International Journal of Database Theory and Application
Vol.8, No.1 (2015), pp.105-110
http://dx.doi.org/10.14257/ijdta.2015.8.1.12

Scalable Multiple NameNodes Hadoop Cloud Storage System

Kun Bi[1] and Dezhi Han[1,2]
[1]*College of Information Engineering, Shanghai Maritime University, Shanghai 201306, China*
[2]*School of Computer Science and Engineering, South China University of Technology, Guangzhou 510632, China*
kunbi@shmtu.edu.cn

Abstract

To solve the problem that the total number of files in HDFS is limited by the memory size of the NameNode, the paper proposed a scalable multiple NameNodes coexisting scheme for cloud storage. NameNode location service (NLS) was introduced into the system. NLS communicated with all NameNodes and was responsible for the file location requests from the clients. Experimental results showed this scheme had good scale-out scalability and the upper limit of the total number of files that the system was able to store could be greatly increased by adding a number of NameNodes.

Keywords: *Multiple NameNode, Hadoop, HDFS, Cloud Storage System*

1. Introduction

Currently, the total amount of data has an explosive growth and mass data storage becomes a hot research topic. Data is generated from various fields, and the data types include structured data, semi-structured data and unstructured data. IDC (International Data Corporation) estimated that in 2020 the total amount of global data would reach 35ZB. A lot of work has been done to solve this problem and several kinds of distributed storage system were developed, such as Google GFS [1], Hadoop HDFS [2], OpenStack Swift [3], Amazon Dynamo [4] and Facebook Haystack [5], *etc*.

In this paper, a distributed file storage system with multiple NameNodes based on Hadoop HDFS was presented. It had a good scale-out performance by deploying multiple NameNodes in the system, and solved the problem that the total amount of files stored in HDFS was limited by the memory volume of single NameNode. This method is applicable to provide cloud storage service for a large number of users.

2. Related Work

One of the key problems in mass data storage is to efficiently locate and access data [6-11]. As the data volume continues to grow rapidly, traditional storage systems such as NAS and SAN have scalability limitations when storing such huge data volume. The use of inexpensive computers and storage devices to build a distributed storage system is a good solution. Most of those solutions adopt the methodology of separating the storage and management of metadata and the file content [1-11]. The metadata management in a distributed storage system can be divided into two categories, centralized management and distributed management. Google GFS [1] and Hadoop HDFS [2] adopt centralized metadata management solution, and OpenStack Swift [3] adopts distributed metadata management solution.

In GFS, the metadata are all stored in master node and data objects are distributed stored in chunk servers. The GFS client gets a file's metadata information from the master node and then access the chunk server to read or write the file content.

Hadoop HDFS is an open source implementation version of Google GFS. The HDFS NameNode is similar with GFS master node and HDFS DataNode is similar with GFS chunk server. OpenStack Swift uses distributed metadata management method and metadata are distributed stored among all the nodes. Swift uses DHT to organize all the metadata into a RING structure. Swift does not support multilevel folders, and all the data objects are stored in containers. It is suitable for storing massive amounts of <Key, Value> types of data objects and the data object can be located by searching the corresponding object id in RING. HDFS supports multilevel folders and all the files' metadata are stored in HDFS NameNode. It is convenient for file access according to the directory, but the total number of files that can be stored in HDFS is restricted by the total memory of NameNode.

HDFS system mainly consists of three parts: the NameNode, DataNode and Client. NameNode stores all the files' metadata, file directories organization, data blocks index and the mapping relations between data blocks and DataNodes. NameNode also manages DataNode, and responds to the file query, read, write requests from the clients. DataNode stores the data block content and communicates with NameNode. When a client wants to read or write a file, it firstly queries NameNode and gets the file metadata from NameNode; then extracts the data blocks information from the metatdata; lastly, the client communicates with DataNode to do read or write operations.

