International Journal of Grid Distribution Computing
Vol.7, No.4 (2014), pp. 169-176
http://dx.doi.org/10.14257/ijgdc.2014.7.4.16

Analyzing and Improving Load Balancing Algorithm of MooseFS

Zhang Baojun[1], Pan Ruifang[1] and Ye Fujun[2]

1. New Media Institute, Zhejiang University of Media and Communications, Hangzhou 310018, China
2. Library, Zhejiang University of Media and Communications, Hangzhou 310018, China
zbjhover@zju.edu.cn

Abstract

Cloud storage is the development direction of data storage nowadays, and MooseFS provides a good solution for cloud storage. But as a kind of typical distributed file system, the problem of load balancing among chunk servers affects the use of MooseFS. Though MooseFS has provided certain load balancing capability, but it takes into account only the consumed space, and which causes the chunk servers with larger space were overburdened. In this study, we analyzed the load balancing algorithm among chunk servers in MooseFS, and proposed an improved load balancing algorithm. Through the experiment and comparative analysis, the improved algorithm enhances the load balancing performance of MooseFS obviously.

Keywords: *Cloud Storage, MooseFS, Distributed File System, Load Balancing*

1. Introduction

As the massive growth of network data, humans entered the age of big data. The data storage is the prime problem in the age of big data, and the traditional file system is unable to meet the storage requirement of the age. The cloud storage based on the distributed file system provides effective solution for the storage of the age of big data [1, 2].

At present, the representative distributed file systems include Google's GFS [3], the open source file system such as Lustre [4, 5], HDFS [6, 7], MooseFS [8] and the CarrierFS developed by Tsinghua University [9,10]. All file systems can be used for large scale cloud storage. Among them, GFS is a proprietary distributed file system developed by Google for its own use [11]. HDFS is designed to reliably store very large files across machines in a large cluster [12]. It is inspired by the GFS. CarrierFS is designed to provide file sharing service for Tsinghua University. It has stored hundreds TB of data now. MooseFS is a distributed file system developed by Gemius SA. It aims to be fault-tolerant, scalable general-purpose file system for data centers. MooseFS is widely used because its source opened, general purpose, online capacity expansion, and easy to deploy.

MooseFS is a universal distributed file system, and its system structure contains four parts, the master server, the metalogger server, the chunk server and the client respectively. In MooseFS, files are divided into chunks with the same size, and stored on different chunk servers. Since MooseFS is fault-tolerant, it enables user to indicate the number of copies of files to provide the data redundancy. These copies can be replicated across chunk servers. So there are many chunks on different chunk servers, how to allocate chunk server to chunk is a problem of load balancing.

ISSN: 2005-4262 IJGDC

Load balancing of data is the core function of a distributed file system, and it determines the performance of the system directly. The main purpose of load balancing is to distribute workloads across multiple computing resources, such as computers, a computer cluster, network links, central processing units or disk drives [13]. It can avoid a node crash for over loading. For distributed file system, the load balancing focuses on the allocation of storage resources. Currently, there are two ways to implement the load balancing, the static mode and the dynamic mode [14-16]. The static mode is to set the load balancing strategy before a system running, once configured, it could not be changed when the system running. The general static load balancing algorithms includes the round robin, the ratio and the priority [17]. On the contrary, the dynamic mode can adjust the load balancing strategy dynamically according to the loading of nodes when the system running. Apparently, the dynamic mode is more complex than the static mode, but it can achieve better results than the static mode. The load balancing algorithm of MooseFS is dynamic.

MooseFS has provided a load balancing mechanism, but the current algorithm seems to take into account only the consumed space. It does not think of the access load of chunk servers. In our study, we analyzed the load balancing algorithm of MooseFS, indicated its problem, and proposed an improved method. The paper is organized as follows: in the second section, load balancing of MooseFS is analyzed. Section three introduces our improved algorithm. The last section is the conclusion.

