HSM^2: A Hybrid and Scalable Metadata Management Method in Distributed File Systems

Yiduo Wang[1], Youxu Chen[1], Xinyang Shao[1], Jinzhong Chen[2], Liu Yuan[3], and Yinlong Xu[1(✉)]

[1] School of Computer Science and Technology, University of Science and Technology of China, AnHui Province Key Laboratory of High Performance Computing, Hefei, China
{duo,cyx1227,sxy799}@mail.ustc.edu.cn, ylxu@ustc.edu.cn
[2] East China Research Institute of Electronic Engineering, Hefei, China
chenjin_zhong@126.com
[3] China Academy of Electronics and Information Technology, Beijing, China
lyuan@csdslab.net

Abstract. In the bigdata era, metadata performance is critical in modern distributed file systems. Traditionally, the metadata management strategies like the subtree partitioning method focus on keeping namespace locality, while the other ones like the hash-based mapping method aim to offer good load balance. Nevertheless, none of these methods achieve the two desirable properties simultaneously. To close this gap, in this paper, we propose a novel metadata management scheme, HSM^2, which combines the subtree partitioning and hash-based mapping method together. We implemented HSM^2 in CephFS, a widely deployed distributed file systems, and conducted a comprehensive set of metadata-intensive experiments. Experimental results show that HSM^2 can achieve better namespace locality and load balance simultaneously. Compared with CephFS, HSM^2 can reduce the completion time by 70% and achieve 3.9× overall throughput speedup for a file-scanning workload.

Keywords: Metadata management · Distributed file systems · Namespace locality · Load balance

1 Introduction

Distributed file systems (DFS) like GFS [10], HDFS [24], Ceph [27] and Lustre [22] have intensively adopted when building highly scalable and reliable Internet services in data centers. As shown in Fig. 1, the metadata and data of DFS are managed by two sets of independent and separated servers, namely, metadata servers (MDS) and data servers. When accessing a file, a client should first fetch the file metadata from MDS to understand the layout and data location, and check the permission. Then, if the file exists and the permission check succeeds, she can read or write data content by directly interacting with data servers.

H. Shen and Y. Sang (Eds.): PAAP 2019, CCIS 1163, pp. 195–206, 2020.
https://doi.org/10.1007/978-981-15-2767-8_19

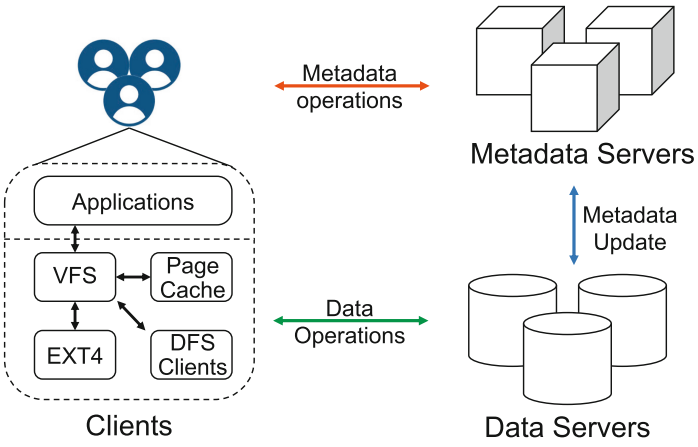


Fig. 1. The architecture of distributed file systems.

In the big data era, small files are dominating the whole file space. For instance, some studies suggest that the mean file size is around hundreds of kilobytes or smaller [4,5,8,11,32]. In addition, compared to the data operations, the metadata operations in file systems are also popular and even account for more than 50% in the overall file system operations [1,3,13,20]. These two trends introduce the design of DFS two challenges. First, the metadata service must host a huge amount of metadata information. Second, it also has to support high-performance metadata accesses.

