International Journal of Database Theory and Application
Vol.9, No.11 (2016), pp.1-12
http://dx.doi.org/10.14257/ijdta.2016.9.11.01

Research on Improved Hadoop Distributed File System in Cloud Rendering

Ren Qin[1], Gao Jue[3], Gao Honghao[3], Bian Minjie[1], Xu Huahu[2] and Feng Weibing[1]

[1]School of Computer Engineering and Science, Shanghai University, 200444, Shanghai, China
2Shanghai Shang Da Hai Run Information System Co., Ltd
[3]Computing Center, Shanghai University, 200444, Shanghai, China
1429082779@shu.edu.cn

Abstract

With the rapid development of cloud computing technology, it's the cloud rendering that cloud computing is applied to render the job in CG (Computer Graphic) industry. The cloud rendering can handle a large number of rendering requests which are enormous pressure for back-end servers in system. Facing with massive data and computing resources, the bottleneck of original HDFS (Hadoop Distributed File System) based on cloud computing has become more and more prominent, such as the failure of single Namenode, scalability issues. Therefore, the paper proposed an improved HDFS which evolved a single Namenode into multi-Namenode. In HDFS, Metadata management is very important. So this paper presented a two-level Metadata distribution algorithm. The two-level algorithm was based on the principle that different distribution strategies were used to different categories of Metadata. The experiments verified that the improved HDFS effectively improved the performance of the system.

***Keywords**: cloud computing, cloud rendering, distributed system, HDFS, Namenode*

1. Introduction

1.1. The Present and Problems of File System in Cloud Rendering

The cloud rendering which involves multiple rendering tasks and large rendering resources is the combination of cloud computing and rendering. For example, the movie "Little Door Gods" involved 1940 scenes, 140 characters and would reach 80 million if you use a single-core CPU to render. Facing with a flood of "Little Door Gods" rendering resources, Aliyun used cloud storage technology to make GB/S even TB/S data the movie "small Door Gods" generated smoothly flow among the various computing nodes. The technology offers three-tier solution, which has IaaS (Infrastructure as a Service), PaaS (Platform as a Service) and SaaS (Software as a Service). IaaS is responsible for storing massive rendering resources and providing computing resources [1].

Literature [2] proposed a GlusterFS distributed file system in the cloud rendering. The GlusterFS meets 24-hour service, high scalability, fast reading and writing, *etc.* But the GlusterFS is only applicable to the UNIX operating system kernel and it hasn't load balancing. In the cloud rendering's heterogeneous environment, the development of GlusterFS has been limited.

Although there have been many more matured distributed file systems, its research and application in the cloud rendering are not yet mature. Each cloud service provider is constantly trying to apply a variety of storage solutions to render.

1.2. The Present and Problems of HDFS

HDFS is the abbreviation of Hadoop Distribute File System [3]. HDFS is widely applied and its research keeps continuing. At present, some improved HDFS have been proposed by the scholars.

Literature [4] proposed energy-saving algorithm based on storage structure configuring and symmetric strategy of Block to solve the problem of low energy utilization in HDFS clusters.

Literature [5] put forward a concept of trust-model to evaluate the security and reliability of Datanode in the cluster.

Literature [6] proposed an improved scheme based on Data Coding Strategy and Dynamic Replication Strategy to improve the lack of HDFS low storage efficiency and load imbalance.

These improvements which are used to upgrade the architecture of HDFS have an important sense. But HDFS is also faced with some new problems in cloud rendering. As the data is growing explosively, single-point of failure, performance bottlenecks and other problems are increasingly prominent. Therefore, this paper aiming at the environment of cloud rendering presented an improved HDFS. The improved HDFS can not only solve the problems of the single-point, but also well suit scalability of the system.

2. The Basic Idea of HDFS

HDFS is ready for large-scale distributed computation. HDFS is based on the GFS (Google File System) architecture and so far it becomes an important realization of GFS. It is seen in the remarkable uniformity of GFS and HDFS for architecture, Block size and Metadata, *etc.* But there are many differences between GFS and HDFS. The relations between GFS and HDFS are shown in Table 1.

