Research on Load Balancing of Cloud Storage Server Based on Fully Connected Group

Xiaodan Chen*, Shuanglong Pang¹, Haiyan Wang², Desheng Zeng¹
¹Guangdong Innovative Technical College, Dongguan, Guangdong 523960, China
²Guangdong Preschool Normal College in Maoming, Maoming, Guangdong 525011, China

*Corresponding author: 120911912@qq.com

Abstract. The scheduling problem of online cloud storage traffic is a hot topic in current network research. Aiming at the scheduling problem of file upload requests in online cloud storage systems, the existing schemes rarely meet the requirement of providing users with different bandwidth guarantees. One of the purposes of using multiple volume servers is to let multiple volume servers share the system load evenly and avoid the system performance bottleneck. Once the load is aggregated and one of the multiple volume servers is overloaded, the volume server will become the system bottleneck again, which will greatly affect the overall performance of the system. On the basis of analyzing the existing scheduling algorithms of server clusters, this paper proposes a dormant scheduling algorithm of fully connected groups, which can automatically equip servers with I D, redirect tasks according to the actual processing capacity, and ensure load balance by unlimited increase. The experimental results show that the load balancing strategy among multi-volume servers ensures the overall reading and writing performance of the system, and effectively improves the I/O throughput and I/O times per second.

Keywords: Fully connected group; Cloud storage; load leveling

1. Introduction
Online cloud storage service is popular and developing rapidly because it provides convenient and reliable file management functions such as access, backup and sharing, and good user experience. More and more users use cloud disk to store data, and cloud storage has become one of the important applications of Internet and cloud computing. On this basis, the cloud computing system is composed of various computers, servers and data storage systems in the background, providing computing services and storage services for a large user group. With the further expansion of the cloud computing system, more and more data are stored in the cloud computing system, which makes the cloud computing system face great storage pressure [1-2]. Therefore, people urgently need a special storage system to store and manage huge data, so that cloud computing technology can be better developed.

The response speed of Web server and database server has repeatedly become a difficult problem hindering the development of network communication, and a large number of data flow requests make the database server must provide data processing services as soon as possible. Due to hardware and
operating system and other software reasons, when a large number of accesses arrive, the server cannot handle all requests in time, resulting in delay in response [3]. Although computer academia has proposed to use various algorithms to achieve load balancing and virtual server clustering to solve this problem, there are still many drawbacks, such as DNS load balancing technology can not distribute the load according to the processing capacity of Web servers, its reliability is not strong, and the high cost and energy consumption of cluster servers [4-5].

Based on this, we propose a new fully connected group sleep scheduling algorithm, which considers the scalability of the network and ensures the coverage and connectivity of the network. It adopts the clustering method of fully connected groups and distributed energy-saving node sleep scheduling strategy to reduce the network energy consumption as much as possible, thus prolonging the network life.

2. Load balancing of distributed file system
In order to ensure the efficient and reliable operation of distributed file system, it is necessary to avoid the bottleneck of system performance. It is found that performance bottlenecks easily occur when system resources are unevenly distributed and nodes fail. Therefore, for any distributed file system, besides realizing basic file operation functions, it is necessary to give a load balancing scheme suitable for this system [6]. The distributed file system based on cloud storage must have good scalability, and this scalability is achieved by increasing the number of nodes in the system. However, if there are bottlenecks in the system, the scalability of the system cannot be guaranteed at all.

The solution of load balancing is of great help to the improvement of system service capability, but there is no standard for the evaluation of load balancing, which is related to the angle of the evaluator. Especially when the attributes of system nodes are quite different, it is difficult for us to achieve real balancing, and even if it is implemented, it will easily bring other problems. Therefore, in the overall design of the system architecture, various factors should be considered comprehensively, so that each node can give full play to its role and achieve balance as much as possible.

In order to make multiple nodes in the system work together better, and to eliminate or avoid the bottlenecks of uneven load distribution, data traffic congestion and long system response time, many load balancing algorithms have been proposed, but there are two main categories [7-8]: static load balancing algorithm and dynamic load balancing algorithm. Static load balancing algorithms mainly include polling method, weighting method and priority method.

3. Sleep scheduling algorithm for fully connected groups
3.1. Fully connected group
The so-called fully connected group refers to a group where any two nodes in the group can reach each other with one hop. It is an ideal grouping method: firstly, the communication between any two nodes in a fully connected group only needs one hop, which greatly simplifies the complexity of intra-group routing; Secondly, the maintenance in the cluster is relatively simple. Once the cluster head fails, it can be directly replaced by any non-cluster head node with sufficient energy in the cluster without regrouping.