To scale out the metadata service, more than one metadata server will be deployed to jointly host metadata and balance the workload received from clients. When designing such a metadata service, we should consider three key problems, which are the metadata partitioning, metadata indexing, and load balance strategy, respectively. Subtree partitioning is a traditional method to distribute metadata of files across multiple metadata servers at the subtree or directory granularity. The benefit of this strategy is to keep better namespace or directory locality, as the metadata of a directory and files in that directory will co-locate in the same metadata server. In addition to the subtree partitioning method, another alternative metadata management method, hash-based mapping, organizes the directories and files as a flat structure rather than a hierarchical structure. In detail, the hash-based method partitions and distributes the metadata by computing hash values of the paths or names of files and directories. Due to the randomness of hashing functions, the hash-based mapping achieves better load balance among metadata servers, compared to the subtree-based method. However, the hash-based method sacrifices the directory locality, because the metadata of files in the same directory might be placed on different metadata servers. In summary, the two traditional metadata management methods cannot achieve good spatial locality and load balance simultaneously.

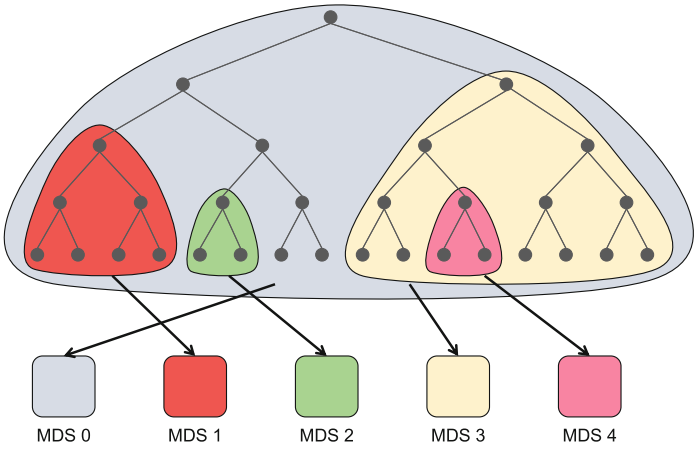


Fig. 2. Subtree partitioning metadata management.

The close the gap between maintaining good namespace locality and achieving good load balance, in this paper, we propose a hybrid metadata management method, HSM^2, which partitions metadata and balances workloads via combining both the subtree partitioning method and the hash-based mapping method. In short, this hybrid method applies different policies for namespaces at the different levels of the whole file system metadata hierarchy. For instance, with regard to directories at the lower-level of the namespace, HSM^2 will ensure their metadata integrity and keep the metadata of files under such a directory reside in the same metadata server, to keep better namespace locality. In contrast, regarding directories at the higher level of the namespace, HSM^2 adopts the hash-based mapping to randomly distribute the metadata of different directories across multiple metadata servers. To demonstrate the benefits of such a design, we implemented HSM^2 in CephFS, a widely deployed distributed file system. We conducted a comprehensive set of metadata-intensive experiments. Experimental results show that HSM^2 can achieve better namespace locality and load balance simultaneously. Compared with CephFS, HSM^2 can reduce the completion time by 70% and achieve 3.9× overall throughput speedup for a file-scanning workload.

The rest of the paper is organized as follows. We describe the background and motivation of our work in Sect. 2. Then we sketch the design and implementation details of HSM^2 in Sect. 3. In Sect. 4, we report the performance evaluation results. Finally, we conclude in Sect. 5.

2 Background and Motivation

In this section, we first introduce the two traditional metadata management methods in distributed file systems. Then we present the motivation of our work

based on results drawn from a comprehensive set of performance evaluation experiments (Fig. 3).

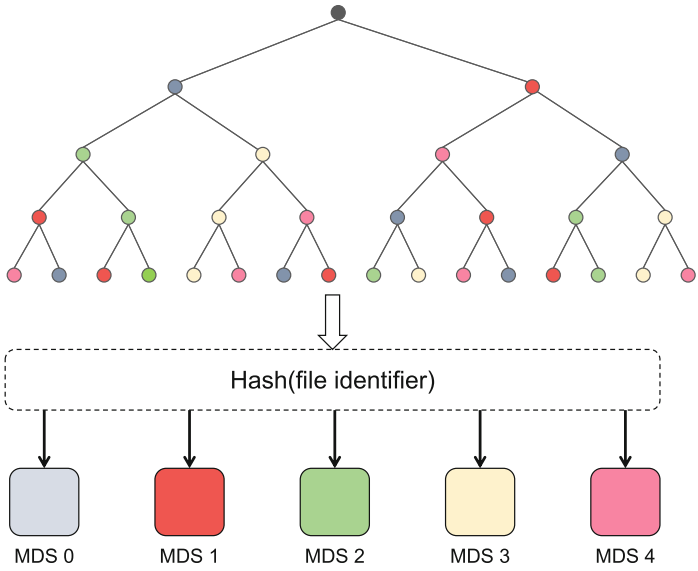


Fig. 3. Hash-based mapping metadata management.