When HDFS is running, all the files' metadata are stored in the memory space of the NameNode and each metadata needs about 150 Byte memory space. So, the memory space of the NameNode restricts the number of files that can be stored in HDFS file system. For example, if the memory is 4GB, and each file uses only one data block, each file's metadata and its corresponding data block's metadata will consume about 300 Bytes memory space in total. In this case, the NameNode can store about 4GB/300B≈14,316,557 file metadata and this is the upper bound. If it is required to further increase the file numbers that a NameNode can manage, the only way is to increase the memory size of the NameNode. In hadoop 2.x, HDFS federation was designed implemented to solve this problem. HDFS federation mode allows multiple NameNodes coexisting and federated in a HDFS system, and this method partly addressed the above issue, but it requires configuring the mount point of the NameNode statically before running HDFS and does not scale-out well with an increasing number of mount points.

3. Cloud Storage System based on HDFS

3.1. Problem Statement and Key Idea

As discussed above, the total number of files that can be stored in HDFS is constrained by the memory size of the NameNode. The key problem is how to extend HDFS to support multiple NameNodes and let each NameNode store a part of files' metadata. In this way, the maximum number of files that a HDFS system is able to store can be linearly increased with the number of NameNodes. In single NameNode model, all metadata are maintained by one NameNode and the metadata consistency could be easily guaranteed. When expanded to multiple NameNodes model, the following four main problems are required to be seriously considered: (1) How to enable multiple NameNodes coexist and cooperation? (2) How to design the file and directory namespace? (3) How to locate the given file's metadata that a client requests to get? (4) How to guarantee the metadata consistency among all NameNodes?

To solve the above four problems, a new service named "NameNode Location Service" (NLS in short) was introduced to the system. The key design idea and principals include the following four parts: (1) The basic HDFS framework remains unchanged and NameNode still manages a set of DataNodes; (2) Multiple NameNodes are allowed to coexist with each other, and each NameNode is in charge of a separate set of DataNodes; (3) NameNode Location Service (NLS) is used to locate NameNode for clients; (4) When a client wants to read or write a file, it firstly contacts with NLS and NLS will reply the NameNode information to the client, and then the client could communicate with the corresponding NameNode and DataNode. This step remains the same with HDFS file access procedure.

This design methodology could solve the above four problems, because (1) each NameNode runs separately and does not interact with each other. So, the first and fourth problems could be solved; (2) NLS is responsible to located the NameNode which really stores the file's metadata that a client wants to get. In this way, the second and third problems could be solved.

3.2. System Architecture Overview

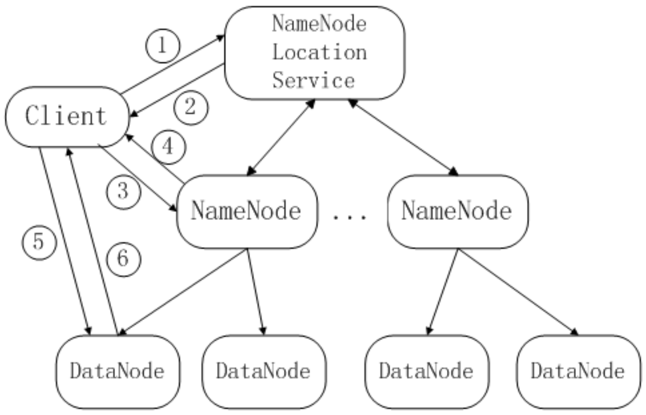


Figure 1. System Architecture

The cloud storage system proposed in this paper was built on an official version of Hadoop HDFS. The system architecture is illustrated in Figure 1. The NameNode module and the DataNode module were standard HDFS modules, and remained unchanged in the system. A NameNode location service (NLS) module was added to the system. It allowed multiple NameNodes coexisting in the system and each NameNode managed a separate set of DataNodes. NLS module collected all of the NameNodes' basic information and responsed to any client's file location query requests.

This system was designed to provide cloud storage service for a large number of users and each user had a separate directory to store their own files and subdirectories. The NLS module only maintained the mapping relationship between the user id and the corresponding NameNode. All the files and subdirectories belonged to the same user would be maintained by the same NameNode. This design methodology made the cloud storage system brief and flexible.