2. Moosefs Load Balancing Algorithm Analyzing

MooseFS is a source opened, fault-tolerant, universal and online capacity expansion distributed file system. It is easy to deploy, so is used widely. The newest stable release of MooseFS is 1.6.27 released on March 24, 2013. We choose this release version as our object of study.

2.1. MooseFS System Structure

MooseFS 1.6.27 comprises four components, the master server, the metalogger server, the chunk server and the client.

1) Master server: Manages metadata of file system, such as namespace hierarchy, access control, the mapping information between file and chunk, and the location of current chunk. The current version of MooseFS does not support multiple master server, so when its load is overweight, it may appear the single point of failure. Clients only talk to the master server to retrieve or update a file's layout and attributes. The data itself is transferred directly between clients and chunk servers. The metadata is kept in memory and lazily stored on local disk.

2) Metalogger server: Only occurred in release 1.6 and later. It is responsible for replicating the log files of the master server so that taking over the work when the master server failed.

3) Chunk servers: They are where the data stored. They store the data and optionally replicate it among themselves. There can be many chunk servers, and now the scalability limit has not been reported. The biggest cluster reported so far consists of 75 servers [18]. The Chunk server is a user-space daemon that relies on the underlying local file system to manage the actual storage.

4) Clients: Talk to both the Master server and Chunk servers. MooseFS clients mount the file system into user-space via “ FUSE” .

In a word, in MooseFS, the Master server manages metadata information, the metalogger server backups the master server, the chunk servers store data and the clients is the end-user.

2.2. Load Balancing of Chunk Server

Since files are stored on multiple chunk servers, in order to achieve high reliability and performance MooseFS offers the feature of load balancing.

In MooseFS, when client uploads file to chunk servers, the file will be divided into chunks. Client connects to the master server, the master server records the chunk information of the file. Then, the master server will allocate chunk servers to the file chunks according to chunk server selection algorithm, and that is the load balancing algorithm. The load balancing algorithm needs to allocate chunk servers to the chunks as appropriate as possible. Specifically, it should allocate the storage space evenly and avoid some chunk servers overburdened.

According to MooseFS 1.6.27, the load balancing among chunk servers is implemented on master server, and corresponding codes are in file "matocsserv.c". MooseFS uses data structure "matocsserventry" defined in "matocsserv.c" to record the information of chunk server. In the structure, there is a variety "carry" which is the key to judge the priority of chunk server when allocated to chunks. Each chunk server has a variety "carry", Master server will sort the chunk servers from largest to smallest according to "carry". The chunk server with the largest "carry" will be selected to the chunk needs to be stored. The flow chart of load balancing of chunk server is showed in Figure 1.

In Figure 1, the means of identifiers are as follows:

"demand" - The number of copies of current file.

"allcnt" - The number of chunk servers in MooseFS.

"availcnt" - The number of available chunk servers in MooseFS, these available chunk servers can be used to store data directly. "availcnt" should be smaller than "allcnt".

"weight" - This is a very important variety used to calculate the carry of chunk servers. In MooseFS, "weight" is calculated with Formula 1.

$$weight = localtotalspace / maxspace \quad \text{(Formula 1)}$$

"localtotalspace" - The total space of local chunk server.

"maxspace" - The largest space of chunk servers.

