Journal of Communications and Information Networks, Vol.2, No.3, Sept. 2017
DOI: 10.1007/s41650-017-0029-3
© Posts & Telecom Press and Springer Singapore 2017

Research paper

Special Focus on Future Internet

An adaptive dynamic feedback load balancing algorithm based on QoS in distributed file system

Ming Wang*, Jianfeng Guan

Beijing University of Posts and Telecommunications, Beijing 100876, China

* Corresponding author, Email: wangmingbupt@163.com

Abstract: An adaptive dynamic load balancing algorithm based on QoS is proposed to improve the performance of load balancing in distributed file system, combining the advantages of a variety of load balancing algorithms. The new algorithm uses a tuple containing the number of files and the total file size as the QoS measure for the requested task. The master node sets a threshold for the requested task based on the QoS to filter storage nodes that meet the requirements of the task. In order to guarantee the reliability of the new algorithm, we consider the impact of CPU utilization, memory usage, disk IO occupancy rate, network bandwidth usage and hard disk usage on load balancing performance when calculating the real-time load balancing of storage nodes. The heterogeneity of the network is considered when the master node schedule task assignments to ensure the fairness of the algorithm. The comprehensive evaluation value is determined based the performance load ratio, which is calculated from the real-time load value of the storage node and a performance value after normalization. The master node assigns tasks to the storage node with the highest comprehensive evaluation value. The storage nodes provide adaptive feedback based on changes in the degree of connectivity, rather than periodic update of the load information. The actual distributed file system environment is set up on the server cluster, the performance of the new algorithm is tested through a contrast experiment. The experimental results show that the new algorithm can effectively reduce the average response time of the system, improve throughput, and enable the system load to reach a good balance.

Keywords: distributed file system, load balancing, QoS, performance load ratio, adaptive dynamic feedback

Citation: M. Wang, J. F. Guan. An adaptive dynamic feedback load balancing algorithm based on QoS in distributed file system [J]. Journal of communications and information networks, 2017, 2(3): 30-40.

1 Introduction

With the continuous improvement of the mobile communication network environment and the rapid popularization of smart phones, mobile Internet usage has risen rapidly, significantly increasing the production of network data. Thus efficiently storing and managing network data has become an urgent problem. Traditional single server storage systems have long been unable to meet data demands. Solving the problem of efficient storage and management of massive amounts of data in distributed file system[1,2] has become a research hot spot.

Distributed file system refers to the file management system for physical storage resources that are not necessarily directly connected to a local node, but through a network of server clusters. The servers can communicate with each other and coordinate, forming a large-scale system that can be used for shared storage space[3]. Distributed

Manuscript received May 29, 2017; accepted Jul. 19, 2017
This work was supported in part by the National Basic Research Program of China (“973” Program) (No. 2013CB329102).

file system largely uses the extensible Master-Slave architecture[4-6], which consists of a master node and multiple storage nodes. The master node is responsible for system management and task scheduling while storage node provides storage space. The master nodes allocate the tasks to the storage nodes through a certain load balancing algorithm[7,8]. Load balancing is the process of redistributing the work load among nodes of the distributed system to improve both resource utilization and job response time while also avoiding a situation where some nodes are heavily loaded while others are idle or doing little work[9]. Multiple storage nodes coordinate with each other to share the system load, thus improving the overall access efficiency and reliability of the distributed file system.

Distributed file system typically uses periodic dynamic feedback load balancing algorithm, but this algorithm has some disadvantages[10]. In this algorithm, the storage nodes collect their own load information, generally considering only CPU utilization and memory utilization. In a distributed file system, the vast majority of operations are file accesses. Access operations involving network data transmission and disk data perform both reads and writes. Network bandwidth and disk IO performance requirements are high for these operations, meaning the collection of load information, disk IO occupancy rate, and network bandwidth utilization must also be considered. The storage nodes periodically send their load information to the master node, but the optimal feedback interval is difficult to determine. When the feedback interval is short, information exchange between the master node and storage nodes is too frequent, which requires additional network overhead, increases network load, and affects the stability of the system. When the feedback interval is too long, the real-time accuracy of the load information obtained by the master node is reduced, resulting in an unbalanced allocation of tasks, which will reduce overall system performance. The master node divides storage nodes into three queues based on the load information of each storage node. The three queues correspond to three load thresholds: light load, moderate load and overload. The queues are sorted by weight after partitioning. This algorithm implementation is complicated, and the master node handles a large number of requests from the client. Additionally, the master node must maintain the three queues in memory, which increases the load on the master node. As the system expands, the number of storage nodes increases and the queue length increases, resulting in increased memory overhead. In cases of large changes in load, the frequent changes in the elements in the queues will further increase the additional load on memory and CPU resources.