2.1 Metadata Management Methods

Subtree Partitioning. This method splits the file system directory tree or namespace into many subtrees, as Fig. 2 shows, and then assigns subtrees to specific targeted MDS. Each MDS manages a set of subtrees. Because the metadata belongs to the same directory is assigned to the same MDS, the subtree partitioning method maintains good namespace locality. In addition, there are two variants of the subtree partitioning method, which are static and dynamic partitioning methods respectively. With regard to the static subtree partitioning method, the namespace is partitioned manually by the system administrator, like NFS [18], Sprite [17], AFS [16], Coda [21], CIFS [12] and PanFS [2]. When the system workload changes dynamically, however, this static partitioning method could introduce a load imbalance problem. To address this problem, CephFS [27] proposes a dynamic subtree partitioning method to adjust the subtree splitting granularity and migrate the metadata across MDS cluster adaptively according to the real-time workloads. But due to the variance in subtree sizes and real-time workloads, it is also challenging for this method to keep the metadata servers load-balanced.

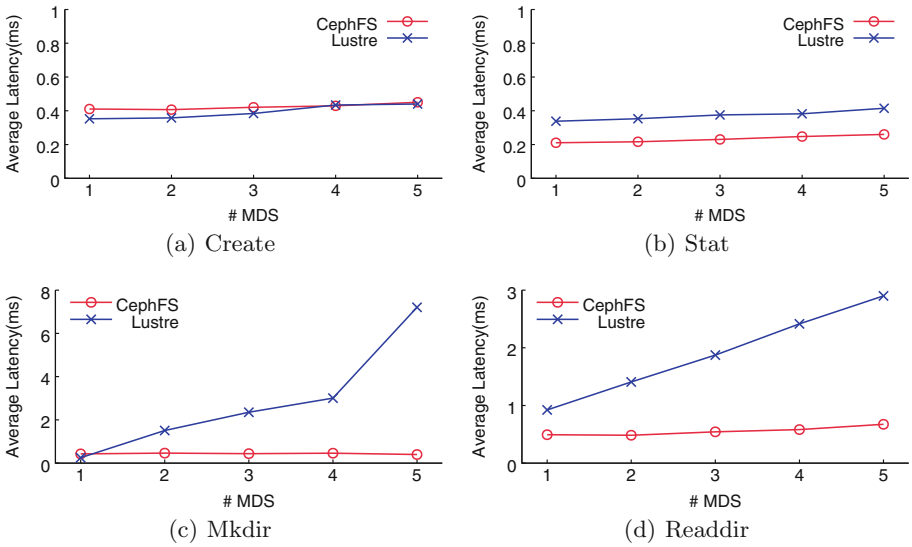


Fig. 4. File and directory metadata performance comparison under subtree partitioning and hash-based mapping metadata management methods.

Hash-Based Mapping. Unlike the subtree partitioning method, the hash-based mapping management strategy distributes the metadata according to the hash value of the unique file identifiers (i.e., file pathname or inode number). This method has been deployed in many file systems such as Vesta [9], RAMA [15], zFS [19], Lazy Hybrid [31], CalvinFS [26], SkyFS [30], Intermezzo [7], ShardFS [29], Lustre [6], and LocoFS [14]. This management method converts the file system directory tree structure into a flat namespace, thus destroying the namespace locality, despite evenly balancing workloads across metadata servers. For example, reading a directory content may need a significant number of communications between multiple metadata servers.

Next, we explore the benefits and flip-sides of both methods via running a few experiments and analyzing their results.

2.2 Namespace Locality vs. Load Balance

To understand the difference between the subtree partitioning and hash-based mapping metadata management methods deeply, we conduct a comprehensive set of experiments to evaluate the overall file/directory metadata performance and the load balance factors across multiple metadata servers. For this set of experiments, we use a popular and widely used file system benchmark, Filebench [25], to generate the metadata-intensive workload (e.g., create and stat files, create and traverse directories). The distributed file systems that we evaluate are CephFS [28] and Lustre [6], respectively, since CephFS uses the subtree partitioning method, while Lustre adopts the hash-based mapping method.