The difficulty in constructing a fully connected group lies in how to ensure that the nodes in the group can reach each other in one hop, that is, the full connectivity in the group. Considering that if the cluster leader has been selected, the connectivity between the cluster leader and the node can be ensured by arbitrarily selecting one of its neighbor nodes as a member node of the cluster. If the third node is selected as a group member based on the intersection of the neighboring nodes of these two nodes, the full connectivity among these three nodes can be guaranteed. And so on until the intersection of all the neighboring nodes of the group members is empty, so that the constructed group has full connectivity within the group. The following is a set expression for the construction of fully connected groups [9]:
In which $a_i, b_j$ is a sensor node; $s_0(a_i)$ is the set of all neighboring nodes of node $a_i$; $s_1(a_i)$ is the set of all member nodes of a fully connected group constructed with $a_i$ as the cluster head; $N$ is the number of nodes in $s_1(a_i)$.

3.2. Sleep scheduling

The primary problem that sleep scheduling needs to face is how to ensure the coverage and connectivity of the network. This problem will be discussed below on the assumption that the geographical location information of nodes in sensor networks is known. Firstly, several definitions are introduced:

(1) Detection range (disk for short): a circular area with the sensor node as the center and the node detection distance as the radius.

(2) Coverage: If a point is covered by the detection range of a certain sensor node, it is said that the point is covered; If any point in the area is covered, it is said that the area is covered (completely covered).

(3) Target area: the area to be monitored by the sensor network, which is denoted by $R$.

(4) Overlapping redundancy: we call the value obtained by subtracting $I_R(x)$ from the number of sensor nodes whose detection range can include point $x$ the overlapping redundancy at point $x$. Overlapping redundancy of a region refers to the sum of overlapping redundancy at all points in the region. The significance of defining overlapping redundancy is that it can show how many sensor nodes' detection ranges cover a certain point repeatedly.

3.3. Implementation of algorithm

First of all, the nodes interact with each other's neighbor information by exchanging "hello" messages (which contain the current neighbor node list of the node). After several interactions, the node can calculate the member nodes of the fully connected group constructed by itself for the cluster head according to the neighbor list information.

Secondly, a group invitation message "invite" is initiated by the group. The cluster leader should be taken by those nodes with large "fully connected neighbor set". To achieve this goal, the random waiting time before the node sends the "invite" message is set as: $\text{delay} = \left[(1 - n_0/n) + u\right] t_0$. In which $n_0$ is the number of member nodes in the group constructed with this node as the cluster head, $n$ is the number of neighbor nodes of this node, $u$ is the random number evenly distributed between $[0,1]$, and $t_0$ is the time required for one hop of message propagation.

Finally, when the node sends the "invite" message, a timer is set to record the response received within the valid time. When the timer expires, the node broadcasts a member node announcement message "men _ announcement" to the neighbor nodes, which contains information such as group head ID, group member ID, group number, and so on. At this time, the initial construction process of the fully connected group ends.

4. Server I D allocation mechanism

Server ID allocation mechanism, hereinafter referred to as mechanism. In this mechanism, the overload server is defined as the main server. When the main server is overloaded, the start up
mechanism determines the subordinate nodes it calls. According to the actual needs of the main server, call the first-level, second-level and even multi-level call nodes in turn to assist the main server in processing its traffic [10].

4.1. Overload of single node

When a single server node $N_0$ in the cluster is overloaded, the traditional idea of "copy transfer" is adopted, that is, the idle server node is selected, and the load of $F$ value is transferred to the server, thus realizing the function of load reduction.

4.2. Two-node overload

When the main server is in the first-level call, take $N_0$ and $N_1$ as the center, draw two circles crossing each other, and get two intersections except the center with radius $r$, then the ID of the first-level call server node is determined as $O_1R_1O_1$, and these three server nodes will provide processing capacity support for the corresponding main server at the same time.

Each first-level calling node serves two main servers at the same time. Assuming that the maximum load assigned by each main server to the lower-level nodes is $F$, the maximum load of calling nodes is guaranteed to be $2F$, thus ensuring the load balance of each node.

When the first-level call cannot meet the actual demand, the mechanism will start the second-level call. In the secondary call, draw a circle with three main servers as the center and $2r$ as the radius, and you will get four intersections. When the second-level call is not enough to meet the actual demand, the mechanism will continue to start the third-level or even higher-level call, and its node call process is the same as the second-level call principle. It is verified that the number of nodes called at each level is 4 when calling at or above the second level.

4.3. Three nodes overload

When the main server is in the first-level call, take $N_0, N_1, N_2$ as the center, draw three circles crossing the center, and get three intersections except the center with radius $r$, then the ID of the first-level call server node is determined as $O_1R_1O_1$, and these three server nodes will provide processing capacity support for the corresponding main server at the same time.

