A Client-Side Directory Prefetching Mechanism for GlusterFS

Zijian Liu, Fang Dong, Junxue Zhang, Pengcheng Zhou, Zhuqing Xu, Junzhou Luo
School of Computer Science and Engineering
Southeast University
Nanjing, P.R. China
{zjliu, fdong, jxzhang, pczhou, zqxu, jluo}@seu.edu.cn

Abstract—**Distributed file system has the characteristics of large capacity, good scalability and high reliability, which make it widely used in many areas involving large-scale data storage. It offers simplified, highly-available services for users to access data. However, due to the non-metadata design, the performance of traversal operation on large directories in those non-metadata distributed file systems is poor. With the increasing amount of files, it severely affects the user experience. In this paper, we present a directory prefetching mechanism on the client side to reduce directory traversal operation latency in non-metadata distributed file system. The mechanism, combined with the client's cache, adopts the directory access history to predict future access pattern and fetches the content of the directory without user intervention. Our goal is to reduce the overall access latency in the non-metadata distributed file system in order to better satisfy the user experience.**

Keywords—GlusterFS; Distributed File System; Prefetch; Metadata

I. Introduction

With the advent of the era of the big data, there are a number of data resources that need to be stored, managed and accessed. As the result of the limitation of capacity, scalability and performance, the traditional pattern of a single storage device cannot meet the massive data storage and access requirements. Cloud storage is one of the best solutions [1]. It provides data storage and data access functions for ordinary users. Many distributed file systems, such as Ceph [2], GFS [3] Lustre [4] and GlusterFS [5], have been proposed in the last few years and widely applied in many fields involving the cloud storage [6]. These distributed file systems assemble many file servers on the network and make them an integrated file system with large storage volume, where files are distributed across these file servers. Users can efficiently access their files without caring about the problem that which specific file server the file is stored on.

In order to distinguish the right file server that a specific file is located on, the distributed file system is required to provide additional information about this relationship between the file and its storage location. This information is also a special kind of file system metadata [17]. There are two kinds of metadata management in the distributed file system, one is metadata service model and the other is non-metadata service model [7]. Some file systems, such as Lustre and GFS, utilize the metadata server to locate the data. When a client accesses the distributed file system, it needs to interact with the metadata server before directly communicating with the I/O storage nodes [8]. Since the performance of the metadata server is limited, the increasingly expanding speed of the storage nodes, files and clients will bring the problems of a single point of failure and performance bottleneck.

To avoid those problems, some other file systems change their design patterns. They have no special metadata storage concept, and the metadata is stored with the file itself. GlusterFS is one of the most representative non-metadata file systems. This kind of file systems uses specific deterministic algorithms to locate the position of the data object without the help of the metadata server [9]. The elastic hash algorithm used in GlusterFS is one of the specific algorithms and can be described as follows: It firstly calculates the hash value based on the file path and the file name and then selects the storage server according to the hash value to locate the file. For a given file name, it is efficient to locate the file. This design mechanism overcomes the shortcomings of the first model, and completely parallelizes data access, implementing the real linear extension performance. However, in some scenarios, such as joint software development and collaborative manufacturing, multiple people need to cooperate to access and share a large number of data, and the number of files in a directory increases abruptly. The performance of some directory query and update operations, such as ls tools on Linux and browsing directories on Windows, in which the file name in the directory isn't given, will degrade sharply. Meanwhile, when the number of users accessing the distributed file system keeps growing, network load becomes heavy, which may result in performance degradation as well. These two aspects make the directory traversal operation become inefficient, and the latency of opening a directory will dramatically affects the user experience in the process of accessing the file system.

In the non-metadada service model, the performance of traversal operation on large directories has raised much concern. GlusterFS uses Gluster Native Client [16] method for high concurrency, performance and transparent fault tolerance in GNU/Linux clients, and provides prefetching support for directories to improve read performance of large directories in Gluster Native Client. However, it is currently designed to handle the simple and sequential directory read operations. If a non-sequential directory read occurs, the prefetching support for directories will fail. Meanwhile, this approach has not been implemented on the client that uses NFS/CIFS protocol to access GlusterFS. For the other operating systems that cannot run Gluster Native Client, such as Windows, the performance of reading large directories is still poor.