Table 1. The Relations between GFS and HDFS

	GFS	HDFS
Implementation architecture	Master/Slave	Master/Slave
Property	Distributed File System	Distributed File System
High performance requirements	Data partitioning and replication	Data partitioning and replication
Append operation	Multiple clients concurrently can append the same file, but there are duplication and disorder problems.	Open the file only once and append the data. The use of temporary file operations can solve disorder and duplication problems.
Data nodes	Primary and secondary points and Lease authorization can reduce the amount of work of Master.	There are not primary and secondary points, Lease authorization.
Single-point of failure treatment	The Master data is backed up. When the Master node failed, it would elect secondary node to service.	Master data is written in local computer. When the Master node failed, human intervention is required.
Support for Snapshot	Internal copy-on-write data structure can implement snapshot of cluster.	Snapshot is not supported.
Garbage Collection	Inert Recycling Strategy	Garbage Collection is not supported

HDFS is a file system which can be deployed in a bit of low-cost hardware and provide high-performance, high fault-tolerance and reliable data storage [7]. HDFS is composed of a single Namenode, a lot of Datanodes and client. There is the architecture of HDFS shown in Figure 1.

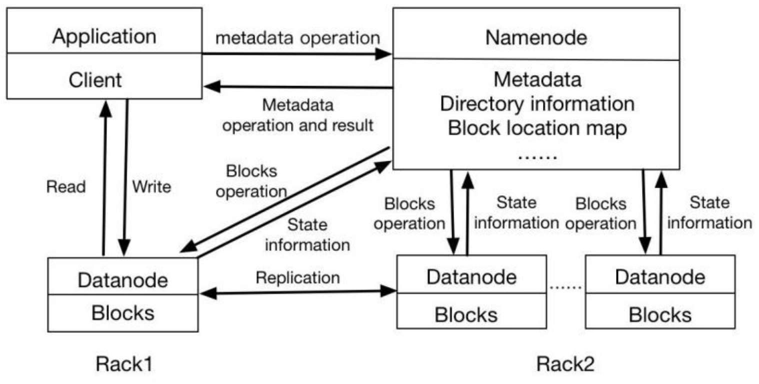

Figure 1. The Architecture of Original HDFS [7]

As seen from the Figure 1, Namenode which is responsible for managing all Metadata information and clients' access to the file system is the central point and manager of the file system. Datanode is the place where the data is stored in HDFS. The following is the detailed description of the system structure and working mechanism of HDFS.

2.1. Namenode

Namenode is also known as Metadata node. There is a single Namenode in Original HDFS, so the system implementation is relatively simple and its control logic is quite clear. Moreover, HDFS can effectively guarantee the uniqueness of the Metadata. There are three main functions of Namenode in HDFS [8].

(1) The management of Metadata and Block

The management of Namenode information includes Metadata information and Block information. Metadata is the data that describes objects such as information resources or data. Metadata is mainly used to identify resources. There are three parts of Metadata information: Metadata namespace, file mapping to Block and Block mapping to Datanode. Block information includes creating a new Block, copying the Block, removing the invalid Block, recycling the isolated Block, *etc.*

(2) The management of the requests from client and Datanode

Namenode not only can process client requests which include the creation, writing, renaming, deletion, opening and removing of files or directories, but also process Datanode requests which include Blocks reporting, heartbeat response, error messages, *etc.*

(3) The persistence Metadata

Namenode maintains the file tree of the entire file system and all files or directories are contained by the tree. This information stores on a local disk in the form of Namespace Image and Edit Log.

In brief, Namenode can control operations with relation to reading and writing in HDFS. All requests initiated by client in system can get right information through Namenode. Besides, when client is writing, Namenode does not allow other operations to ensure data consistency.

2.2. Datanode

Datanode is truly a place to store data in HDFS. Datanode contains many Blocks which are the smallest unit to store data in HDFS. The default size of Block is 64MB. There are three main features of Datanode.

(1) Reading and writing of Block

The operations of reading and writing which are initiated by client must firstly obtain the location information of Block from Namenode. Then client can interact with Datanode.