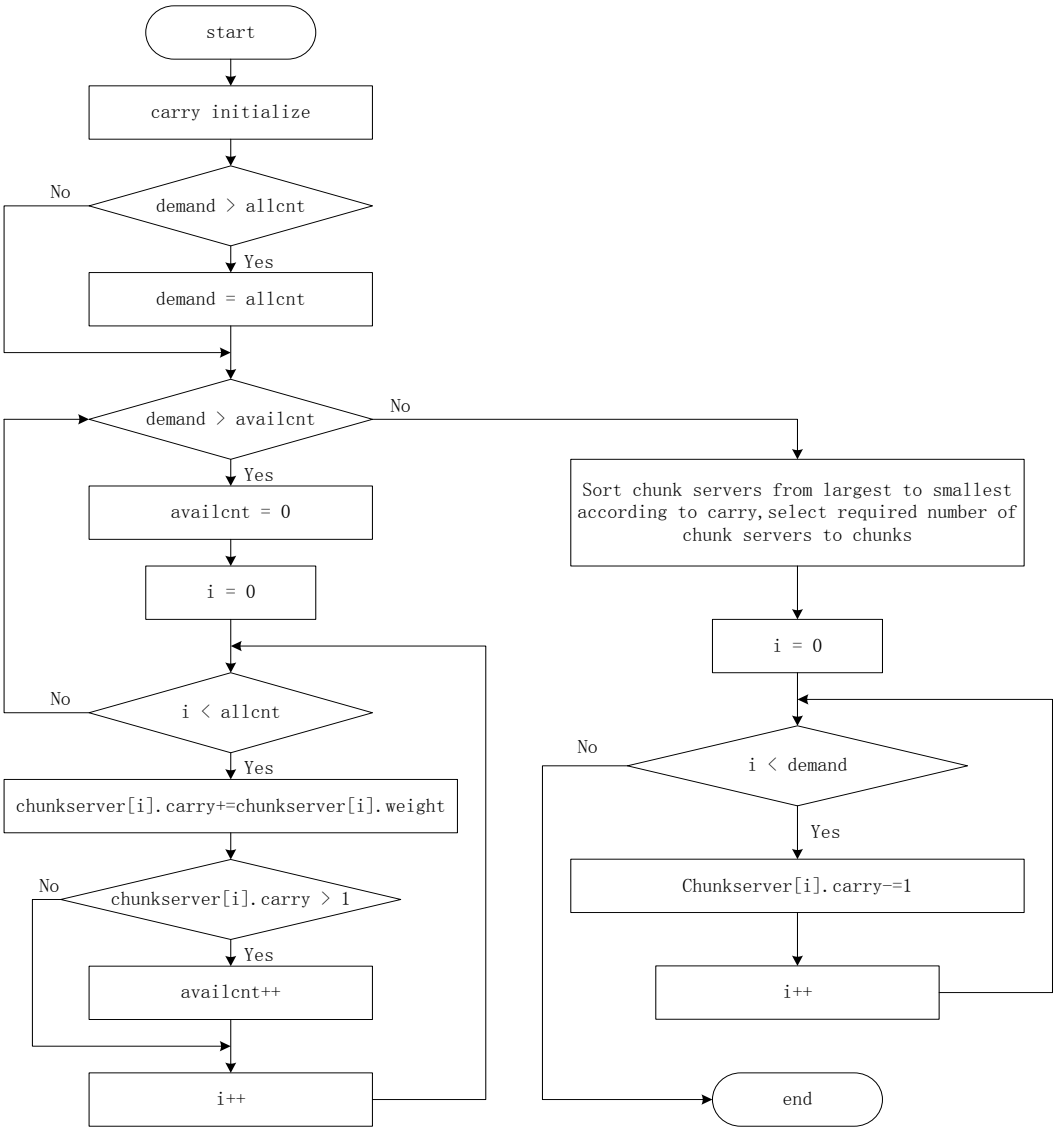

Figure 1. Flow Chart of Load Balancing in MooseFS

According to Figure 1, the flow of load balancing of chunk server is as follows.

Step 1: Initializa “carry” for each chunk server, it is a random number in region (0, 1).

Step 2: Compare “demand” with “allcnt”, judge whether the number of backups of file is bigger than the number of chunk servers or not. It can’t be bigger than the number of chunk servers.

