PPMS: A Peer to Peer Metadata Management Strategy for Distributed File Systems

Di Yang, Weigang Wu, Zhansong Li, Jiongyu Yu, Yong Li

PPMS: a Peer to Peer Metadata Management Strategy for Distributed File Systems

Di Yang, Weigang Wu, Zhansong Li, Jiongyu Yu, Yong Li
Department of Computer Science, Sun Yat-sen University
Guangzhou 510006, China
{ yangdi5, lizhans, yujiongy, liyong36}@mail2.sysu.edu.cn, wuweig@mail.sysu.edu.cn

Abstract. Distributed file system is one of the key blocks of cloud computing systems. With the fast increase of user scale and data amount, metadata management has become a crucial point affecting the overall performance of a distributed file system. In this paper, we design and implement PPMS, a novel metadata management strategy in a peer to peer way. Different from existing metadata management methods, we adopt a two layer structure to achieve high scalability and low latency. The upper layer is metadata index server, which is used to store metadata of directories, while the lower layer consists of metadata servers to store the metadata of files. More importantly, the lower layer is organized in a peer to peer way to further improve scalability. We implement a prototype file system based on PPMS and evaluate its performance via experiments. The results show that our design can achieve high performance with in terms of time latency and system throughput.

Key words: Distributed File System, Metadata Management, Scalability, Low Latency, Peer-to-Peer

1 Introduction

Distributed file system is one of the key enabling technologies for distributed computing, especially cloud computing [7][8][17]. Although metadata usually accounts for only a very small part of a distributed file system in terms of data size, more than half (50%~80%) of file operations are involved with metadata [1]. Metadata management has become one of key issues in distributed file system [13], and it can significantly affect the overall performance and scalability in large-scale distributed file system [2][17].

In most existing distributed file systems, such as HDFS of Hadoop [7] and GFS from Google [8], there is only one single metadata server (MDS for short), which is likely to become a bottleneck as users and the quantity of files increase. With the rapid increase of user scale, such metadata management is definitely not scalable enough.

Although there have been quite a number of distributed metadata management solutions proposed by researchers, including static subtree partitioning [1], dynamic subtree partitioning [4], such tree-based metadata management strategies cannot scale well due to the tight coupling among metadata servers. To address this problem, peer

to peer based metadata management strategies [5] have been recently proposed, which organize metadata servers in an ad hoc way. Such metadata management strategies are well scalable, but they cannot achieve fast metadata access due to the lack of connections among metadata servers and consideration of user behaviors.

In this paper, we propose a novel peer to peer based metadata management strategy, named PPMS (Peer-to-Peer Metadata Service), which also organizes metadata servers in a peer to peer way. However, different from existing peer to peer metadata management, we combine hierarchy structure with peer to peer way. More precisely, we propose the concept of metadata index server (MIS for short). Our design has two layers of servers, which are in charge of metadata of directories and files respectively. MIS is in the upper layer and takes charge of managing the metadata of directories. MDSs compose the lower layer and manage the metadata of files. The correspondence between a file metadata and its local MDS is established based on the location of the client that creates the file. Compared with existing metadata management strategies, PPMS can achieve a better tradeoff between scalability and latency.

To validate the correctness of PPMS and evaluate its performance, we have also developed a prototype file system, named PPFS. We test PPFS using the popular benchmark tool Postmark [16] and the RES trace. Various operations, including read, creation, are executed to measure access latency and system throughput. MooseFS [14] is also tested for comparison purpose. The results show that PPMS can outperform MooseFS in nearly all cases.

The rest of the paper is organized as follows. Section 2 briefly reviews existing solutions for metadata management, especially peer to peer based ones. We describe the design of PPMS and PPFS in Section 3. Section 4 presents the performance evaluation based on experiments. Finally, Section 5 concludes the paper and suggests future directions.