Each first-level calling node serves two main servers at the same time. Assuming that the maximum load assigned by each main server to the lower-level nodes is $F$, the maximum load of calling nodes is guaranteed to be $2F$, thus ensuring the load balance of each node.

When the second-level call is not enough to meet the actual demand, the mechanism will continue to start the third-level or even higher-level call, and its node call process is the same as the second-level call principle. It is verified that the number of nodes called at each level is 6 when calling at or above the second level.

5. Decentralized scheduling mechanism

Although the maximum weight scheduling mechanism can maximize the system throughput and keep the system stable, it has the following limitations and is difficult to be applied in real systems.

First, assume preemptive allocation. In the actual system, the request can not be interrupted, or it is very expensive to restore resources after interruption. Therefore, it is necessary to design a scheduling algorithm of nonpreemptive, so that the request is assigned to the server for processing until the task is completed. Secondly, it is assumed that there is a centralized scheduler in the system. A single centralized scheduler is easy to cause a single point of failure. In addition, online storage systems need hundreds of servers, and a single scheduler will make the system difficult to expand. Therefore, referring to Dropbox online storage system, multiple independent distributed load balancers are
needed.

Firstly, centralized scheduler and centralized queue are no longer set in the system. Each server \( i \) has a separate cache queue for each type of request, and its queue length is expressed as \( Q_{im}(t) \).

Secondly, when each upload request comes, the request is randomly sent to a load balancer for processing according to the uniform probability distribution. At the start time of each time slice \( t \), each load balancer randomly samples two block servers \( i_1 \) and \( i_2 \) for \( m \)-type requests according to uniform probability distribution to obtain queue length information \( Q_{1,m}(t) \) and \( Q_{2,m}(t) \) of the servers \( i_1 \) and \( i_2 \), and then sends the \( m \)-type requests on the load balancer to the servers with smaller \( m \)-type queues.

Finally, the server \( i \) schedules available resources in a non-preemptive manner. When the upload request is completed, the occupied resources are recycled by the server \( i \) as available resources. At the beginning of each time slice, the server \( i \) calls the available resources according to the maximum weight mechanism, that is,

\[
N_i(t) \in \arg \max_{N_i,N_i' \in \mathbb{N}} \sum_{m} Q_{im}(t)N_{im}
\]

In the formula (2), \( N_i(t) \) is the request that the time slice \( t \) on the server \( i \) has run at the start time and has not yet ended.

6. Experimental result

6.1. Introduction of experimental tools

Iometer is a testing tool used to measure the I/O performance of disk or file system. It is both a load generator and a performance tester. It can simulate various types of loads according to requirements and test the performance of the system in the corresponding environment. Iometer is also suitable for testing the performance of multi-core servers or multiple servers working in parallel in a cluster, which is very powerful.

Iometer can generate different types of loads by customizing four parameters: block size, read-write ratio, random ratio and waiting queue size, and record three IO performance indicators of the tested partition in terms of I/O times per second, throughput and response time under such loads.

6.2. Experimental process

Experiment 1: Do not start the dynamic load balancing strategy

1) Initialization: four guest virtual machines request to use nine logical volumes established in the storage array through two volume servers with different performances.

2) In order to simulate the unbalanced load state, in the experiment, after establishing multiple iSCSI connections with Orthrus system only with static load balancing strategy, unload multiple logical volumes on server A with large bandwidth, so that the system is in the unbalanced load state (at this time, the initial load distribution string of sleep scheduling algorithm of fully connected group is 11111100), and the iSCSI connection deployment process is completed.

3) Use iometer tool to generate loads (8k, 50%read, 50%random) with the same type and size for the remote logical volumes mounted on each guest virtual machine, and test the relationship between the system's read/write times per second ((IOPS)), total read/write throughput, average response time and the number of waiting requests in the queue (OIOs) under these loads.

Experiment 2: Start the load balancing strategy

1) Initialization: four guest virtual machines request to use nine logical volumes established in the storage array through two volume servers with different performances.
2) In order to simulate the unbalanced load state, seven connections were established on the server A with smaller bandwidth and two connections were established on the server B with larger bandwidth in the experiment, and the iSCSI connection deployment process was completed.

3) Use the iometer tool to generate loads (8k, 50% read, 50% random) with the same type and size for the remote logical volumes mounted on each guest virtual machine.

4) Start the dynamic load balancing strategy, find the optimal load distribution mode (the optimal load distribution string is 010110010 through the simulation of sleep scheduling algorithm of fully connected group), and start the load migration mechanism to realize load balancing.