We propose an adaptive dynamic load balancing algorithm based on QoS to solve the problems discussed above. We build the a real distributed file system to verify the feasibility and efficiency of the new algorithm through a comparison experiment. The performance of the adaptive dynamic load balancing algorithm based on QoS and the traditional periodic dynamic load balancing algorithm are both tested. The experimental results show that the new algorithm can effectively reduce network overhead, reduce memory load on the master node, reduce the average response time of the system, and improve throughput. The major innovations of the proposed algorithm are listed below and a visualization of the proposed algorithm is presented in Fig. 1:

- We set a threshold value at the storage node, and send load information to the master node adaptively, based on changes in the degree of accessibility, rather than periodically updating the load information.

- We fully consider the various dynamic factors that affect the real-time load of the storage nodes, including CPU utilization, memory usage, disk IO occupancy rate, network bandwidth usage and hard disk usage, improve the accuracy of real-time load.

- Distributed file system has network heterogeneity. In order to account for the influence of network heterogeneity on the system, a comprehensive evaluation value for storage nodes is characterized by the performance load ratio. The new algorithm will calculate the normalization performance of the storage node, and the performance load ratio is calculated from the real-time load value and the normalized performance.

• The concept of QoS is introduced into the distributed file system. We use a tuple composed of the number of files and total file size of requested the task as a measure of QoS. The master node sets a threshold based on the QoS of the requested task to filter the set of storage nodes that meet the requirements of the task.

2 Related work

Load balancing is a higher-level load allocation strategy than load sharing. It must distribute the system load to each node, eliminate or avoid any load imbalance problems, and optimize the overall performance of the distributed file system. Load balancing algorithms can be divided into two categories[11]: static load balancing[12] and dynamic load balancing[9,13].

Static load balancing[14,15] is also known as state-independent balancing. It determines a load allocation strategy before a task is triggered, meaning the master node does not consider the real-time load status of each storage node while processing a request, but instead operates based on known system static information to make decisions and assign tasks. The advantages of static load balancing are that the logic is simple, the overhead is small, and a task request can be quickly allocated to each storage node. However, it does not consider the real-time load of the storage nodes or dynamic changes in the system state. The task assignments are made blindly and the accuracy is low, causing task allocation to be uneven and limiting the system load balance[10].

Dynamic load balancing[10] focuses on the state of information in the system, by analyzing the real-time load of each storage node, tasks are allocated to the storage nodes dynamically. The advantages of dynamic load balancing are that it can adjust the allocation of tasks in real time based on the load information of the storage nodes, adapt to changes in the load state of the system, and that it has excellent flexibility. However, it also has some disadvantages. The master node must periodically collect the status of the storage nodes necessitating, frequent information exchange between the master node and storage nodes to make network overhead, resulting in a waste of network bandwidth[10].

The weighted rotation scheduling algorithm[16] is an upgraded version of the round robin algorithm. The round robin algorithm assigns all tasks in turn to each storage node in the system. The round robin algorithm causes all work for nodes to be handled in a circular pattern. In other words, each node is fixed with a time slice and performs a task at designated time on its turn[17,18]. A scheduling and load balancing algorithm that considers the capabilities of each VM, the task length of each requested job, and the interdependency of multiple tasks was proposed[17]. As a result, some nodes may encounter heavy loads while others may have no task requests. This issue could be improved by using a weighted round robin algorithm, where each node can to possess a specific number of requests according to its assigned weight[19,20]. The weighted rotation scheduling algorithm sets different weighting factors for different storage nodes based on their processing ability. The assignment of tasks is based on the weights, and higher priorities are assigned to storage nodes with higher weight factors. The algorithm is more efficient when dealing with requests with smaller time spans, because the load becomes unbalanced when handling tasks with large time spans.