(2) Reporting node state to Namenode

Datanode needs to report regularly heartbeat information and Block state to Namenode. Once Datanode failed, Namenode would back up and relocate the data which used to be stored in failed Datanode.

(3) Executing the pipeline copy of data

When client creates a new file, Datanode needs to get the location of the secondary node. Firstly, client writes data in Block on local computer. Then client copies the data in Block to secondary node by the way of pipeline.

2.3. The Operations of Reading and Writing in HDFS

The operations of reading and writing mainly refer to the interaction among client, Namenode and Datanode. The operations of writing data in HDFS are shown in Figure 2.

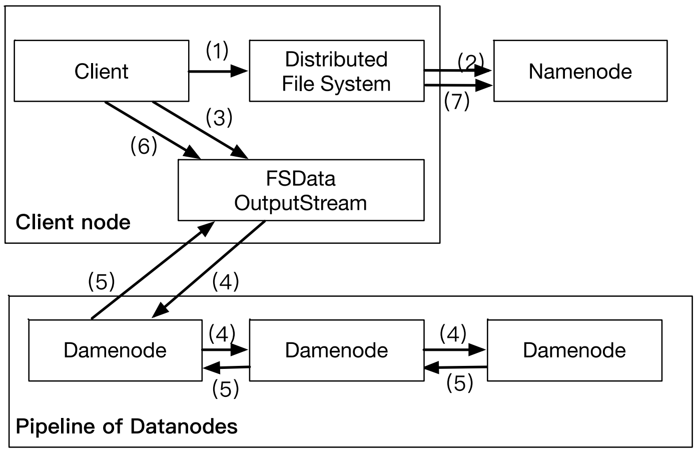

Figure 2. The Operations of Writing Data in HDFS [9]

(1) Client calls the create () operation of DFS (Distributed File System) to create a new file;

(2) DFS calls RPC to create new file information in namespace of Namenode. If it is confirmed that the new file didn’t exist, Namenode would authorize client to create a new file. Otherwise, client will throw an exception;

(3) DFS returns FSDataOutputStream flow to Client. FSDataOutputStream is responsible for communication between Namenode and Datanode;

(4) Client writes data to Datanode. When client starts to write data, FSDataOutputStream splits files into multiple packets and writes packets to Data Queue. Namenode selects the appropriate Datanode to store each new packet;

(5) FSDataOutputStream maintains an ack queue which stores packets waiting for Datanode to acknowledge. If the packets are confirmed by Datanode, they will be removed from the ack queue;

(6) When client completes writing operations, it will call the close () operation to end writing data;

(7) Client notifies Namenode that it completes writing operations.

The operations of reading data in HDFS are shown in Figure 3.

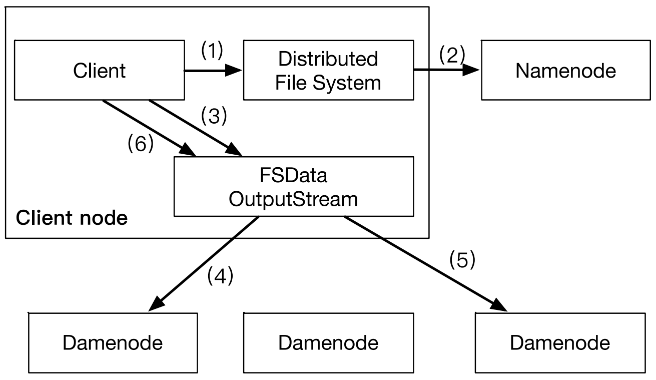

Figure 3. The Operations of Reading Data in HDFS [9]

(1) Client calls the open () operation to send a read request for file to DFS;
(2) DFS interacts with Namenode to get the location information of file through RPC;
(3) Client starts reading data through FSDataOutputStream flow;
(4, 5) FSDataOutputStream calls read () operation to read data from one or more Datanodes;
(6) After reading, client calls close () to complete reading operation.