The data flow of the system is illustrated with the case of client reading a file. As shown in Fig 1, when a client wants to read a file, it sends the file owner's user id to the NLS at first; and then, the NLS replies to the client with the information of the NameNode which stores the metadata of the given file; after that, the client could communicate with the NameNode and then access the DataNode in a standard HDFS access manner. The data flow of client writing a file is similar with the reading process.

3.3. NameNode Location Service

NLS is the core module on supporting multiple NameNodes in this system. By using NLS, different users' files could be distributed among several NameNodes and the upper limit of the number of files that the system can manage could be greatly improved. Multiple NameNodes could work independently and deal with the requests from different clients in parallel. So, the system throughput could be increased. New NameNode can be easily added into the system to increase the system storage capacity, which makes the cloud storage have a good scale-out performance.

NLS module deals with the file location requests from clients and reply to the client that which NameNode stores the file's metadata. NLS has to build and maintained a <key, value> type data structure: the key is the user id and the value is the corresponding NameNode's information. NLS does not map file name or file id to the NameNode, instead, NLS only maintains the map information between user id and the corresponding NameNode. All files belonged to the same user could be managed by the same NameNode. The mapping relationship data in NLS should be with high availability and reliability. NoSQL database such as Redis database is a good choice to store the mapping data in NLS.

If a user id does not exist in NLS, NLS could choose a NameNode to store the new user's files. To balance the work load among all NameNodes, NLS should be aware of the basic statistical information of all NameNodes. NLS queries and collects all NameNodes' information in an active way. After that, NLS could be aware of the work load of every NameNode. The amount of data that NLS needs to maintain is proportional to the number of users in the system and it would not become the bottleneck of system.

3.4. Performance Optimization and Discussion

To further reduce the overload of NLS, the client could cache the mapping data between the user id and the corresponding NameNode.

NLS collects information from every NameNode to achieve load balance among all NameNodes. When a new user directory needs to be created, the NLS could choose a NameNode with the lightest work load.

4. Experimental Result

The experimental setting is described as below. Several virtural machines were built on ESX server and the cloud storage system was installed into those virtual machines. Every virtual machine had the same hardware configuration. Each virtual machine had 512MB memory and 20GB hard disk, running CentOS 5.6 and Java 1.6. The Hadoop version is 1.2. The physical memory that the NameNode service itself could utilize is about 256MB in practice. Experiment ran 3 times. The number of virtual machines was 4, 8 and 12 respectively in experiment 1, 2 and 3. Each NameNode managed three DataNodes and there are 1, 2 and 3 NameNodes respectively in experiment 1, 2 and 3. NLS service ran on a separate virtual machine. Client generated files with size between 1KB and 8KB and stored into the system. The total number of files that had stored into the system was recorded when the memory of NameNode had been exhausted.

Experimental results are shown in Figure 2. When there was one NameNode in the system, the client had written about 230, 000 files into the system in total. After that, the system response was very slow and request timeout warnings were reported. The system became saturated. In the second experiment, there were two NameNodes in the system, and the total number of files that the system had stored was nearly doubled. When there were three NameNodes in the system, the total number of files that the system had stored was nearly increased by two times comparing with that of the first experimental result. Those results showed that the system had good scalability.

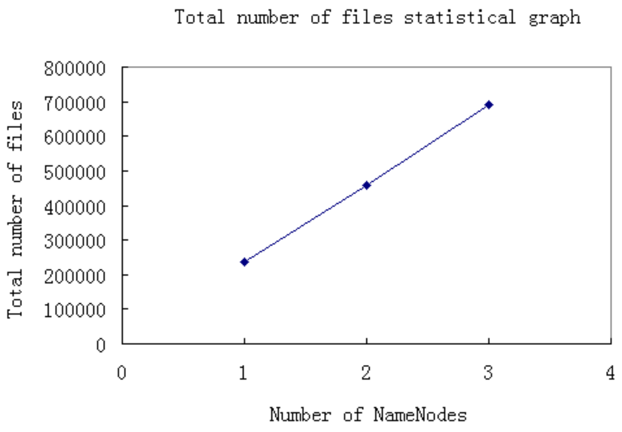


Figure 2. Total Number of Files Statistical Graph

5. Conclusions

This paper presented a cloud storage scheme to utilize multiple NameNodes. This storage system was built on official Hadoop HDFS. NLS module was added into the system and redirected the client's file access request to the corresponding NameNode. Multiple NameNode coexisted and worked separately in the system. Experimental results showed that this storage scheme had good scale-out capability. It provided a solution to the problem that the total number of files that could be stored in HDFS wass constrained by the memory size of the NameNode. It was suitable for providing cloud storage service for lots of users.