Step 3: Compare “demand” with “availcnt”, judge whether the number of backups of file is bigger than the number of available chunk servers or not. If bigger, it means that the number of available chunk servers is not satisfy the requirement of backups currently, than increases the number of “carry” for each chunk server circularly until the number of chunk servers whose “carry” is bigger than 1 equals “availcnt”. The increment is variety “weight” above. Else, enter step 4 directly.

Step 4: When the flow path enters here, it means the available chunk servers satisfy the storage requirement of files. Sort all chunk servers from biggest to smallest according to the value of "carry", the number with the value "demand" of chunk servers ahead will be allocated for data storage.

Step 5: Reset the value of carry. Subtract 1 the value of the carry of the chunk servers selected in Step 4.

The load balancing algorithm of MooseFS is dynamic, and the key of the algorithm is the increment "weight". The "weight" is the ratio between the total space of the local chunk server and the total space of the chunk server with the largest storage space. We can see that the load balancing algorithm of MooseFS takes the total space of chunk servers into account. The chunk servers with a large space will have a big "weight", and the "carry" of the chunk servers increases quickly, accordingly, the "carry" of the chunk servers will exceed digital 1 more quickly and then be selected for data storage.

Here we test the load balancing of MooseFS. In order to describe the testing procedure easily, we use the number of chunks to measure the size of files and the space of chunk servers. In our testing, we suppose four chunk servers with the total space 10000 chunks, 20000 chunks, 30000 chunks, 50000 chunks respectively. We suppose the initial "carry" for all chunk servers is 0.3, 0.5, 0.2, 0.6 and the "demand" is 2. According to the load balancing algorithm, the "weight" of chunk servers is showed in Table 1.

Table 1. Testing Data

Chunk Server	Chunk Server1	Chunk Server2	Chunk Server3	Chunk Server4
Total Space	10000	20000	30000	50000
Initial carry	0.3	0.5	0.2	0.6
weight	0.2	0.4	0.6	1

We suppose 500, 1000 and 2000 chunks will be stored on these chunk servers, since the number of backups is 2, there are 1000, 2000, 4000 chunks respectively. Table 2 shows the result when all the chunks are stored.

Table 2. Chunk Distribution with Load Balancing of MooseFS

Chunk Server / **Chunk Number**	Chunk Server1	Chunk Server2	Chunk Server3	Chunk Server4
1000	74	175	275	476
2000	150	350	550	950
4000	300	700	1100	1900

From Table 2, the chunks are allocated to each chunk servers unequally. The algorithm takes into account only the consumed space. It causes the chunk servers with larger space

were accessed frequently. Investigate its reason the value of “weight” is fixed. That is to say the increment of the “carry” of chunk servers is fixed. So the chunk server with a larger space will be selected more frequently though its idle space is small as the system running. In order to acquire a good access load balancing, we should not take the used space of chunk servers into account.

3. Improved Moosefs Load Balancing Algorithm

In section 2, we analyzed the load balancing algorithm of MooseFS and indicated its shortness. Here we proposed an improved load balancing algorithm for MooseFS which takes into account the access balancing of the chunk servers. In our method, we calculate the value of “weight” with the Formula 2.

weight = localidlespace / maxidlespace (Formula 2)

“localidlespace” - The idle space of local chunk server.

“maxidlespace” - The largest idle space of chunk servers.

Table 3 shows the test result under the same conditions as above.

Table 3. Chunk Distribution with our Improved Load Balancing

Chunk Server / Chunk Number	Chunk Server1	Chunk Server2	Chunk Server3	Chunk Server4
1000	250	250	250	250
2000	500	500	500	500
4000	1000	1000	1000	1000

According to Table 3, the chunks are allocated to each chunk servers averagely. The access loadings of chunk servers are balanced. With our algorithm, the chunk servers with smaller storage space will be exhausted quickly, but it is no matter. We can set those chunk servers without enough space as unavailable, and use the available chunk servers.