2 Related Works

With the emergency of large-scale distributed file systems that separate metadata from file read/write operations, metadata management strategies has become a hot research topic and quite a number of metadata management strategies have been proposed.

Static subtree partitioning [1] divides the whole file directory tree into non-overlapped partitions, which are assigned to different MDSs by the system administrator. The partitioning is static and can only be changed manually. This strategy is very simple and easy to implement. However, it is not flexible and may face the problem of workload imbalance among MDSs. Re-balancing will cause large overhead. Dynamic subtree partitioning [4] is proposed to solve the load imbalance problem of static partitioning. It divides the whole directory tree into overlapped partitions, each of which is assigned to one MDS dynamically. By migrating heavily workload metadata automatically and overlapping popular partitions, the workload among different MDSs can be well balanced [12]. However, such design requires

additional mechanism to maintain consistency among different copies of the same piece of metadata.

Hash based partition [3] can also solve the imbalance problem of static partition. A hash function based on file identifier is used to distribute the workload among metadata servers. With a well designed hash function, load balance among MDSs is achieved easily. However, rename operations or change of MDSs may cause lots of metadata migrations crossing MDSs. Another drawback is that hashing inherently discards the advantage of locality.

All strategies above are based on portioning of the directory tree. Such strategies can achieve high performance in terms of access latency, but may suffer from poor scalability. On the other hand, with the increase of user scale and data scale, scalability is becoming more and more important. To achieve high scalability in metadata management, peer to peer based strategies have been proposed.

Hierarchical Bloom-Filter Array (HBA) [5] uses a two-tier probabilistic array, i.e. Bloom filter array, to calculate corresponding MDS to the file a user want to query. In the probabilistic array, the first layer has a higher accuracy ratio but only part of the metadata stored, and the second layer stores all the metadata about the files but has a lower accuracy ratio. When the number of files increases, HBA will have a demand of large memory space to ensure a certain degree of accuracy.

Grouped Hierarchical Bloom-Filter Array (G-HBA) [6] is an extension of HBA by introducing the concept of group of MDS. This scheme logically organizes MDSs into a multi-layered query hierarchy and exploits grouped Bloom filters to efficiently route metadata requests to desired MDS through the hierarchy.

Besides, there is some particular metadata management for special requiremnt including Spyglass and SmartStore [9] [10] .

Although peer to peer metadata management based on Bloom filter can scale easily due to the loose coupling among MDSs, such strategies are generally probabilistic in terms of locating a file, and consequently may suffer from long access latency [11] .

3 The Design and Implementation of PPMS

3.1 Overview of PPMS

Basically, we follow the idea of peer to peer file sharing, where each node can access data at other peers in a fully distributed way. Peer to peer approach can achieve high scalability easily and is also suitable for metadata management. However, to avoid long file access latency, we extend peer to peer based approach by introducing a two-tier hierarchy.

Metadata generally includes directory information and file information in distributed file system. And our proposed metadata strategy PPMS consists of two types of servers, i.e. metadata index server (MIS) and metadata server (MDS) correspondingly. MIS is mainly responsible for directory attributes, query and load balancing, while MDS is primarily responsible for the file attributes. MIS and MDS interact with each other, work together to complete the management of metadata, and

accordingly handle a variety of user operations. In addition to the herein of metadata types, there is a classification for MDS logically, i.e. the local MDS and the related MDS. Local MDS is that a client mounts initially, while related MDSs represents MDSs that are binding with individual directories and most likely to store nonlocal file in that directory. The overall architecture of the PPMS is shown in Fig. 1.

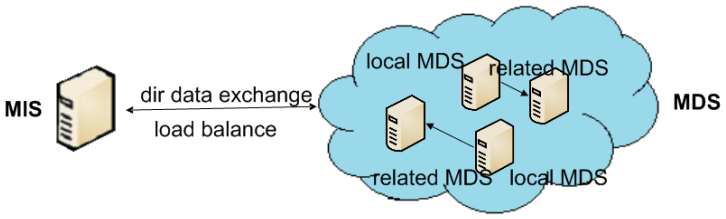

Fig. 1. Overview of PPMS. There are one MIS and multiple MDSs in PPMS, and MDSs are divided into local MDS and related MDS logically.