Acknowledgements

This work was partly supported by National Natural Science Foundation of China (No. 61373028 and No. 61070154), Shanghai Universities Young Teacher Training Foundation (2012), Foundation for University Key Teacher of Shanghai Maritime University (11th), Science & Technology Program of Shanghai Maritime University (20130471), Innovation Program of Shanghai Municipal Education Commission (12ZZ153).

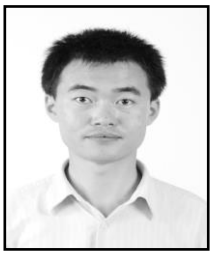
References

[1] S. Ghemawat, H. Gobioff and S. T. Leung, "The Google file system", Proceedings of the 19th ACM symposium on Operating systems principles, Bolton Landing, NY, USA, **(2003)** October 19–22, pp. 29-43.
[2] Apache™ Hadoop®. Hadoop documentation, http://hadoop.apache.org/, **(2014)** February 11.
[3] OpenStack, Swift documentation, http://docs.openstack.org/developer/swift/, **(2014)** June 20.
[4] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall and W. Vogels, "Dynamo: amazon's highly available key-value store", Proceedings of 21st ACM symposium on Operating systems principles, Stevenson, Washington, USA, **(2007)** October 14–17, pp. 205-220.
[5] D. Beaver, S. Kumar, H. C. Li, J. Sobel and P. Vajgel, "Finding a needle in Haystack: facebook's photo storage", Proceedings of the 9th USENIX conference on Operating systems design and implementation, Vancouver, BC, Canada, **(2010)** October 4–6, pp. 1-8.
[6] J. Cui, T. S. Li and H. X. Lan, "Design and development of the mass data storage platform based on Hadoop", Journal of Computer Research and Development, vol. 49, **(2012)**, pp. 12-18.
[7] F. Mu, W. Xue, J. W. Shu and W. M. Zheng, "Metadata management mechanism based on route directory", Journal of Tsinghua University (Sci & Tech), vol. 49, no. 8, **(2009)**, pp. 1229-1232.
[8] L. N. Song, H. D. Dai and Y. Ren, "A learning method of hot-spot extent in multi-tiered storage medium based on huge data storage file system", Journal of Computer Research and Development, vol. 49, **(2012)**, pp. 6-11.
[9] L. Ao, D. S. Yu, J. W. Shu and W. Xue, "A tiered storage system for massive data: TH-TS", Journal of Computer Research and Development, vol. 48, no. 6, **(2011)**, pp. 1089-1100.

[10] G. G. Zhang, C. Li, Y. Zhang and C. X. Xing, “A kind of cloud storage model research based on massive information processing”, Journal of Computer Research and Development, vol. 49, **(2012)**, pp. 32-36.

[11] S. Yu, X. L. Gui, R. W. Huang and W. Zhuang, “Improving the storage efficiency of small files in cloud storage, Journal of Xi'An Jiao Tong University, vol. 45, no. 6, **(2011)**, pp. 59-63.

Authors



Kun Bi received the PhD degree in computer system architecture from University of Science and Technology of China in 2008. He is currently a lecture in the College of Information Engineering at Shanghai Maritime University. His current research interests include cloud storage, cloud security, network security and cloud computing.

Dezhi Han received the PhD degree in computer system architecture from Huazhong University of Science and Technology in 2005. He is currently a professor and master instructor in the College of Information Engineering at Shanghai Maritime University. His current research interests include cloud storage, cloud security and cloud computing.