5) Test the relationship between the system's read/write times per second (IOPS), total read/write throughput, average response time and waiting requests in queue (OIOs) under these loads.

6.3. Experimental results and analysis
The experimental results are shown in fig. 1 to fig. 3.

![Figure 1](image1.png) **Figure 1** Comparison of I/O times per second before and after load balancing

![Figure 2](image2.png) **Figure 2** Comparison of total read-write throughput before and after load balancing

![Figure 3](image3.png) **Figure 3** Comparison of system average I/O response time before and after load balancing

Through the above three sets of experimental results, it can be seen that the number of I/O times
per line second and the total read-write throughput rate of the system have been greatly improved after Orthrus system adopts dynamic load balancing based on machine performance, especially the throughput rate has reached about 1.8 times before, which shows that Orthrus has better read-write performance under the condition of load balancing of various volume servers. However, the average I/O response time of the system has not improved, which shows that when the number of guest virtual machines in the whole system is small, the main factor affecting the average I/O response time of the system is the number of I/O requests waiting in line, not the performance of the volume server.

7. Conclusion
Although centralized scheduling mechanism can maximize system throughput, it has many limitations. Therefore, a distributed quadratic random selection algorithm is proposed to avoid the two bottlenecks of centralized scheduling: single point failure and global information maintenance. On the basis of systematic analysis of existing load balancing algorithms, this paper proposes a sleep scheduling algorithm for fully connected clusters. When the nodes in the cluster are overloaded, the cloud storage server load balancing algorithm is started. Through mechanism, the server cluster is redistributed, the server ID is automatically configured, indicating the affiliation between servers, and the corresponding call level is started according to the need, finally, the traffic exceeding the processing capacity of nodes is reasonably distributed to its subordinate call server cluster. Theoretical analysis and experimental results show that the load balancing strategy proposed in this chapter effectively solves the single-point performance bottleneck problem of multi-volume server block-level cloud storage system and improves the overall I/O performance of the system.

Acknowledgements
In 2018, the key scientific research platform and scientific research project of colleges and universities in Guangdong Province "Research on User-oriented Software Defined Storage Resource Allocation Strategy" (project number: 2018 GQNCX 111);
In 2020, the young innovative talents project of colleges and universities in Guangdong Province "Application Research of SDN Architecture in Data Center Network" (ProjectNo.: 2020 KQ NCX 258);
In 2019, the characteristic innovation project of colleges and universities in Guangdong Province "Research and application of cluster resource scheduling technology based on Kubernetes" (project number: 2019 GKTSCX173);

References
[1] Li Qiang, Liu Xiaofeng. Load balancing strategy of cloud storage based on Hopfield neural network. Computer Applications, vol. 037, no. 008, pp. 2214-2217, 2017.
[2] Ma A, Sn B, Ta A. Meta heuristic-based task deployment mechanism for load balancing in IaaS cloud. Journal of Network and Computer Applications, 2019, 128:64-77.
[3] Ala'Anzy M, Othman M. Load Balancing and Server Consolidation in Cloud Computing Environments: A Meta-Study. IEEE Access, 2019, 7:141868-141887.
[4] Shao, Xun, Jibiki, et al. An efficient load-balancing mechanism for heterogeneous range-queriable cloud storage. Future generations computer systems: FGCS, vol. 78, no. 3, pp. 920-930, 2018.
[5] Majhi S K, Pal S S, Bhuyan S, et al. Queuing Analysis of Cloud Load Balancing Algorithms. International Journal of Knowledge-Based Organizations (IJKBO), vol. 8, no. 1, pp. 50-67, 2018.
[6] Wang Limei. Research and preliminary implementation of cloud storage based on cheap computer cluster. Information recording materials, vol. 019, no. 003, pp. 74-75, 2018.
[7] Ebadifard F, BabaMir S M. A PSO-based task scheduling algorithm improved using a load-balancing technique for the cloud computing environment. Concurrency and Computation: Practice and Experience, vol. 30no. 1, pp. 4368, 2018.
[8] Shwe T, Aritsugi M. Avoiding Performance Impacts by Re-Replication Workload Shifting in HDFS Based Cloud Storage. IEICE Transactions on Information and Systems, vol. 101, no. 12, pp.
2958-2967, 2018.

[9] Udhayakumar U, Murugaboopathi G. To improve user key security and cloud user region-based resource scheduler using rail fence region-based load balancing algorithm. Journal of Ambient Intelligence and Humanized Computing, no. 6, pp. 1-8, 2020.

[10] Mahapatra B, Turuk A K, Panda S K, et al. Utilization-aware VB migration strategy for inter-BBU load balancing in 5G cloud radio access networks. Computer Networks, 2020, 181:107507.