In the minimum connection scheduling algorithm[21], the master node of the distributed file system detects and records the current number of active connections of its storage nodes in real time. When a new request arrives, the master node assigns it to the storage node with the smallest number of active connections, and increments the number of active connections for that node by one. When the task is completed, the number of active connections is decremented by one. If the processing capacity of all the storage nodes in the distributed file system is the same, the minimum connection scheduling algorithm will distribute requests with large loads to each storage node in a balanced manner, which is more efficient. However, the real environment of a

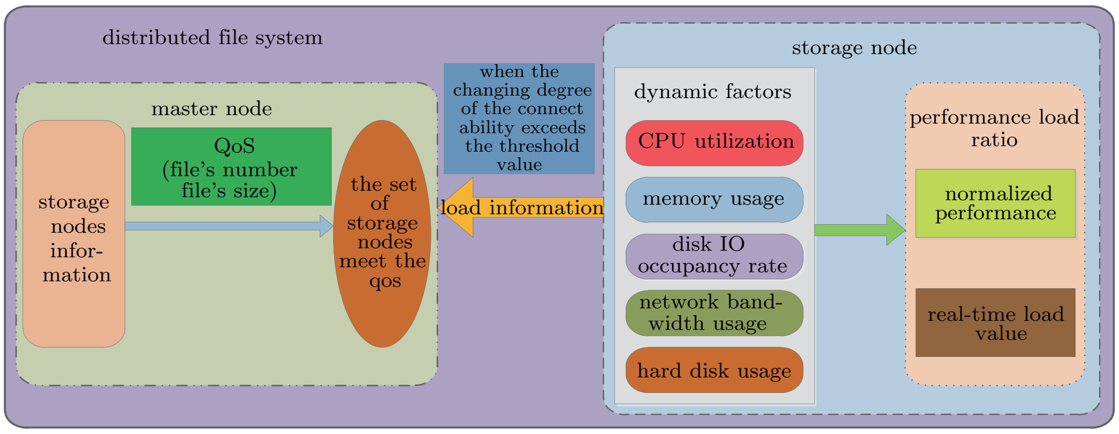

Figure 1 Visualization of the proposed algorithm

distributed file system is a heterogeneous network, where the processing capacity of each storage node is different. By allocating requests, solely on the basis of the current number of connections, high performance storage nodes will have more tasks and a shorter processing time; and storage nodes with low performance will receive fewer tasks, but the processing time for each task will be long, and the node will continue to accept new tasks.

The periodic dynamic feedback load balancing algorithm is a common load balancing algorithm in current distributed file systems. In this algorithm, storage nodes periodically send the real-time load information to the master node. The master node divides the storage nodes into light-load, moderate-load and heavy-load queues based on this load information, and then sorts the three queues by storage node weight. The master node then assigns new requests to storage nodes in the light-load queue in order of their weight values. However, this algorithm uses periodic updates to send load information to the master node. If the information exchange between the storage nodes and the master node is too frequent, it can lead to excessive message traffic and inefficient bandwidth usage. Additionally, the master node must maintain three queues in memory, increasing its memory burden.

To avoid excessive message traffic, several schemes were proposed[22,23]. A load balancing scheme with an information exchange policy based on a random sampling of packets for systems with a decentralized nature was proposed in order to improve system resource utilization while retaining a limited number of message exchanges [22]. Two message replication strategies were proposed to improve the efficiency and scalability of unstructured P2P networks while maintaining query performance[23]. A dynamic replica technique for cloud storage that utilizes acceleration was also studied to enhance reliability and access latency[24]. The acceleration response is based on the nature of speed trends, which can be used to predict file access variations. Using this information, this technique can identify hot spot data for the next period and determine the best node in which to establish a replica.

3 Adaptive dynamic feedback load balancing algorithm based on QoS

With the goal of solving the problems discussed above, we propose an adaptive dynamic feedback load balancing algorithm based on QoS by combining the advantages of multiple load balancing algorithms. The core concept of our algorithm is to allocate requests to storage nodes with small loads and high performance, to realize the maximum potential performance of the system while ensuring load balancing for each storage node. In the following sections, the proposed algorithm is described in detail from four viewpoints: load information collection strategy, load information processing strategy, load information update strategy, and task assignment strategy as follows.