In this paper, in order to improve the experience of Windows users who are accessing GlusterFS, we propose a prefetching mechanism based on the user access history to reduce the access latency of opening a directory when the network load becomes heavy and the number of the files in a directory keeps increasing. We design and implement the prefetching mechanism on the NFS client on Windows without any user-supplied intervention. Before a directory is opened, the prefetching operation is responsible for determining the directories that the user is going to open in the next step and storing the information in the cache. Cooperating with the caching technology on the client or the server, this mechanism significantly reduces perceived access latency.

The remaining of this paper is organized as follows. Section II provides the background information and related work. Section III presents the motivation of the prefetching mechanism on the NFS client for GlusterFS. Section IV describes users' browsing directory behaviors and the details of the mechanism design. Section V demonstrates performance optimization. Section VI concludes this paper and introduces the future work.

II. Background and Related Work

A. Introduction to GlusterFS

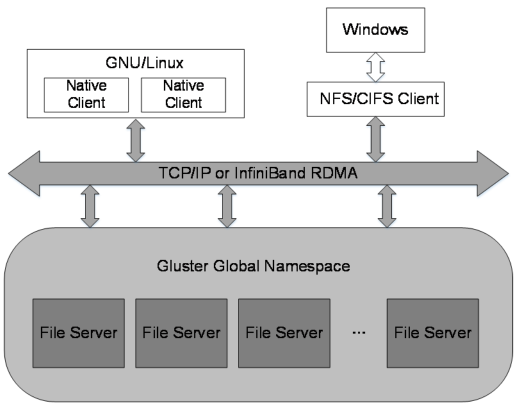

Fig. 1. Overall architecture of GlusterFS

GlusterFS is an open source distributed file system without metadata server. The architecture gives it powerful ability of horizontal development. As shown in Fig. 1, it aggregates the storage resources distributed on different commodity computers together through the interconnection of TCP/IP or InfiniBand RDMA, and manages the data using a single global namespace. Users can access Gluster volumes through Gluster Native Client in GNU/Linux clients. For the other operating systems that cannot run Gluster Native Client, such as Windows, users can also use NFS/CIFS protocol to access the data.

B. Introduction to NFS Protocol

NFS protocol is designed by Sun Microsystems to provide transparent remote access to shared file systems across network. It is independent of the specific machine, operating system, transport protocol and network architecture. The NFS server can be seen as a file server and the NFS client can use MOUNT protocol to attach remote directory trees. Users and programs can access files on the remote the NFS server as well as access the local file system. NFS protocol has 3 versions, namely NFS v2, NFS v3 and NFS v4. Users can use NFS v3 to access Gluster volumes.

C. Related work

In order to improve the performance of the distributed file system, many technologies have been widely used, including the performance optimization for massive numbers of small files [12], fault-tolerant mechanism [10], load balancing approach [11] and so on. Due to the limitation of the disk I/O rate and the network speed, access latency has become one major restricting factors in distributed file system. Caching is a kind of solutions to this problem [13]. It is used to store the contents that have been requested by users and will be possibly used in the future, thereby it may reduce the access latency. Client side cache [14], server side cache and intermediate cache [15] are three levels of caching strategies in the distributed file system. They can accelerate the speed of accessing the file system on the whole. Our work differs from the cache in that we take an initiative approach to fetching cache data. Before the data is actually accessed, it has been read into the cache to reduce the access latency of the file system. At the same time, the contents that we prefetch are the attributes of directories and files rather than the file data itself. In another way, Gluster Native Client [16] has provided read-ahead support for directories to improve read performance of large directories in GlusterFS, but it currently supports a maximum readdir buffer of 4k, and is not based on the user's historical behavior. More importantly, it is a FUSE-based client running in user space. This reason limits its application scope.

III. Motivation

In this section, we consider the necessity of using the directory prefetching mechanism based on the user access records on the client side in GlusterFS, and the reason why we implement the prefetching mechanism on the NFS client. In order to markedly illustrate the problem of the latency of accessing large directories mentioned above, we use six Fedora 23 server virtual machines to set up a distributed file system based on GlusterFS for the experimental test.