4. Conclusion

The load balancing algorithm of MooseFS takes into account only the consumed space. It results in the chunk servers with larger space were overburdened. This study analyzed the load balancing algorithm of MooseFS, and proposed an improved method. Experiment result proved that the improved algorithm solves the problem exist in the old one, and makes the access load of chunk servers balance.

References

[1] M. Carroll, P. Kotzé and A. van der Merwe, “Securing Virtual and Cloud Environments [M]”, Cloud Computing and Services Science, Springer New York, **(2012)**, pp. 73-90.
[2] Cloud Storage [EB/OL]. http://en.wikipedia.org/wiki/Cloud_storage, **(2013).**
[3] S. Ghemawat, H. Gobioff and S. T. Leung, “The Google File System [C]”, ACM SIGOPS Operating Systems Review, vol. 5, no. 37, **(2003)**, pp. 29-43.

[4] J. Peter and Lustre, "The Inter-Galactic File System. Presentation slides", Lawrence Livermore National Laboratory, Retrieved, **(2013)** September 23.
[5] Index of /public/lustre/latest-maintenance-release [EB/OL]. Download from commercial site, http://downloads.whamcloud.com/public/lustre/latest-maintenance-release/ **(2013).**
[6] D. Borthakur, "The Hadoop Distributed File System: Architecture and Design [J]", The Apache Software Foundation **(2007).**
[7] Hbase Development Team, Hbase: Bigtable-Like Structured Storage for Hadoop Hdfs [EB/OL], http://wiki.apache.org/hadoop/Hbase **(2011).**
[8] Moose File System [EB/OL]. http://en.wikipedia.org/wiki/Moose_File_System **(2013).**
[9] C. C. Xu, X. M. Huang, P. Z. Xu, N. Wu, S. Liu and G. W. Yang, "CarrierFS: a virtual memory based distributed file system [J]", Journal Huazhong University of science and technology (Natural science edition), vol. 6, no. 38**, (2010)**, pp. 37-42.
[10] C. C. Xu, X. M. Huang, N. Wu, N. W. Sun and G. W. Yang, "Performance testing and analysis to storage medium of distributed file system [J]", Chinese journal of computers, vol. 10, no. 33, **(2010)**, pp. 1873-1880
[11] GOogle File System [EB/OL], http://en.wikipedia.org/wiki/Google_File_System, **(2013).**
[12] Hadoop Distributed File System [EB/OL], http://wiki.apache.org/hadoop/HDFS, **(2010).**
[13] Load balancing (computing) [EB/ol], http://en.wikipedia.org/wiki/Load_balancing_(computing), **(2014)**
[14] J. Jiang, M. Zhang and X. Liao, "Study of Load Balancing Algorithms Based on Multiple Resources [J]", Acta Electronica Sinica, vol. 8, no. 30, **(2002)**, pp. 1148-1152.
[15] Z. P. Tan, "Researches of replication management on object storage system [D]", Ph.D. thesis of Huazhong University of science and technology, **(2008).**
[16] C. P. Zhang and J. W. Yin, "Dynamic Load Balancing Algorithm of Distributed File System [J]", Journal of Chinese computer systems, vol. 7, no. 32, **(2011)**, pp. 1424-1426.
[17] The guide of load balancing [EB/OL]. http://cisco.chinaitlab.com/case/12415.html, **(2003).**
[18] M. Gądarowski, "MooseFS: Bezpieczny I Rozproszony System Plików", Linux Magazine Poland, **(2010).**

Authors

Baojun Zhang, he was born in 1977. Doctor of computer science and technology of Zhejiang University. Lecturer of Zhejiang University of Media and Communications. Research on computer network and information security for many years.

Ruifang Pan, she was born in 1959. Professor of Zhejiang University of Media and Communications. Dean of New Media Institute. Research on digital media, database and network.

Fujun Ye, she was born in 1975. Associate professor of Zhejiang University of Media and Communications. Associate director of library of Zhejiang University of Media and Communications. Research on digital media, database and network.