3.2 The Design of MIS

MIS manages the entire directory metadata information within PPMS, such as directory name, permissions, user name, group name, related MDS list and so on. MIS receives directory-related requests from MDS and provides directories operations, such as directory deletion, directory creation and so on. If a client queries for file metadata, the metadata that cannot be found in both the local MDS and related MDS or there is no related MDS for the file's parent directory, then the request will be forwarded to MIS to retrieve the corresponding metadata.

Although it hasn't had time to realize, MIS is a coordinator for load balancing among MDSs. MIS can monitor the workloads of MDSs via metadata requests received. If some MDSs have too much more workloads than others, MIS will invoke the migration procedure to migrate metadata from busy MDS to those with low workloads. With such mechanism, workloads balance is achieved in the scope of metadata service.

3.3 The Design of MDS

A MDS stores the metadata of files, including file name, permission, user name, user group, size, etc. Each client is associated with its local MDS. When a client creates a file, the file' s metadata will be stored at the local MDS, and the metadata of its parent directory will be sent to the MIS.

One MDS becomes a local MDS once a client has mounted on it. As the local MDS for a client, it is directly responsible for the client's requests. Before mounting, the client configures the IP and port information of the local MDS. Then the client

keeps contact with its local MDS. Also, the local MDS has become the only entrance for the client to the entire metadata management system. Compare to the other MDSs, the local MDS has a greater possibility to store the metadata that its corresponding clients requests.

Besides, every MDS also maintains a related MDS list for each directory of the files whose metadata is stored locally. A related MDS is designed for directory, and it is the node that has the file's attributes under the same directory. That is, a related MDS has metadata of files in the same directory. Related MDS also has a good possibility to have files under the same directory, and can be queried when the metadata requested is missed at the local MDS itself.

The design of related MDS is the core of PPMS. In the beginning, none of related MDS is defined in each MDS. When a client reads the metadata of given file, which is missing at the local MDS, MIS will be queried and the metadata of the file will be found at another MDS through MIS. Then, the requested MDS is defined as the related MDS for this file's parent directory in the local MDS. Since the number of MDS is uncertain, which the related MDS for a directory has one is not an effective solution when there is a very large number of MDS. For each directory, there may be more than one related MDSs. The number of MDS can be determined based on the availability of storage space and other factors.

3.4 Data Access

PPMS provides low latency and improves service quality continually through three layers of query structure after Related MDS appeared. The procedure of accessing a file is shown in Figure 2.

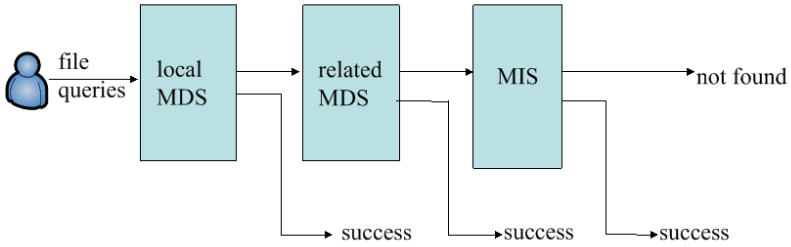

Fig. 2. The procedure of accessing a file. The query of a file involves three levels: looking up the storage of local MDS, looking up the related MDS and looking up MIS.

The first layer of the query structure is the local MDS, which has great probability to meet client needs by directly dealing with write and read requests. In general, files under the same directory have a great correlation and it is ordinary for a client who has interests in the same types. Therefore, the related MDS has also a high hit ratio as the second layer of the query structure. Moreover, the last layer of the query structure

is MIS, which masters all the directory information to satisfy all client requests and avoid global broadcasting. As a result, PPMS has low latency to content clients' requests through hierarchical query structure after analyzing the user possible behavior.