3. Improved HDFS Based on the Multi-Namenode Architecture

As seen from Figure 1, there is a single Namenode which is responsible for managing all Metadata information in original HDFS. The Metadata operations account for 50% to 80% in the distributed file system. From Figure 2 and 3 we can see client needs to obtain the location information of Block through Namenode when it starts to read or write data. With the increasing amount of data and cluster nodes, the load of single Namenode increases greatly and the bottleneck problem of single Namenode also restricts the development of the system. Therefore, the paper proposed improved HDFS based on the multi-Namenode. The architecture of improved HDFS is shown in Figure 4.

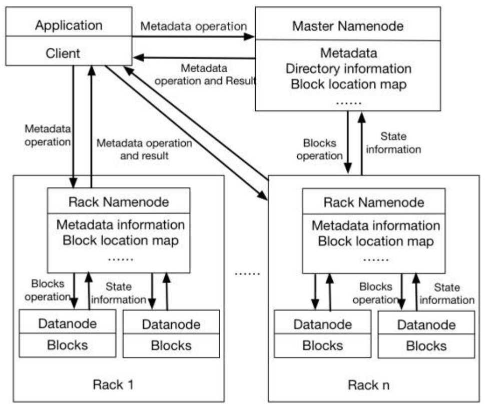

Figure 4. The Architecture of Improved HDFS Based on the Multi-Namenode

Figure 4 shows that there is a Master Namenode and many Rack Namenodes. The Master Namenode is responsible for maintaining all Metadata information in improved HDFS. Each Rack has a Rack Namenode which is responsible for the maintenance of Metadata information in its rack and stays in touch with Master Namenode by heartbeat detection. When client requests a file operation, it must firstly interact with Rack Namenode to get related information of Metadata. If failed, client must obtain information of Metadata from Master Namenode. The improved architecture can decrease the load of Master Namenode by increasing Rack Namenodes. That is because most of the client requests will be completed in the rack, the second requests of client to Master Namenode are small. More importantly, there are a limited number of files in the rack, so the data can be stored in memory. When client requests the operations, the operations can speed up.

3.1. The Election Mechanism of Master Namenode

The improved HDFS faces the primary problem is how to elect the Master Namenode. All nodes can fail at any time in HDFS, so the system needs a reliable election mechanism to elect the Master Namenode. The election of the Master Namenode is a matter of the consistency in distributed file system. Considering the environment of the cloud rendering, this paper presented an election mechanism which is Paxos algorithm based on the timeout mechanism.

The timeout mechanism is proposed because the network bandwidth in cloud rendering cannot be ignored. More importantly, if the timeout mechanism wasn't taken, any algorithm couldn't guarantee to complete in a limited time [10].

The Paxos algorithm is based on the consistency of message passing algorithm [11]. The Paxos algorithm has three roles: Proposer, Acceptor and Learner. But the main interaction involves only Proposer and Acceptor. Proposer is the sponsor who can make a proposal. The proposal contains proposal number and proposal value. Acceptor is the decider who can decide whether he accepts the proposal. Learner can only learn approved proposal.

The Paxos algorithm is divided into two phases, namely the prepare phase and the accept phase. The basic idea of the Paxos algorithm is as follows:

(1)The prepare phase

When each Proposer receives a request of the certain value from Client, it sends a proposal number K to most of the Acceptors. K is the only one in the system and the value of K is increasing. The default value of K is the ID of Proposer while it is also the initial value of first proposal. Then the increasing formula of K is as follows [12]:

Suppose there are N Proposers, each Proposer numbered $I\_r\ (0 \leq I\_r < N)$. The proposal number K submitted by Proposer should be larger than the maximum value. K should satisfy the following formula:

$$K\%N = I\_r \Longrightarrow K = M \times N + I\_r \tag{1-1}$$

After receiving K, each Acceptor should firstly judge the value of K. Suppose the maximum value of proposal number is $MaxN$. If $(MaxN > K)$, Acceptors should reply $< error1 >$ or provide non-response. Then Acceptors must end this proposal process. If $(MaxN < K)$, Acceptors should reply $< MaxN, AcceptN, AcceptV\_N >$ to Proposers. Then Acceptors promise not to accept other proposals whose proposal number are less than K.If there are no previous proposals, Acceptors should reply $< OK >$ to Proposers.