3.1 Load information collection strategy

Through a multi-test of a distributed file system, it was found that load changes in the storage nodes have a strong correlation with changes in the number of active connections. The maximum number of active connections that a storage node can establish is limited, but the value is not equal to the number of threads available to the server and must be determined. When the system is running, the storage node real-time monitors the number of available connections, and defines the ratio of the number of available connections and the maximum number of active connections to the connect ability degree of the storage node.

The distributed file system is a heterogeneous network, meaning there are performance differences between storage nodes. To ensure that the master node assigns requests to storage nodes with low loads and high performance, the following performance-related dynamic factors must be considered: CPU utilization, memory usage, disk IO occupancy, and network bandwidth usage. To improve the overall resource utilization of the system, hard disk space usage is determined based on these dynamic factors and a tuple composed of the available connections of the storage node and the available storage space is used as a storage node service quality metric.

3.2 Load information processing strategy

Prior to starting the system, we obtain hardware configuration information such as CPU performance, memory capacity, maximum IO rate, network bandwidth and hard disk space for each storage node. Due to the different dimensions of various pieces of hardware configuration information, the data must be normalized. The maximum performance of each storage node is calculated from the hardware configuration information following the normalization process.

After performing real-time monitoring of the dynamic factors in the storage nodes, the connection ability of the nodes is calculated from the current number of available connections and the maximum number of active connections. According to a formula for all dynamic factors to calculate the real-time load value of the storage node, the performance load ratio of the storage node is calculated in combination with the real-time load value of the storage node and the maximum performance value normalized. The calculation of the connect ability degree and the performance load ratio is carried out at the storage node, which can effectively reduce the burden of the master node.

3.3 Load information update strategy

The new algorithm replaces the periodic update in the traditional algorithm with adaptive feedback based on the degree of connectivity. The connect ability degree of the storage node is dynamically changed during the operation of the system. When the change degree of the connect ability degree exceeds the threshold value, the storage node sends the performance load ratio as well as the service quality evaluation value to the master node. Reduce the frequency of the interaction between the storage node and the master node, and reduce the network overhead caused by the periodic update.

3.4 Task allocation strategy

The new algorithm introduces the concept of QoS, which is represented by a tuple consisting of the number of files to be uploaded in an upload task and the total size of the files. The master node selects the storage nodes that satisfy the requirements based on the QoS of the upload task and the quality of service evaluation value of each storage node. The master node then assigns the new task to the storage node with the largest performance load ratio from the set of selected nodes. The master node only needs to maintain this set of nodes in memory, reducing total memory overhead.

4 Determination and calculation of relevant parameters

4.1 The QoS of the request task and the service quality evaluation value

When the client initiates an upload task, the task request information is sent to the master node. The request information contains the number of files to be uploaded and the total size of the files, denoted $FNUM$ and $FSIZE$ respectively. The master node uses the tuple $(FNUM, FSIZE)$ to characterize the QoS of the requested task. The storage nodes monitor their number of available connections and available storage space. The number of available connections and the available storage space are denoted Ci and ASi, respectively. The tuple (Ci,ASi) is used as the service quality evaluation value for a storage node.

4.2 Connect ability degree

A storage node denoted Ni, where i falls in the range $[1{:}\,n]$ and n is the total number of storage nodes. The maximum number of available connections for storage nodes Ni is $CMAXi$, which is obtained through testing. During the operation of the system, the storage node monitors the number of available connections Ci in real time. It calculates $CMAXi$ and Ci to get ACi, as shown in the formula below. Through real-network testing, we determined that system performance is best when the threshold value V to take the golden ratio of 0.618.

$$ACi = \frac{Ci}{CMAXi}.$$

4.3 The real-time load value of the storage node

The storage node monitors the dynamic factors associated with the server load during system operation in real time. The adaptive dynamic load balancing algorithm based on QoS considers CPU utilization, memory usage, disk IO occupancy, network bandwidth usage, and hard disk usage as a dynamic factor after considering the characteristics of the actual distributed file system to calculate the real-time load value of the storage node Ni. Li is used to represent the real-time load value obtained by the following formula.