3.5 PPFS -- a prototype file system using PPMS

In order to verify the feasibility and correctness of our strategy, we have implemented a prototype system, called PPFS, using C programming language. The system consists of three modules: metadata management module, chunk server (i.e. node storing file data) module, and client module. In addition, metadata management module, which is also called PPMS module, is responsible for managing allover metadata and namespace, and this module also includes MIS module and MDS module designed as stated above. And the job of chunk server module is to store actual file data. In addition, client can get file data in the distributed file system through client module on the mounted point. To simplify the implementation, client module is developed based on FUSE [15], a file system in user space included in the kernel of linux and widely used by many fields system, such as ZFS, glusterfs and lustre. Besides, we have also implemented client cache, MDS cache and chunk server cache to improve the performance of file access referring to other file system. Figure 3 shows the overall architecture of our PPFS prototype.

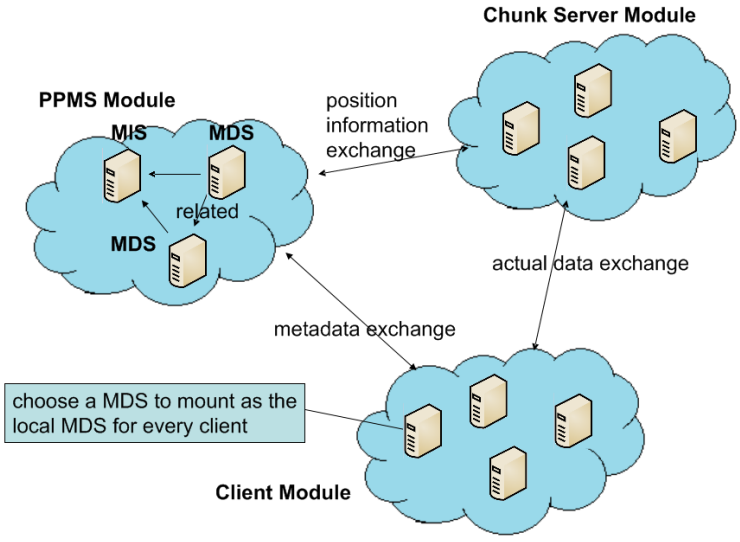

Fig. 3. Overview of PPFS. The system consists of three modules: PPMS module, chunk server module, and client module.

4 Performance Evaluation

In order to evaluate the performance of PPMS, we deploy the PPFS prototype. To make the experiment more persuasive, we did two experiments.

In the first experiment, this test was divided into two parts to show the advantage of PPMS. In the first part, we simply choose MooseFS to compare, because the first part just want to run PPFS with a MDS, and to look for a single MDS system to make a comprision. In addition, PPFS implementation refers to MooseFS, which has only one MDS, and MooseFS is a light weight distributed file system that has been widely used for research and testing [19] with a single MDS [14] . In the second part, we test the performance of the system by increasing the number of MDS isometric.

In the second experiments, we simulate the metadata operations using the RES traces and measure the performance in terms of hit ratio of the local MDS and the related MDS.

4.1 Testing using Postmark

The testing is conducted using Postmark [16], a file system benchmarking software widely used. Postmark generates an initial pool of random text files ranging in a configurable size. Once this pool is created, a specified number of transactions, including create, delete, read and append, are performed on these files randomly. When all the transactions have completed, the remaining files and directories are deleted and statistics are done to compute the performance metric values. We use several metrics, including total time, number of operations per second, system throughput, etc.

We installed MooseFS on a computer equipped with 1G memory and running Ubuntu 11.10 and deployed PPFS on a machine with the same deployment. We use four performance metrics to measure performance of PPMS. These four metrics includes total time to complete all the transactions, number of transactions per second, number of creation per second and number of read per second.