(2)The accept phase

After a while, Proposers could receive some replies from Acceptors. These replies can be divided into the following categories:

①Most Proposers receive replies and all replies are <OK>. Then Proposers make the request with $Accept\ (K, V)$ to Acceptors. K is the proposal number in prepare phase and V is the value corresponding to $MaxN$.

②Most Proposers receive replies but the replies are not the same. They include $< Max1, Accept1, AcceptV\_1 >$, $< Max2, Accept2, AcceptV\_2 >$ and so on. Then Proposers send $Accept\ (K_i, V_i)$ which is corresponding to most replies.

③ Less Proposers receive replies. Then Proposers try to increase the value of K and turn to prepare phase to continue.

Receiving the proposal, Acceptors check whether the $Accept$ would violate one in prepare phase. If not, Acceptors accept this $Accept$.

After the proposal is accepted by most Acceptors, Learner starts to learn the proposal. Acceptors send approved $\mathrm{Accept}\,(K,V)$ to each subset of Learner. Then the subsets inform other Learners in cluster. Figure 5 shows the process of Paxos algorithm.

The Paxos algorithm based on timeout mechanism adds timeout mechanism to the accept phase. Suppose timeout time is T. If Proposer receives a reply from Acceptor within T, its submitted proposal is determined to be accepted. So this Proposer marks himself as Master Namenode.

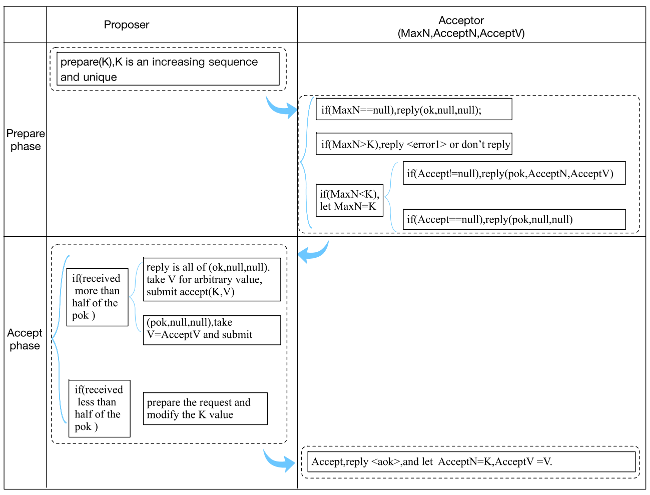
Proposer
Acceptor (MaxN,AcceptN,AcceptV)
Prepare phase
prepare(K),K is an increasing sequence and unique
if(MaxN==null),reply(ok,null,null);
if(MaxN>K),reply <error1> or don't reply
if(MaxN<K), let MaxN=K
if(Accept!=null),reply(pok,AcceptN,AcceptV)
if(Accept==null),reply(pok,null,null)
Accept phase
if(received more than half of the pok)
reply is all of (ok,null,null). take V for arbitrary value, submit accept(K,V)
(pok,null,null),take V=AcceptV and submit
if(received less than half of the pok)
prepare the request and modify the K value
Accept,reply <aok>,and let AcceptN=K,AcceptV =V.

Figure 5. The Process of Paxos Algorithm

3.2. Metadata Information Based on the Two-Level Distribution Strategy

How the Metadata information is distributed and managed is also an important research problem in improved HDFS. Therefore, the paper proposed a two-level distribution strategy based on the combination of hash algorithm and dynamic load distribution.

The Metadata information can be divided into two types: (1) the Metadata information of directory and file; (2) the Metadata information of Block. The Metadata information of directory and file stores some attribute information, such as file name, directory name, parent directory information, file size, *etc*. The Metadata information of Block combines the mapping relation between Blocks and Datanodes. Owing to the different characteristics of structures and access information in both types of Metadata, they are given different distribution strategies.

3.2.1. Directory and File Distribution Strategy Based on $\boldsymbol{hash}$ Algorithm

All file operations must operate Metadata information in HDFS at first. Considering the locality, this paper proposed hash algorithm to store the Metadata information of directories and files.