CPU utilization, memory usage, disk IO occupancy, network bandwidth usage, and hard disk usage are denoted $Rcpu$, $Rram$, Rio, $Rnet$ and Rhd respectively. $Wcpu$, $Wram$, Wio, $Wnet$ and Whd are the coefficients of each dynamic factor. The size of the coefficients represents the degree of influence the dynamic factors have on the load. Because a distributed file system is primarily used for file access, the main operation of the system is file reading and writing, which are IO-intensive tasks. Therefore, the requirement for hard disk IO performance is very high. There are many interactions between the memory and the hard disk during the process of file reading and writing, and memory performance plays an important role in the system. The storage node and the client must transfer files through the network, meaning network transmission rate is the key factor that affects transmission delay. Hard disk IO, memory usage, and network bandwidth all play a key role in the performance of the system. Thus, the coefficients of these dynamic factors are higher. We performed many experiments to boost the performance of the distributed system through parameter optimization. The resulting coefficients for each dynamic factor are: $Wcpu = 0.15$, $Wram = 0.25$, $Wio = 0.3$, $Wnet = 0.25$, $Whd = 0.05$.

$$Wcpu + Wram + Wio + Wnet + Whd = 1,$$

$$Li = (Wcpu, Wram, Wio, Wnet, Whd) * \begin{pmatrix} Rcpu \\ Rram \\ Rio \\ Rnet \\ Rhd \end{pmatrix}.$$

4.4 Maximum performance of a storage node

The performance of a storage node is determined by its hardware configuration. The proposed algorithm uses CPU performance, memory capacity, maximum IO rate, network bandwidth and hard disk space as the parameters for calculating maximum performance, denoted $Pcpu$, $Pram$, Pio, $Pnet$, Phd respectively. The values correspond to the dynamic factors used for the calculated load value. The maximum performance of the storage node is denoted $PMAXi$. The CPUs in the storage node are not unique, and the CPU frequencies and core numbers are not exactly the same. Thus, the product of CPU core count and the frequency is used as a single CPU performance value. The CPU performance of the storage node is the sum of all CPUs in the node.

Due to the different dimensions and units for each parameter, a direct calculation would result in $PMAXi$ being inaccurate, which would negatively affect the performance of the algorithm. In order to ensure the accuracy and reliability of the algorithm, the parameters must be normalized. The proposed algorithm uses the Z-score normalization method to normalize the parameters through the following formula:

$$X* = \frac{x-\mu}{\sigma},$$

Where x is the original data, $X*$ is the normalized data, is the mean value of all data, and is the standard deviation. $P*cpu$, $P*ram$, $P*io$, $P*net$, $P*hd$ were obtained by normalizing $Pcpu$, $Pram$, Pio, $Pnet$, Phd according to Z-score normalization method, the maximum performance of the storage node is calculated by the following formula.

$$PMAXi = (Wcpu, Wram, Wio, Wnet, Whd) * \begin{pmatrix} P*cpu \\ P*ram \\ P*io \\ P*net \\ P*hd \end{pmatrix}.$$

4.5 The performance load ratio of Storage node

The real-time load and maximum performance of a storage node have been obtained. The performance load ratio of a storage node is denoted $RPLi$ and is obtained using the following formula:

$$RPLi = \frac{PMAXii}{Li}.$$

4.6 Pseudo code implementation

```
Storage nodes
1.  while $True$
2.      Calculates Li, calculates $RPLi$ with $PMAXi$ and $Li$
3.      Collect quality of service evaluation $(Ci, ASi)$
4.      Calculate the $ACi$
5.      if change range of $ACi > 0.618$
6.          Send $(Ci, ASi)$ and $RPLi$ to the master node
7.      if get upload task
8.          $Ci$-1
9.          Connect with the client
10.         Processing upload task
```

```
Master node
1.  while $True$
2.      Gets $(Ci, ASi)$ and $RPLi$ of each storage node
3.      Sort all storage nodes by $RPLi$
4.      if Received client's upload request
5.          Get the QoS of upload task
6.          Filters out the set of storage nodes that meet
            the requirements according to QoS
7.          if set is not empty
8.              Select the storage node $i$ with largest
                $RPLi$ from the set
9.          else
10.             while set is empty
11.                 Wait 1 seconde
12.                 Filters out the set of storage nodes
13.             Select the storage node $i$ with largest
                $RPLi$ from the set
14.         Send the $IP$ and $Port\ Number$ of the node
            $i$ to the client
15.     else
16.         continue
```

5 Algorithm performance test and analysis

5.1 Test environment

In order to test the performance of the new algorithm, the actual distributed file system is set up