A. Large Number of Files Causes Performance Dropping

Since GlusterFS utilizes elastic hash algorithm to locate the file storage location according to the file name, when a directory is read, the file names in the directory are unknown, thus GlusterFS needs to search the entire storage pool to locate the files in the directory. The effect of this operation on the user experience is shown in Fig. 2, which depicts the relationship between the access latency of opening a directory and the number of files in the directory. The more the number of files is, the higher the access latency will be. When the number of files in a directory reaches 500, the access latency is approximately

2000ms. In the real application scenarios involving massive data, with the number of files growing sharply, the user experience becomes poorer.

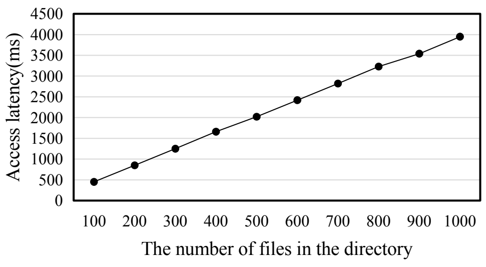

Fig. 2. Access latency of opening directory

B. Network Load Causes Performance Degradation

Although GlusterFS consists of many file servers, and adopts the load-balancing mechanism to fairly distribute the access load of the entire file system, when a client is mounted on the distributed file system through the NFS protocol, all interactions between the client and the file system is through the mounted file server. With the increasing quantity of clients, the network load of the mounted machine becomes heavy, and the client performance will vary with the network load. Fig. 3 shows the comparison of the access latency between the network load increases before and goes up to 40Mbps. It vividly illustrates the influence of network load on the delay of opening directories.

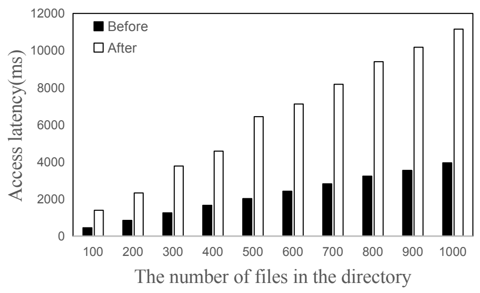

Fig. 3. Comparison of the access latency

C. NFS client is more general than Gluster Native Client

In GlusterFS, Gluster Native Client provides high concurrency and performance to access the distributed file system, however, it is only compatible with GNU/Linux clients. In order to expand the application scope of GlusterFS, NFS server has been implemented in GlusterFS, which enables users access GlusterFS through NFS Client on different operating systems. With designing the prefetching directories strategy on NFS client, it will really reduce the latency of opening large directories.

In conclusion, since the performance of GlusterFS drops dramatically when the number of files in the directory becomes large and network load rises, and has caused a serious impact on the user experience, it is necessary to take measures to reduce the access latency. In addition, Gluster Native Client cannot be run on multiple platforms. These factors prompt us to take a prefetching directory mechanism on the NFS client to improve the user experience.

IV. DESIGN

In this section, according to the directory structure of the distributed file system, we analyze the characteristics of the user's browsing directory behavior on the Windows operating system and find out the influencing factors that can be used to predict the user's future behavior. On the basic of these factors, we build a model, called record tree, and put forward the prefetching directory mechanism and the caching scheme cooperating with the prefetching mechanism.

A. Analysis of user's browsing directory behavior

We hypothesize that directories in the distributed file system are orderly organized, and with the increasing frequency of users accessing the file system and the reason that users tend to access the frequently accessed content, there is pronounced regularity in directory access characteristics for each user [18]. During the process that users access the distributed file system, we continue to record the user's behavior, and take these records to predict user's future behavior.

When a user accesses the distributed file system through NFS on Windows, the entire storage space of the distributed file system is mapping to a local disk on Windows, and the operations on the distributed file system are analogous to the ones on local file system. So user behaviors of accessing distributed file system can be modeled as long as the operations of ordinary users on the local file system have been analyzed.