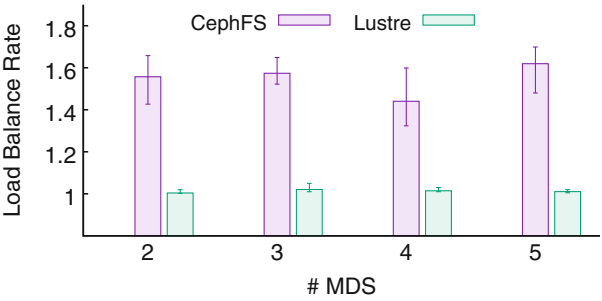
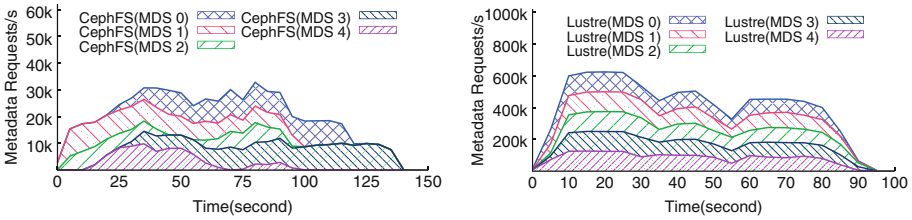


Fig. 5. Storage load balance comparison result. The load balance rate is defined as $\frac{load_{max}}{load_{avg}}$ [23].



(a) Ceph (b) Lustre

Fig. 6. Workload load balance comparison result.

Namespace Locality. Figure 4 summarizes the latency results about different metadata operations achieved by two different metadata management methods. First, as shown in Fig. 4(a), for the `create` operation, the subtree partitioning method offers similar latency numbers as the hash-based mapping one. This is because this operation only contacts a single MDS. As a result, the metadata access latency of the `create` operation remains constant when the MDS cluster becomes larger. In contrast, for directory metadata operations, the subtree partitioning method performs better than the hash-based mapping method. For example, as depicted in Fig. 4(d), reading whole directory content only needs 1ms by the subtree partitioning method when deploying 5 MDS. However, with the same setup, the hash-based mapping method needs almost 5 ms. This is because, within the subtree partitioning method, the `readdir` operations only contact a single MDS, while the hash-based mapping method may have to traverse the file metadata from all 5 MDS. Consequently, the subtree partitioning method offers better namespace locality than the hash-based mapping one.

Load Balance. In addition to namespace locality, we also evaluate the storage and workload load balance performance. For this set of experiments, we run a file creation workloads via Filebench. Figures 5 and 6 (stacked presentation) show that compared to the load balance factor of the hash-based mapping method

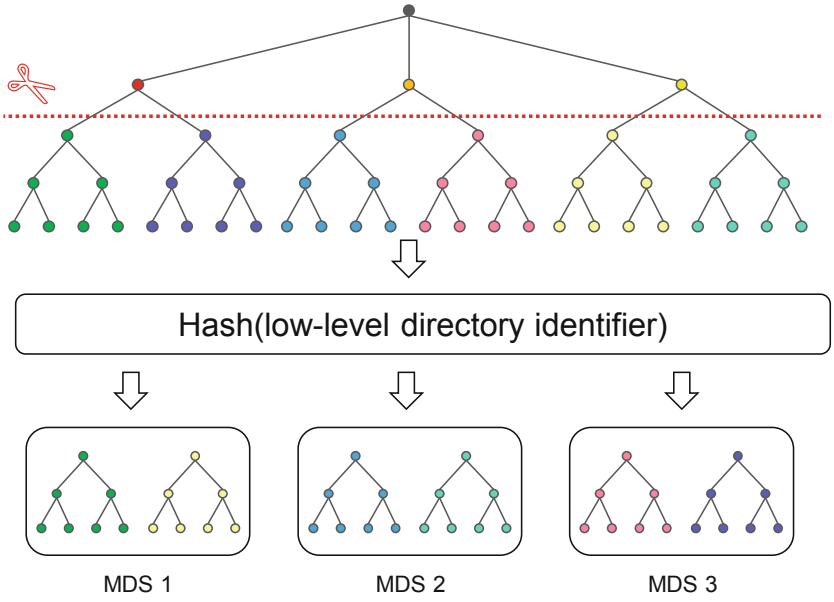


Fig. 7. HSM^2 metadata partitioning method.