Based on the hash algorithm, client can obtain $\mathrm{Rack\ Namenode} - \mathrm{id}$ of a directory or file by the formula (1-2) and (1-3):

$$Key = Hash(ParentPath) \tag{1-2}$$

$$Rack\ Namenode - id = F(Key) \tag{1-3}$$

$\mathrm{ParentPath}$ represents the parent directory of directory or file. Key represents the value which $\mathrm{ParentPath}$ obtained through Hash. $\mathrm{Rack\ Namenode} - \mathrm{id}$ is the id value of Rack Namenode in N servers and the range of $\mathrm{Rack\ Namenode} - \mathrm{id}$ is from 0 to $N - 1$. $F()$ is the distribution function and Key will be distributed to the interval of $[0, N - 1]$. The attribute of $\mathrm{ParentPath}$ could determine the number of Rack Namenode, which ensures locality of Metadata. Rack Namenode stores the Metadata information which belongs to the same file. The distribution strategy of directory and file based on hash algorithm is as follows:

(1)Client initiates a request to write files or directories and it provides FileName (orPathName) and ParentPath to request Rack Namenode − id;

(2)Client obtains Rack Namenode − id through formula (1-2) and (1-3);

(3)Client determines whether the load of Rack Namenode has exceeded the threshold. If exceeded, turn to step (4). If not exceeded, jump to step (5).

(4)Client hashes again to get a new Rack Namenode − id.

(5)Client returns the Rack Namenode − id and continues the subsequent operations.

3.2.2. The Location Information of Block Based on Dynamic Allocation Ratio

Each file is divided into many Blocks in HDFS. The default size of each Block is 64MB and each Block has three backups. Suppose the size of file is 4GB and its directory information is only 200B and its Block location information is 12KB. The location information of Block is 60 times the size of directory information. It is obvious that the number of location information of Block reaches a large scale in the cloud rendering. Therefore, the paper proposed the distribution strategy of location information of Block based on dynamic allocation ratio.

This algorithm assigns computing resources to each node by considering the resources available and transmission cost [13]. Then the system can find out the optimum node to complete the task. The algorithm is mainly based on two dynamic global tables which are AGT_1(Active Global Table) and $AGT_2$to solve the problem of Namenode load. The table AGT_1 can indicate load situation when there is a task to apply. The table structure of AGT_1 is shown in Table 2.

Table 2. AGT_1 Structure

Node ID	Available resources

AGT_2 is based on AGT_1 and it is the table of candidate node information. The table structure of AGT_2 is shown in Table 3.

Table 3. AGT_2 Structure

Node ID	Available resources	Transfer cost	Allocation proportion

The value in AGT_1 and AGT_2 dynamically updates according to the network tasks. The system calculates the allocation proportion of each candidate node according to the formula (1-4), (1-5), (1-6), (1-7), (1-8), so the rendering resources can be allocated to higher node.

When the system startups, it needs to dynamically generate AGT_1 and AGT_2. The set Rack Namenode represents the set S.The performance of node initiates available resources of $AGT_i$and it includes CPU frequency $F\_t\ (t = 1,2,3 \cdots m)$, memory capacity M_i, disk I/O rate R_i and network throughput T_i [14]. Suppose the weight is $k_p (p = 1,2,3,4, \sum k_p = 1)$ and the available resources of node S_i can use the following formula to initialize AGT_i [14]:

$$S_{i-available} = k_1 \times (\sum F_t) + k_2 \times M_i + k_3 \times R_i + k_4 \times T_i \tag{1-4}$$

If $H\ (H \subseteq S)$ represents the set of candidate nodes, the nodes in H should satisfy the following conditions[15]:

Setting a threshold value ε, if any node S_i in AGT_1 satisfies expression (1-5) and (1-6):

$$S_{-available} = \max\{S_{i-available}\}, i = 0,1 \cdots n. \tag{1-5}$$

$$S_{i-available} > S_{-availble} - \varepsilon \tag{1-6}$$

Please add S_i to the set H.