In the file system, directories are organized as a tree structure. After a user opens the parent directory, subdirectories and files in the parent directory will be shown, and the user can keep opening the subdirectories. The process of accessing directories in the file system can be shown in Fig. 4. The files in the directories are omitted. Directories are the nodes, and the directed edge from parent A to child B denotes the directory access sequence that the directory A is accessed firstly and the directory B is accessed afterwards. Accessing directories from top to bottom is a sequential process. Users can not open the directory D ahead of opening the directory B on Windows. Meanwhile, when users reach an intermediate directory D in the process of accessing the file system, users are likely to come down to open subdirectory G or K, and may also return to the parent directory B or the directory B's parent directory A.

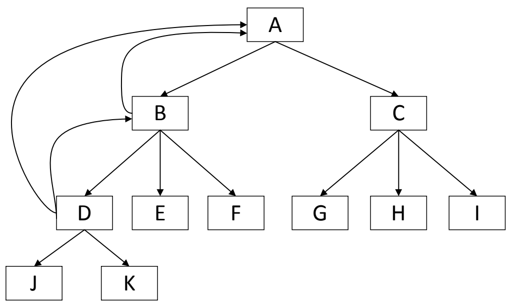

Fig. 4. Process of accessing directories

When users access a file system frequently, some historical information is left. The historical information includes the directory's creation time, modification time, last access time and so on. Users tend to access the directories that just have been modified and visited a few times ago. At the same time, some frequently accessed directories are more significant to users, and users are more likely to visit them next time. So the number of directories being accessed also should be taken into account.

B. The Prefetching and Caching Mechanism

Based on behaviors of users accessing distributed file system on Windows through the NFS client, which include sequential reads from the upper directories to the underlying directories and returns to the upper directories from the current directory, two methods are respectively adopted to deal with the two cases. We first use the prefetching algorithm to predict the subdirectories that users may open, and then use the caching mechanism to store the just visited directories and the parent directories that users may open. We shall illustrate the reason why we take the two methods later.

1) The Prefetching Mechanism

In order to record the user's access history in the process of accessing the file system, a tree structure that records the directory information of the file system is required to be maintained on the user's local machine, dubbed a record tree. Each node represents a directory entry, and contains a tuple property:

(*Size, Last Access Time, Access Frequency*)

Regardless of the user's access privileges, the directories in the distributed file system can be visited by multiple users from different machines. Thus, the record tree is closely related to each user and the information in the tuple may be inconsistent with the attributes of the directory itself. For example, the last access time of a directory entry in the record tree is the time that the particular user visited the directory last time. Other users may visit the directory afterwards and change the last access time of the directory itself. This is an important reason that we design the prefetching mechanism in the client.

If a user accesses the file system for the first time, the record tree in the local machine only has a root node representing the root directory. The record tree is constructed in the course of user accessing the file system. Whenever a new directory is visited by the user, some new child nodes representing subdirectories are added to the parent node, and change the last access time and access frequency of the parent node. At the same time, the subdirectory attributes obtained from the file system are added to those child nodes' tuple properties.

Thus, when a user wants to open a subdirectory in the current directory, the tuple property of the nodes representing the subdirectories can be gained from the record tree. According to the information, some analysis can be performed to estimate which subdirectory the user is going to visit next time. The analysis process is summarized as an algorithm as shown in Algorithm 1, called subdirectory prefetching algorithm. The correlation function calculates the correlation between the subdirectory and the current directory, and then the subdirectory prefetching algorithm determines the subdirectories that users may visit in the next time.

Algorithm 1 Subdirectory Prefetching

```
1:  Input: Current Directory $D$
2:  Let $S_k = N_1, N_2, \ldots, N_k$ be the child node set of $D$ in the record tree
3:  for each child node $N_i$ in $S_k$ do
4:      $correlation_i \leftarrow$ CorrelationFunc($D, N_i$)
        Normalize $correlation_i$ to $p_i$
        $t_i \leftarrow Size$
        Insert $p_i$ into Possibility Set $P$
        Insert $t_i$ into Time Set $T$
5:  End for
6:  Choose $N_i$ that satisfies the requirement according to $P$ and $T$
7:  Prefetch the subdirectory referring to $N_i$
```