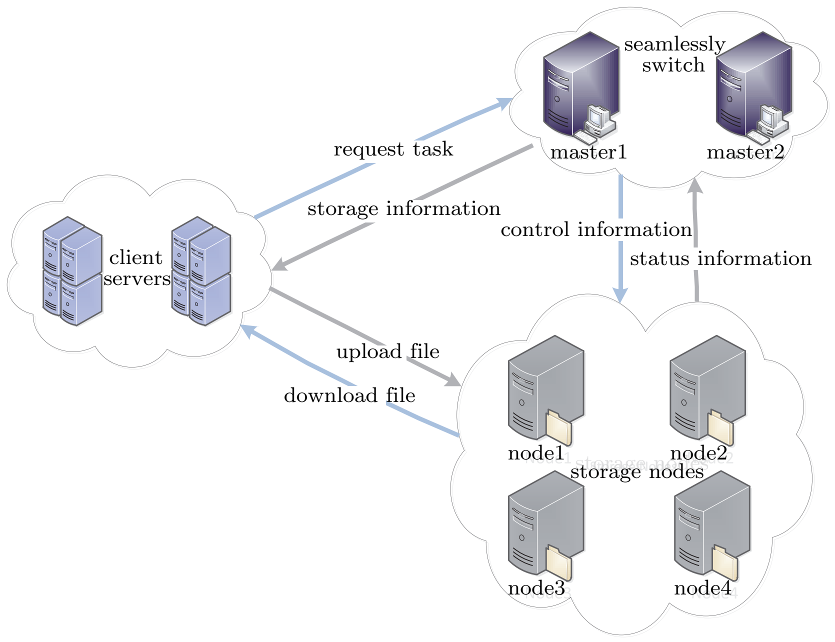

Figure 2 System architecture

as the test environment on the server cluster while system architecture shown in Fig. 2. Among which four servers are the storage nodes, select two servers as master node, one of the other can be seamlessly switched when one breakdown in order to ensure the reliability of the system. Select the other two as client test servers, through multiple processes to start multiple upload client, simulated multiple users randomly initiated upload request.

5.2 Performance testing and results analysis

In a distributed file system, the load balancing algorithm's test objectives include two aspects: on the one hand from the system point of view, the utilization of storage resources is the most important issue, which can through the system throughput and hard disk space utilization to express. On the other hand, from the top application point of view, the user is most concerned about the response time of the distributed file system to the request, which can be described by the average response time of the system.

The periodic dynamic feedback load balancing algorithm and the proposed adaptive dynamic load balancing algorithm based on QoS are applied to the distributed file system, and the comparison test is carried out.

Experiments in the use of multi-threaded technology to start a different number of upload services, simulation of multiple users randomly issued upload request, each "user" random upload 10 000 files, get the experimental results shown in Figs. 2∼4.

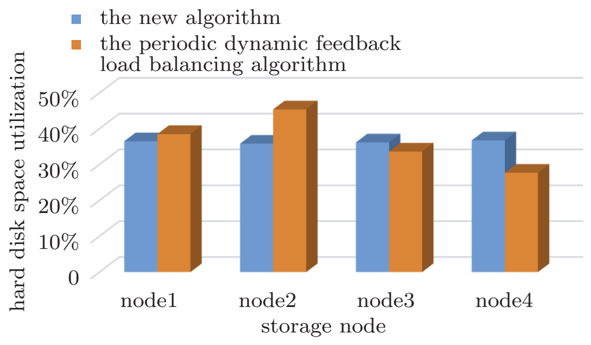

Figure 3 Hard disk utilization

The periodic dynamic feedback load balancing algorithm only considers the CPU utilization and memory utilization when computing the storage node load. In the new algorithm, the storage node considers the dynamic factor of the hard disk usage