When there is a network task to apply, the transfer cost to reach the candidate node $S_j(j < n)$ in set H must consider the number of nodes D, the performance of nodes C and the bandwidth of nodes B. Suppose the weight is $k_u(u = 1,2,3, \sum k_u = 1)$.When the task gets to node S_j, the transfer cost is calculated by the formula (1-7).

$$S_{j-transfer} = k_1 \times D + k_2 \times C + k_3 \times B \tag{1-7}$$

The allocation proportion of node S_j can be calculated by expression (1-8).

$$S_{j-allocation} = S_{j-available} \div S_{j-transfer} \tag{1-8}$$

By comparing the allocation proportion of each Rack Namenode, the system can determine where to store the data in the rack. The larger the allocation proportion is, the higher the availability of the rack is.

4. Experimental Results

Original HDFS faces the increasing number of files which has been a bottleneck in the cloud rendering. So this paper proposed an improved HDFS which was based on the multi-Namenode. Therefore, experiments of this paper compared the time difference in terms of reading and writing between original HDFS and improved HDFS by changing the size of files.

4.1. Experimental Environment

This experiments contained 10 computers which was divided into three racks. There was a Master Namenode and each rack has a Rack Namenode and two Datanodes. The experimental architecture is shown in Figure 6.

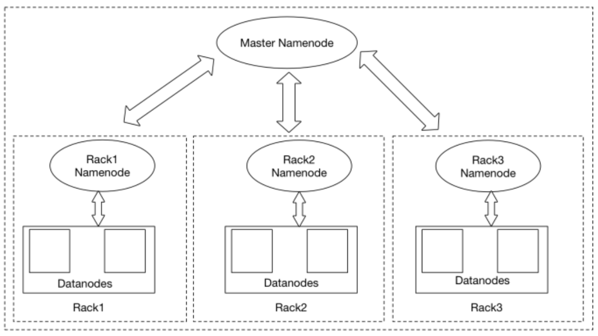

Figure 6. The Experimental Architecture

Each node communicated with Gigabit Ethernet in improved HDFS and its information used in experiments is shown in Table 4.

Table 4. Node Information in Experiments

Node	CPU Model	CPU Frequency	Memory	Hard disk
Master Namenode	i7-4790	3.60GHz	4GB	1T
Rack Namenode	i5-4460	3.20 GHz	4GB	512G
Datanode	i3-4160	3.60GHz	4GB	64G

4.2 Experimental Results and Analyses

It can be seen from the Figure 7 that time of writing file was no significant difference between original HDFS and improved one when the file was small. With the file increasing, the time of writing was nonlinear growth and the ratio coefficient was increasing in original HDFS. It was indicated that the original HDFS showed instability during writing operations. But in improved HDFS, time of writing significantly reduced and changed gently. It was concluded that improved HDFS has been optimized and stable in terms of performance.

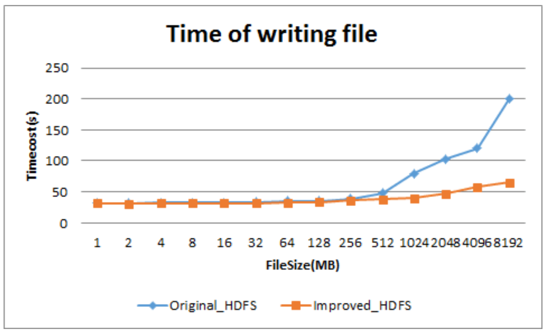

Figure 7. Time of Writing File

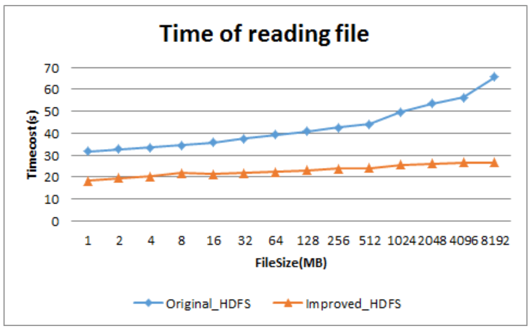

Figure 8. Time of Reading File

Figure 8 showed the time of reading data. It can be seen from the picture that compared with original HDFS, improved HDFS enhanced greatly in reading operations. Moreover, there was also a very good improvement in the stability of the multi-reading and multi-writing when faced large files in improved HDFS.