outperforms the subtree partitioning method. For example, as shown in Fig. 6, the workloads were distributed evenly across 5 MDS by the hash-based mapping. However, this is not applied to the subtree partitioning method. This reason is as follows: When more workloads arrive, the migration process in the subtree partitioning method has been triggered. However, this migration does not balance well the load to all MDS, due to the complex load balance policy and metadata migration mechanism.

From these experiment results, we observe that the subtree partitioning method and hash-based mapping method cannot achieve namespace locality and load balance simultaneously. Therefore, designing an efficient metadata management method to close the gap between namespace locality and load balance is critical and challenging.

3 HSM^2

Based on the above observations and reconsideration on metadata server cluster architecture in Sect. 2, we propose an efficient metadata management method, HSM^2 to make an effective tradeoff between namespace locality and load balance. We redesign and replace the metadata management module of Ceph, using HSM^2 to determine the MDS to place instead of the original subtree partitioning method. In this section, we first introduce the metadata distribution and indexing methods, then discuss the load balance and scaling strategy.

3.1 Metadata Partitioning

HSM^2 splits the file system namespace in the middle level of the namespace hierarchy into several smaller subtrees on metadata servers to keep namespace locality, as Fig. 7 shows. Then, HSM^2 assigns the metadata of high-level subtree to the targeted MDS by applying the hash-based mapping method. Inside each MDS, HSM^2 will keep the original subtree structure unchanged. Note that the subtree splitting granularity is not fixed and HSM^2 can adaptively adjust the subtree granularity to select appropriate subtree sizes, according to the dynamic workloads. Therefore, compared to the hash-based mapping method, HSM^2 can keep namespace locality, while compared to the subtree partitioning method, HSM^2 can achieve better load balance across different MDSs.

3.2 Metadata Indexing

When a client needs to lookup metadata, it will first search the target metadata in its metadata cache. If not exist, it will send a request to a certain MDS. Different from the original subtree partition method, HSM^2 will not send a metadata request to a random MDS which does not possess the corresponding metadata. On the contrary, the client could calculate the target MDS which holds the requested metadata of the corresponding directory through a deterministic hash algorithm. By this method, unnecessary forwarding requests can be efficiently reduced. Once MDS receives the metadata request, it first checks whether the corresponding subtree of the target file already exists in the cache, and read the metadata into the cache if the check fails. Then MDS will respond to the client's request with the expected metadata.

3.3 Load Balance

For the traditional subtree partitioning method, keeping good load balance is a difficult problem to solve. In recent years, a few metadata migration methods have been proposed, but the improvement is not guaranteed. HSM^2 successfully avoids this heavy overhead because the partition method offers good load balance without sacrificing namespace locality.

3.4 Downsides

Due to splitting of subtree, the namespace locality in this level will be destroyed by HSM^2. Nevertheless, compared with hash-based mapping methods, HSM^2 successfully controlled the damage to locality to an acceptable extent. Considering the huge benefits that HSM^2 brings, we think it worthwhile.

4 Evaluation

4.1 Experiment Setup

We implement HSM^2 and integrate it with CephFS. We conduct experiments on 10 Sungon I60-G20 servers. Each server has two Intel (R) Xeon (R) E5-2650 V4

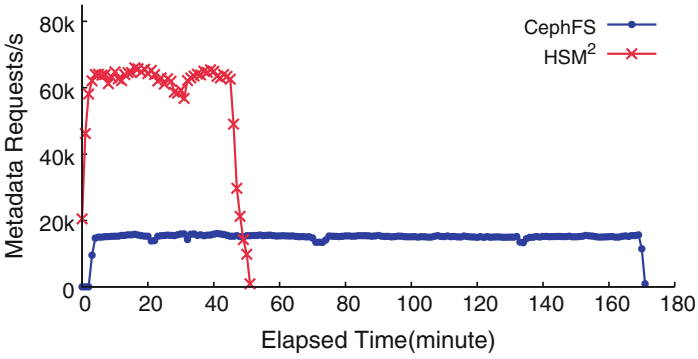


Fig. 8. The overall throughput with elapsed time.