When transaction is 2000 and number of files increase, the results of total execution time are plotted in Figure 4. First, we can see obviously the effect of number of files. More files are in the system, more time is needed. This is expected. Compared with MooseFS, PPFS can execute much faster in nearly all cases of file numbers. This clearly shows the advantage of our design. In PPFS, two-layer hierarchy helps much in locating a file.

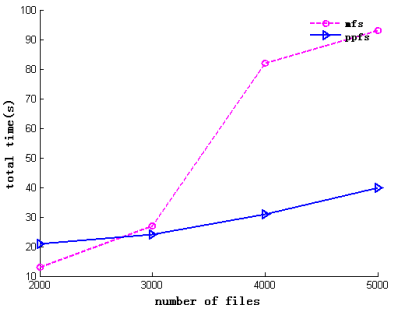

Fig. 4. Total execution time.

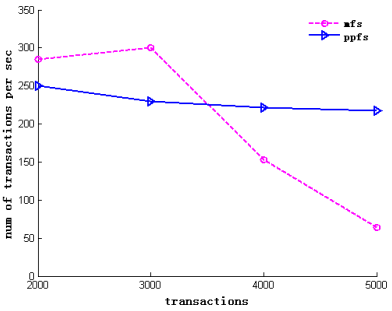

Fig. 5. Number of transactions per second.

Figure 5 demonstrates how many transactions can be completed per second. We test different numbers of transactions to show the performance under different cases. With the number of transaction increases from 2000 to 5000, the number of transactions processed by either system decreases. This is because that, with more transactions, there may be more conflicts in data update, and then fewer transactions can be completed per second.

Compared with MooseFS, PPFS performs much better since PPFS is not affected much by the increase of transaction number. With the help of MIS, which has a whole view of PPFS, PPFS can avoid conflicts in operations and consequently handle more transactions in the same time duration.

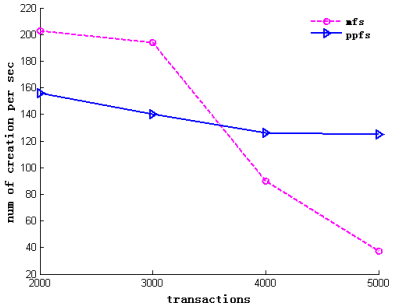

Fig. 6. Number of creations per second.

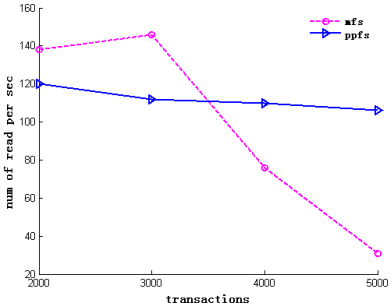

Fig. 7. Number of read per second.

Figure 6 and Figure 7 show the results of file creation and file read respectively. Comparing PPFS and MooseFS, we can see that PPFS can read/create files faster than MooseFS, in most cases. The difference increases with the increase of transaction number. This can be explained as follows. When a file is created in MooseFS, not only the metadata of the file need to be added, but also the hierarchical directory structure needs to be updated at the MDS. In PPFS, two different nodes are used to maintain the file metadata and directory metadata respectively, and obviously the task can be conducted faster. Of course, collaboration between MDS and MIS may cause addition overhead.

MooseFS is faster than PPFS only when the number of transactions is small. This because that, with few files, the directory structure is simple and the benefit of separating file metadata and directory metadata is counteracted by the overhead of cooperating MIS and MDS.

Finally, we examine the effect of number of MDSs in terms of total time by Postmark. Different from previous experiments, it has 14 machines, one of which running Windows 7 and others still running Ubuntu 12.04. The only one running Window 7 manipulates all Ubuntu machines using Xshell. Every client node creates 300 files and deletes all the files by Postmark at the same time. The results are plotted in Figure 8. We vary the number of MDSs from one to four. As expected, the total time decreases when the number of MDSs increases.