As mentioned earlier, those subdirectories that are frequently visited and just have been visited are more likely to be visited in the future, and as time goes, the possibility of an earlier visited subdirectory to be visited at the next time gradually becomes low. Because reading directories on Windows is a process from the upper directories to the underlying directories, the number of the parent directory being visited is roughly equal to the sum of the number of all subdirectories being visited, and the last access time of subdirectories is also related to that of the parent directory. So CorrelationFunc(D, N_i) can be formulated as follows:

$$correlation_i = max\left\{\frac{N_{i,frequency}}{D_{frequency}}, \frac{CurrentTime - D_{last}}{CurrentTime - N_{i,last}}\right\}$$

Where N_i represents a subdirectory and D represents the current parent directory. *CurrentTime* is the time that the user open directory *D*. We select the last access time and access frequency of a directory to measure the correlation between the subdirectory and the parent directory.

After we have calculated the correlation between each subdirectory and the parent directory, we should choose some subdirectories to be fetched. Although the more subdirectories being fetched in the cache are, the higher the hit rate is when subdirectories are opened, it will bring a lot of network overhead to the file system. Our prefetching goal is to minimize the overall access latency to the file system under the condition of limited size of cache. We call the cache used to store the prefetched directories prefetching cache. We normalize the correlation to the probability and formalize the subdirectories selection strategy as a 0-1 knapsack problem:

$$p_i = \frac{correlation_i}{\sum_1^k correlation_i},$$

$$max \sum_1^k p_i \times t_i \times x_i$$

$$\text{s.t. } \sum_1^k x_i \times cost_i \le C \text{ and } x_i \in \{0,1\},\ i = 1,2,\ldots,k$$

Where $cost_i$ is the size of the content which is used to record the file information in directory N_i and t_i is the time of opening N_i. According to our previous experimental result, $cost_i$ is positively related to t_i in general, and t_i can be estimated by *Size*. We can gain $cost_i$ from the property of N_i, and the sum of the size of directories being prefetched should be less than *C*, the size of the prefetching cache. When a subdirectory N_i is prefetched into the cache, $x_i = 1$. And if not, $x_i = 0$.

With these formulas, we can find out the maximum access delay which can be optimized under the given prefetching cache size. Then we put these subdirectories into the cache and can reduce the overall maximum access delay. The 0-1 knapsack problem is NP-Complete, and we can use a heuristic algorithm to find a better result to satisfy the condition. Algorithm 2 is the algorithm that we use at present.

Algorithm 2 Greedy algorithm

```
1:  Input: p, t, cost
2:  for each $p_i$, $t_i$, $cost_i$ do
3:      $U_i \leftarrow \frac{p_i \times t_i}{cost_i}$
        Insert $U_i$ into Set U
4:  End for
5:  Sort Set U
6:  Choose $N_i$ that $U_i$ is larger
```

In Algorithm 2, U_i represents the utilization rate of the unit prefetching cache when the directory N_i is put in. We always try to prefetch the subdirectory that maximizes the utilization rate.

2) The Caching Mechanism

In view of the case that users may return to upper parent directories from the current directory, if we put the parent directories as child nodes of the current directory in the operation of reading directory, since the frequency of the former is greater than the latter, and the last access time of the former is less than the latter, so the correlation between parent directories and the current directory is greater than that between the current directory and subdirectories. In addition, the size of the parent directory is also greater than the size of the subdirectories, thus the possibility that the parent directories being chosen is always greater than subdirectories , and the directories being prefetched first are the parent directories.

In order to make the parent directories do not affect the performance of the prefetching mechanism, we do not prefetch the parent directories, and store the content of parent directories in the cache separately. At the same time, for the subdirectories that have been visited, the contents of them are also stored in this cache. The cache is updated regularly to keep consistent with the data in the file system.

V. Evaluation

In this section, we implement our proposed prefetching mechanism for GlusterFS on the NFS client, and evaluate the performance of the prefetching mechanism brought to the opening directory operation on Windows.

A. Experiment and Parameter Settings

We use a 6 node cluster to build a distributed file system based on GlusterFS. Each node is deployed on a Fedora 23 Server virtual machine on the different physical machines connected by Ethernet, and each virtual machine has one CPU, 1G memory and 20G RAM. The NFS client on Windows uses NFS v3 to access GlusterFS.