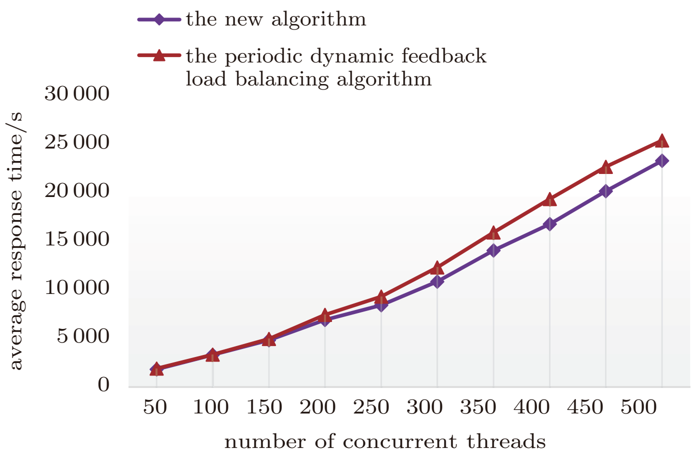

Figure 4 Response time

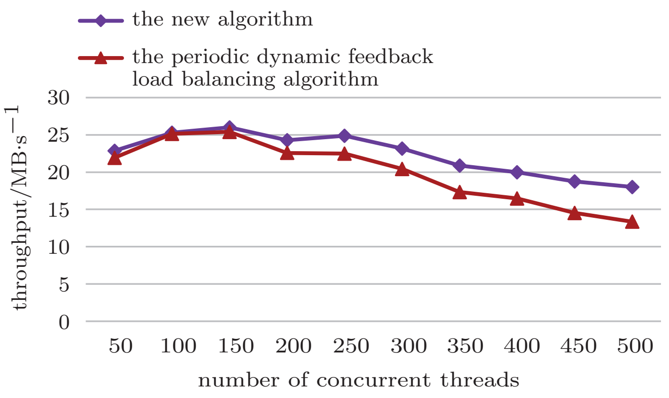

Figure 5 Throughput

when calculating the performance load ratio. The master node in the task allocation will inevitably be affected by the hard disk usage, so the results can be seen from the results of Fig. 3, the application of the new algorithm for distributed file system storage node hard disk load more balanced, hard disk utilization is higher.

From the results shown in Figs. 4 and 5, it is found that in the distributed file system to which the new algorithm is applied, the average response time is lower and the throughput is higher. Analysis of the reasons can see that the new algorithm improves the periodic feedback to the adaptive feedback based on the threshold, which reduces the network overhead. At the same time, the new algorithm fully considers the dynamic factors that can affect the performance of the distributed file system. The load condition of the storage node obtained by the master node is more accurate, the system load can be better allocated, overall performance of the system is improved. Through the experimental results in the two figures, we can find that the performance improvement is not obvious when the number of “users” is small. When the number of “users” increases, the advantage of the new algorithm is more and more obvious. This shows that when the load is low, the two algorithms can achieve the purpose of load balancing. When the load of the server is large, the new algorithm can allocate the system task better and allocate the system load to each storage node more reasonable, improve the overall performance of the system.

6 Conclusion

Load balancing problem is an important topic in distributed file system research. Aiming at the shortcomings of periodic dynamic load balancing algorithm, a new load balancing algorithm which named adaptive dynamic feedback load balancing algorithm based on QoS is proposed by analyzing various load balancing algorithms in distributed file system. Introducing the concept of QoS into distributed file system. The master node filters out the storage node set satisfying the requirements according to the QoS of the upload task and the quality of service evaluation value of each storage node, reduces the size of the storage node queue and reduces the memory cost. The load information of storage node is fed back to the master node by the adaptive feedback method based on the threshold, which not only realizes the real-time and reduces the network overhead. The new algorithm takes full account of the dynamic factors that affect the performance of distributed file system, and considers the performance difference of each storage node as well as network heterogeneity, performance load ratio is used to characterize the evaluation values of each storage node, improve the system throughput, reduce the average response time, so that the overall system performance is improved. In this algorithm, the threshold V is determined by the actual test in the current distributed file system. When the system is dynamically expanded, the threshold V needs to be adjusted. In the future, we can introduce the machine learning algorithm, so that the distributed file system can adaptively adjust according to the current state without human intervention, so as to improve the ease of use of the algorithm.

References

[1] E. Levy, A. Silberschatz. Distributed file systems: concepts and examples [J]. ACM computing surveys, 1990, 22(4): 321-374.

[2] T. Z. Zhao, S. B. Dong, M. Verdi, et al. Performance evaluation and relative predictive model of parallel file system [J]. Journal of software, 2011, 22(9): 2206-2221.

[3] M. Satyanarayanan. Distributed file systems [J]. Distributed systems. Addison-Wesley and ACM Press, 1993, 821: 145-154.