The experiments showed that the speed of reading and writing have been improved in improved HDFS. Furthermore, the stability of improved HDFS is better than that of the original one. So the experiments proved that the architecture of improved HDFS is rational and available.

5. Conclusions

Aiming at the storage problem in Cloud rendering, the paper proposed a multi-Namenode architecture based on original HDFS. Improved HDFS must select a Master Namenode to manage all Metadata information while Rack Namenode is responsible for the management of the Metadata information in each rack. When the file reads or writes data, it firstly accesses Rack Namenode to obtain the location information of Block. If fails, it has to visit Master Namenode. The experiments proved that the improved architecture has availability and rationality.

Acknowledgements

This paper is supported by National Natural Science Foundation of China (NFSC) under Grant No.61502294, Natural Science Foundation of Shanghai under Grant No.15ZR1415200, Innovation Project of Next Generation Internet Technology of

CERNET under Grant No. NGII20150609, Foundation of Science and Technology Commission of Shanghai Municipality under Grant No. 14590500500, and The Special Development Foundation of Key Project of Shanghai Zhangjiang National Innovation Demonstration Zone under Grant No. 201411-ZB-B204-012.

References

[1] W. Bin, "Research for SME business model IAAS clod computing architecture", Beijing University of Posts and Telecommunications, **(2014)**.
[2] L. Miamian, "Research of the GlusterFS distributed storage system applied in the cloud rendering platform", Jiangsu University of Science and Technology, **(2015)**.
[3] X. Xiaojun, G. Yang and S. Lin, "Parallel text categorization of massive text based on Hadoop", Computer Science, vol. 38, no. 10, **(2011)**, pp. 184-188.
[4] L. Bin, Y. Jiong and Z. Tao, "Energy-Efficient Algorithms for Distributed File System HDFS", Chinese Journal of Computers, vol. 36, no. 5, **(2013)**, pp. 1047-1064.
[5] W. Yongzhou, "Research of storage technology based on HDFS", Nanjing University of Posts and Telecommunications, **(2013)**.
[6] L. Xiaokai, D. Xiang and L. Wenjie, "Improved HDFS scheme based on erasure code and dynamical-replication system", Journal of Computer Applications, vol. 32, no. 8, **(2012)**, pp. 2150-2153.
[7] X. Dawen and R. Zhuobo, "Research and application of key technology of Hadoop", Computer and Modernization, no. 5, **(2013)**, pp. 138-141.
[8] Z. Bo, "Research and optimizing of data storage under HDFS", Guangdong University of Technology, **(2013)**.
[9] Z. Bo, "Research on the metadata management of multi-Namenodes based on HDFS", University of Electronic Science and Technology of China, **(2013)**.
[10] M. J. Fischer, N. Lynch and M. S. Paterson, "Impossibility of distributed consensus with one faulty process", Journal of the ACM, vol. 32, no. 2, **(1985)**, pp. 374-382.
[11] Y. Ping'an, "Research and design of HDFS high availability based on Paxos", South China University of Technology, **(2012)**.
[12] M. Burrows, "The Chubby lock service for loosely-coupled distributed systems", Osdi Proceedings of Symposium on Operating Systems Design & Implementation, **(2006)**, pp. 335-350.
[13] Z. Xiaojing, W. Rong and N. Lei, "A server load balancing algorithm based on dynamic allocation ratio", Henan Science, no. 7, **(2014)**, pp. 124-1243.
[14] Z. Yufang, W. Qinlei and Z. Ying, "Load balancing algorithm based on load weights", Application Research of Computers, no. 12, **(2012)**.
[15] N. Nehra, R. B. Patel and V. K. Bhat, "Loading balancing in heterogeneous P2P systems using mobile agents", World Academy of Science, Engineering and Technology, vol. 18, no. 6, **(2006)**, pp. 77-82.

Authors

Ren Qin, The author is a Master candidate in graduate students in School of Computer Engineering and Science Shanghai University. her main research is about distributed storage and cloud rendering.