In the process of the experiment, we calculate the size of the directory and file attributes, and find that in order to save the information in the NFS client, we need at least 36 bytes in the memory without the file name or directory name. In order to store the directory content with 200 files, we require about 10K cache. In the experiment which is used to test the accuracy of prefetching, the size of file name and directory name is taken into consideration and the average number of files in the directory is 200, thus we set the prefetching cache 200K.

B. Access latency with prefetching mechanism

Before, in order to illustrate the problem that the latency of accessing large directories is serious in GlusterFS, we store 10 folders with different number of files in the cluster. The size of each file is 64K bytes. The goal of our proposed prefetching mechanism is to improve the performance of accessing large directories in GlusterFS. The comparison result between no prefetching mechanism and prefetching mechanism on the access latency is shown in Fig. 5. We ignore the time required to prefetch the directory. With the increasing number of files in the directory, the access latency without the prefetching mechanism ascends continuously. If the prefetching mechanism is adopted into the client, the variation of the access latency is not obvious. It demonstrates that our mechanism can significantly improve the access performance in the case of large directories being prefetched in advance.

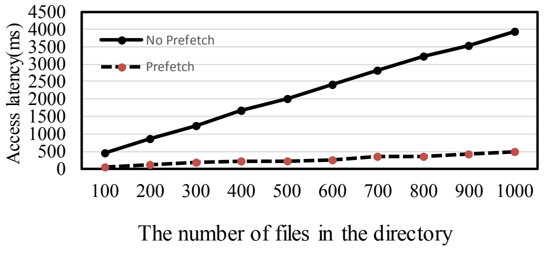

Fig. 5. Access latency before and after the prefetching mechanism is adopted

C. Accuracy of prefetching

To evaluate the effectiveness of the proposed prefetching mechanism on predicting the future access without user intervention, our second metric is the accuracy of prefetching directories, and defines it as the percentage of directory access predictions that are actually made. We invent five volunteer users to copy their files and directories in the frequently-used disk into GlusterFS, and perform their normal work activity on GlusterFS. Fig. 6 depicts the average hit rate of the five users in different number of directory clicks.

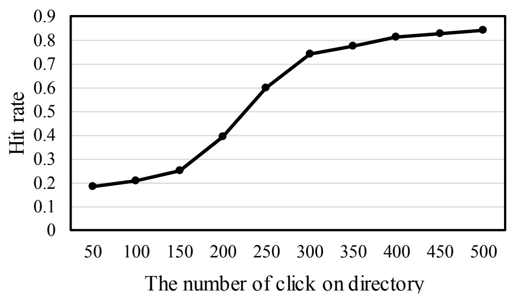

Fig. 6. Hit rate of opening directories

At the beginning stage, due to the lack of adequate user historical access records, our prefetching mechanism has little work on predicting user future access. There is little difference in the correlation between the parent directory and subdirectories, and our objective is to minimize the overall access latency, so the main influencing factor is the size of the directory. After a certain period of time, the NFS client accumulates some user operation records, and the accuracy of the correlation between the parent directory and the subdirectories increases, so the reliability of predicting the future access ascends sharply. In the final stage of the record, the hit rate of opening a directory rises slowly and gradually reaches 90%.

Although we predict users' future access based on a large number of access records, there is a certain change in the behaviors of the user accessing directories. Meanwhile, the hit rate is related to the size of the prefetching cache and the size of the directory, and our goal is to minimize the overall access latency. Thus, there must be some errors in our predictions.

VI. Conclusion

In this paper, we systematically discuss the problem that the performance of accessing large directories in non-metadata distributed file system is low. We analyze the user's access behavior in the file system, and design a prefetching mechanism to utilize the previous access history to predict the future access. The mechanism has been implemented in the NFS client. And the evaluation result shows that our prediction mechanism has a high hit rate in the case of adequate user historical access records, and can significantly reduce the latency of accessing large directories.

There are several works that we can continue to accomplish in the future. First, the network load in the server cluster caused by prefetching mechanism should be considered. Second, the size of the cache that store the prefetching content has an impact on the hit rate. We need a lot of experiments to study the problem. Third, we do not take into account the time required to prefetch directory. In order to overcome the shortcoming, we can take multiple directories prefetching mechanism.

