, USA: IEEE, 2018: 1 – 8
33. Yang Z, Miao Y, Chen Z Y, et al. Zero-determinant strategy for the algorithm optimize of blockchain PoW consensus. Proceedings of the 36th Chinese Control Conference (CCC'17), 2017, Jul 26 – 28, Dalian, China. Piscataway, NJ, USA: IEEE, 2017: 1441 – 1446
34. Kroll J A, Davey I C, Felten E W. The economics of Bitcoin mining, or Bitcoin in the presence of adversaries. Proceedings of the 12th Workshop on the Economics of Information Security (WEIS'13), 2013, Jun 11 – 12, Washington, DC, USA. 2013: 1 – 11
35. Kroer C, Sandholm T. Extensive-form game abstraction with bounds. Proceedings of the 15th ACM Conference on Economics and Computation (EC'14), 2014, Jun 8 – 12, Palo Alto, CA, USA. New York, NY, USA: ACM, 2014: 621 – 638
36. Mehrara M, Rezaei S, Razi D H. Determinants of renewable energy consumption among ECO countries; based on Bayesian model averaging and weighted-average least square. International Letters of Social and Humanistic Sciences, 2015, 54: 96 – 109
37. Chen L, Wan P B. Product quality regulation and online movie ratings: based on Bayesian propensity score estimators. Economic Perspectives, 2020, (3): 69 – 85 (in Chinese)
38. Li B Z, Lu H Y, Ma S H. A survey of social choice function research. Systems Engineering, 2015, 33(3): 72 – 77 (in Chinese)

(Editor: Wang Xuying)