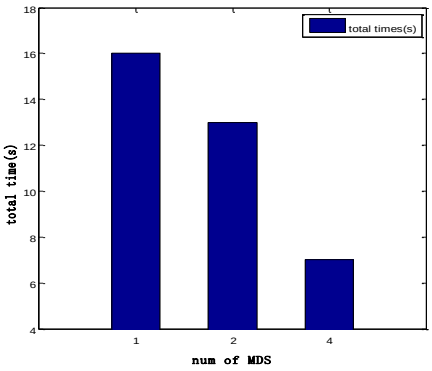

Fig. 8. Average latency of each request.

4.2 Trace Simulation

To verify our system better, we simulate the metadata operations using RES trace and measure the performance in terms of hit ratio of the local MDS and the related MDS. RES trace was collected from 13 machines on the desktops of graduate students, faculty, and administrative staff of their research group project during one year at University of California Berkeley in 1996 and 1997 [18]. These hosts were used for a wide variety of tasks including document processing, program development, graphically displaying research results, email, and web browsing [18].

We downloaded part of data from official website and analyze data referring to the online prompts step. Since we only care about the metadata, operations that are not related was not extracted. Due to PPFS does not have directories and files at the beginning, we should create corresponding files and directories for replaying using a appropriate strategy and then replay RES trace on PPFS.

Limited by the experiment environment, we deploy a mini systems composed of 9 machine. Because real data is not involved in the replay, the chunk server module was not involved. In this system, there are one MIS, four MDSs, and four clients. After

different machines running corresponding processes separately, the results are shown in Figure 9 and Figure 10.

Figure 9 and Figure 10 show hit ratio of MDSs by replaying RES trace on PPFS. We can clearly tell that the hit ratio of local MDSs are at a high level from the first figure. As time goes by and more traces are performed, the hit ratio of local MDSs are almost increasing gradually. This is because that design of local MDS in PPMS refers to the user's behavior, and it is directly responsible for user's write and read operations. At the same time, hit ratio of related MDSs has a common trend that the hit ratio is getting higher and higher in a long time. At the beginning, related MDSs need to establish and replace the antiquated, inefficient related MDSs, so the related MDSs were not efficient at that time. Due to the design of related MDS is based on user behavior in accessing files in the same directory with a high frequency and the files in the same directory is more likely in one MDS. Related MDSs has a high hit ratio overall with time increasing.

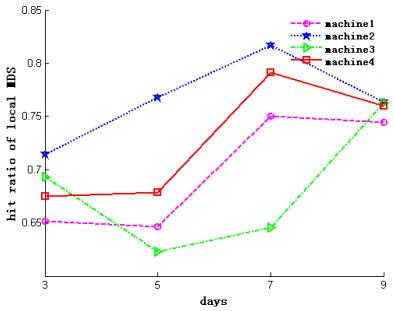

Fig. 9. Hit ratio of local MDS.

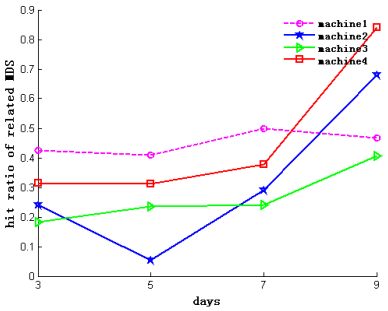

Fig. 10. Hit ratio of related MDS.

5 Conclusion and Future Work

Distributed file system is one of the key blocks for distributed computing systems including cloud computing platforms. We focus on metadata management to achieve high scalability and low access latency simultaneously. With the novel concept of metadata index server, we divide metadata into two layers, i.e. file metadata and directory metadata, and propose a corresponding two layer metadata management strategy. In the lower layer, MDS servers are organized in a peer to peer way, so as to achieve high scalability. In the upper layer, MIS is used to achieve low latency. We have implemented a prototype file system and tested it using Postmark and RES. Compared with MooseFS, our design can achieves significant improvement.