VII. Acknowledgements

This work is supported by National Natural Science Foundation of China under Grants No.61320106007, No.61572129, No.61502097, No.61370207, National High-tech R&D Program of China (863 Program) under Grants No.2013AA013503, International S&T Cooperation Program of China No. 2015DFA10490, Jiangsu research prospective joint research project under Grants No.BY2013073-01 , Jiangsu Provincial Key Laboratory of Network and Information Security under Grants No.BM2003201, Key Laboratory of Computer Network and Information Integration of Ministry of Education of China under Grants No.93K-9, and partially supported by Collaborative Innovation Center of Novel Software Technology and Industrialization and Collaborative Innovation Center of Wireless Communications Technology.

References

[1] B. Mao, S. Wu, and H. Jiang, "Improving Storage Availability in Cloud-of-Clouds with Hybrid Redundant Data Distribution," in 2015 IEEE IPDPS, 2015, pp. 633–642

[2] Weil, Sage A., et al, "Ceph: A Scalable, High-Performance Distributed File System," 7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA, 2006:307--320.

[3] Garcia, Heynert, and A. Ludu. "The Google file system." *Acm Sigops Operating Systems Review* ACM, 2003:29-43.

[4] Braam, Peter J, "The Lustre storage architecture," CLUSTER File Systems, Inc. Mountain View, Ca 2004.

[5] Davies, Alex, and A. Orsaria, "Scale out with GlusterFS," Linux Journal 2013.235(2013).

[6] T. S. Soares, M. A. R. Dantas, D. D. J. de Macedo, and M. A. Bauer, "A Data Management in a Private Cloud Storage Environment Utilizing High Performance Distributed File Systems," in Proceedings of IEEE WETICE. IEEE, Jun. 2013, pp. 158–163.

[7] S. A. Brandt, L. Xue, E. L. Miller, and D. D. E. Long, "Efficient metadata management in large distributed file systems," In Proceedings of the 20th IEEE / 11th NASA Goddard Conference on Mass Storage Systems and Technologies, pages 290–298, Apr. 2003.

[8] S. A. Weil, K. T. Pollack, S. A. Brandt, and E. L. Miller, "Dynamic metadata management for petabyte-scale file systems," in SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing. Washington, DC, USA: IEEE Computer Society, 2004, p. 4.

[9] Xie, Tao, and A. Liang, "Small file access optimization based on GlusterFS," Journal of Shenyang Jianzhu University (2015):101-104.

[10] Krishnan, Sasidharan, and S. G. Urkude, "Providing fault tolerant storage system to a cluster," US, US 7886183 B2. 2011.

[11] Zhang, Cong Ping, and J. W. Yin, "Dynamic Load Balancing Algorithm of Distributed File System." Journal of Chinese Computer Systems 32.7(2011):1424-1426.

[12] Dong, Bo, et al, "A Novel Approach to Improving the Efficiency of Storing and Accessing Small Files on Hadoop: A Case Study by PowerPoint Files," IEEE International Conference on Services Computing IEEE Computer Society, 2010:65-72.

[13] Cantrell, Thomas George, et al. "System and method for efficient cache management in a distributed file system." US, US 6119151 A. 2000.

[14] M. Dahlin, et al, "Cooperative Caching: Using Remote Client Memory to Improve File System Performance," In Operating Systems Design and Implementation, pages 267.280, 1994.

[15] Noronha, Ranjit, and D. K. Panda, "IMCa: A High Performance Caching Front-End for GlusterFS on InfiniBand," International Conference on Parallel Processing IEEE, 2008:462-469.

[16] GlusterFS: A scalable network filesystem. http://www.gluster.org/.

[17] Biardzki, Christoph, and T. Ludwig, "Analyzing Metadata Performance in Distributed File Systems," Parallel Computing Technologies. Springer Berlin Heidelberg, 2009:8-18.

[18] Yamakami, Toshihiko. "An Exploratory Analysis on User Behavior Regularity in the Mobile Internet." Knowledge-Based Intelligent Information & Engineering Systems, International Conference, Kes, Bournemouth, Uk, October 2006:14