[4] S. Ghemawat, H. Gobioff, S. T. Leung. The Google file system [J]. ACM SIGOPS operating systems review, 2003, 37(5): 29-43.

[5] K. Shvachko, H. Kuang, S. Radia, et al. The hadoop distributed file system [C]//2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), IEEE, 2010: 1-10.

[6] X. Liu, Q. Yu, J. Liao. FastDFS: a high performance distributed file system [J]. ICIC express letters, Part B, 2014, 5(6): 1741-1746.

[7] R. Achar, P. S. Thilagam, N. Soans, et al. Load balancing in cloud based on live migration of virtual machines [C]//Annual IEEE India Conference (INDICON), IEEE, 2013: 1-5.

[8] D. R. Karger, M. Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems [C]//Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, ACM, 2004: 36-43.

[9] A. M. Alakeel. A guide to dynamic load balancing in distributed computer systems [J]. International journal of computer science and information security, 2010, 10(6): 153-160.

[10] S. Sharma, S. Singh, M. Sharma. Performance analysis of load balancing algorithms [J]. World academy of science, engineering and technology, 2008, 38(3): 269-272.

[11] T. L. Casavant, J. G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems [J]. IEEE transactions on software engineering, 1988, 14(2): 141-154.

[12] A. N. Tantawi, D. Towsley. Optimal static load balancing in distributed computer systems [J]. Journal of the ACM, 1985, 32(2): 445-465.

[13] J. Andonieh, A. Rahman. Dynamic feedback load balancing [P]. U.S. Patent Application 13/173,995, 2011-6-30.

[14] T. Kunz. The influence of different workload descriptions on a heuristic load balancing scheme [J]. IEEE transactions on software engineering, 1991, 17(7): 725-730.

[15] H. Rahmawan, Y. S. Gondokaryono. The simulation of static load balancing algorithms [C]//International Conference on Electrical Engineering and Informatics, IEEE, 2009, 2: 640-645.

[16] D. Grosu, A. T. Chronopoulos. A truthful mechanism for fair load balancing in distributed systems [C]//Second IEEE International Symposium on Network Computing and Applications, IEEE, 2003: 289-296.

[17] D. C. Devi, V. R. Uthariaraj. Load balancing in cloud computing environment using improved weighted round robin algorithm for nonpreemptive dependent tasks [J]. The scientific world journal, 2016, 2016: 3896065.

[18] A. Roy, D. Dutta. Dynamic load balancing: improve efficiency in cloud computing [J]. International journal of emerging research in management technology, 2013, 2(4): 78-82.

[19] N. S. Raghava, D. Singh. Comparative study on load

balancing techniques in cloud computing [J]. Open journal of mobile computing and cloud computing, 2014, 1(1): 18-25

[20] B. S. Rajeshwari. Comprehensive study on load balancing [J]. An international journal of advanced computer technology, 2014, 3(6): 900-907.

[21] X. Qin, H. Jiang, Y. Zhu, et al. A dynamic load balancing scheme for I/O-intensive applications in distributed systems [C]//International Conference on Parallel Processing Workshops, IEEE, 2003: 79-86.

[22] T. Alam, Z. Raza. Load balancing with random information exchanged based policy [C]//IEEE International Advance Computing Conference (IACC), IEEE, 2014: 690-695.

[23] O. A. H. Hassan, L. Ramaswamy. Message replication in unstructured peer-to-peer network [C]//International Conference on Collaborative Computing: Networking, Applications and Worksharing, IEEE, 2007: 337-344.

[24] M. X. Huang, X. L. Ye, S. P. Wei, et al. A strategy of dynamic replica creation in cloud storage [C]//1st International Workshop on Cloud Computing and Information Security. Atlantis Press, 2013.

About the authors

Ming Wang [corresponding author] received the B.E. degree in Communication engineering from Jilin University. He is currently working toward the master's degree in Beijing University of Posts and Telecommunications. His current research interests are in the areas of distributed systems, mobile Internet and future network. (Email: wangmingbupt@163.com)

Jianfeng Guan received his B.S. degree from Northeastern University. He received the Ph.D. degree in communications and information system from the Beijing Jiaotong University. He is an associate professor at Beijing University of Posts and Telecommunications. His main research interests focus around mobile Internet, network security and future network. (Email: jfguan@bupt.edu.cn)