Our design can be further improved and extended in many directions as the first stage. One extension may be multiple MISs. In the current design, there is only one MIS, which is prone to single point failure and may become a bottleneck in performance. A peer to peer MIS layer will be obviously more scalable and reliable. Another interesting work is metadata replication, which should be an effective way to

reduce metadata access latency and improve reliability. Finally, the system implementation should be further improved.

Acknowledgments. This research is partially supported by National Natural Science Foundation of China (No. 61379157), Guangdong Natural Science Foundation (No. S2012010010670), and Pearl River Nova Program of Guangzhou (No. 2011J2200088).

References

1. Roselli, D. S., Lorch, J. R., Anderson, T. E.: A Comparison of File System Workloads. USENIX Annual Technical Conference, General Track. 41-54 (2000)
2. Brandt, S. A, Xue L., Miller, E. L., et al.: Efficient metadata management in large distributed storage systems. 2012 IEEE 9th International Conference on Mobile Ad-Hoc and Sensor Systems (MASS 2012). IEEE Computer Society, 290 (2012)
3. Corbett, P. F, Feitelson, D. G.: The Vesta parallel file system. ACM Transactions on Computer Systems (TOCS), 14(3): 225-264 (1996)
4. Weil, S. A., Pollack, K. T., Brandt, S. A., et al.: Dynamic metadata management for petabyte-scale file systems. Proceedings of the 2004 ACM/IEEE conference on Supercomputing. IEEE Computer Society, 4 (2004)
5. Zhu, Y., Jiang, H., Wang J.: Hierarchical bloom filter arrays (hba): a novel, scalable metadata management system for large cluster-based storage. Cluster Computing, 2004 IEEE International Conference on. IEEE, 165-174 (2004)
6. Hua, Y., Zhu, Y., Jiang, H., et al.: Scalable and adaptive metadata management in ultra large-scale file systems. ICDCS, pp.403-410 (2008)
7. Borthakur, D.: The hadoop distributed file system: Architecture and design. Hadoop Project Website, 11: 21 (2007)
8. Ghemawat, S, Gobioff, H, Leung, S T.: The Google file system. ACM SIGOPS Operating Systems Review. ACM, 37(5): 29-43 (2003)
9. Leung, A. W., Shao, M., Bisson, T., et al.: Spyglass: Fast, Scalable Metadata Search for Large- Scale Storage Systems. FAST, pp.153-166 (2009)
10. Hua, Y., Jiang, H., Zhu, Y., et al.: SmartStore: A new metadata organization paradigm with semantic-awareness for next-generation file systems. High Performance Computing Networking, Storage and Analysis, Proceedings of the Conference on. IEEE, 1-12 (2009)
11. Broder, A, Mitzenmacher, M. Network applications of bloom filters: A survey. Internet mathematics, 1(4): 485-509 (2004)
12. Weil, S. A., Brandt, S. A., Miller, E L, et al.: Ceph: A scalable, high-performance distributed file system. OSDI, pp.307-320 (2006)
13. Wang, J., Feng, D., Wang, F., et al.: MHS: A distributed metadata management strategy. Journal of Systems and Software, 82(12): 2004-2011 (2009)
14. Moosefs, http://www.moosefs.org/
15. FUSE, http://fuse.sourceforge.net/
16. Katcher, J.: Postmark: A new file system benchmark. Technical Report TR3022, Network Appliance, 1997. http://www. netapp.com/tech_library/3022.html, (1997)
17. Patil, S., Gibson, G. A.: Scale and Concurrency of GIGA+: File System Directories with Millions of Files. FAST. 11: 13-13 (2011)
18. Trace,tracehost.cs.berkeley.edu
19. Yu, J., Wu, W., Li, H.: DMooseFS: Design and implementation of distributed files system with distributed metadata server. APCloudCC, pp.42-47 (2012)