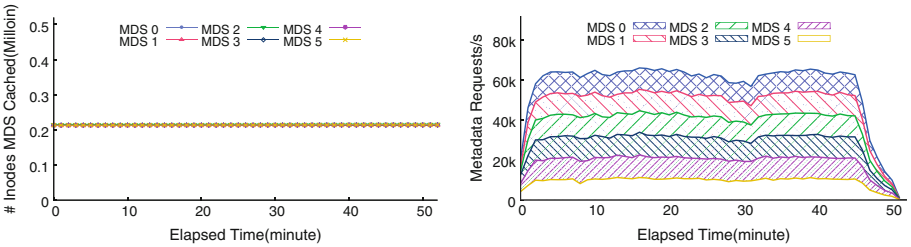
CPUs, 64 GB 2133 MHz DDR4 memory space and 12 1TB HDDs. The operating system is CentOS 7 and kernel is Linux 3.10.0-957.1.3.el7.x86_64. All servers are connected by 56 Gbps Infiniband network and the communication protocol is IPoIB. We deploy a metadata server cluster on 2 separated nodes and each of which deploys multiple MDSs. We also deploy a data server cluster on another 3 physical nodes, each of which runs 2 data servers. The remaining 5 nodes are used to deploy clients, which issue metadata requests.

We generate a directory-scanning workload in which 100 clients perform directory traversing operations on a shared dataset which consists of 10,000 directories and almost 1.2 Million files in total, to evaluate CephFS and HSM^2 metadata performance.

4.2 Experimental Results

Figure 8 shows the system overall throughput of Ceph and HSM^2 with elapsed time respectively. According to the result, we find that traversing all directory content needs almost 175 min in CephFS. But HSM^2 only needs around 53 min and can reduce the completion time by 70% compared to CephFS. Besides, HSM^2 achieves 3.9× overall throughput speedup compared to the original CephFS. Furthermore, HSM^2 can handle 57k metadata requests per second on average, but the overall throughput of CephFS is only around 14.4k.

The second primary focus of this evaluation is to understand the load balance achieved by the new design. To this end, we collect the metadata and workload distribution on each MDS as time evolves, for both CephFS and HSM^2. Figure 9 represents the metadata distribution and IOPS distribution results on each MDS per second in HSM^2. Due to the high-level hash-based mapping method implemented in HSM^2, as illustrated in Fig. 9(a), the file system metadata were distributed evenly across all five metadata servers. In addition to the storage load balance, Fig. 9(b) highlights that the workloads were also distributed more evenly across all metadata servers, compared to CephFS (see Fig. 6(a)). As a consequence, the better load balance performance enables more metadata operations



(a) Metadata distribution (b) IOPS distribution

Fig. 9. Workload load balance comparison result.

to be processed in parallel, shortens the completion time and improves the overall metadata performance.

5 Conclusion

In this paper, we proposed a hybrid and scalable metadata management method for distributed file systems, HSM^2, which combines the subtree partitioning and hash-based mapping method together to maintain both namespace locality and load balance. We implemented HSM^2 atop of CephFS. The experimental results show that, compared to CephFS, HSM^2 can reduce the completion time of a representative file-scanning workload by 70% and achieve 3.9× overall throughput speedup.

Acknowledgement. This work is supported in part by National Key R&D Program of China under Grant No. 2018YFB1003204, NSFC under Grant No. 61772484, and the Joint Funds of CETC under Grant No. 20166141B08080101.

References

1. Abad, C.L., Luu, H., Roberts, N., Lee, K., Lu, Y., Campbell, R.H.: Metadata traces and workload models for evaluating big storage systems. In: 2012 IEEE Fifth International Conference on Utility and Cloud Computing, pp. 125–132. IEEE (2012)
2. Abbasi, Z., et al.: Scalable performance of the panasas parallel file system. In: FAST 2008 Proceedings of the 6th USENIX Conference on File and Storage Technologies, pp. 17–33 (2008)
3. Alam, S.R., El-Harake, H.N., Howard, K., Stringfellow, N., Verzelloni, F.: Parallel I/O and the metadata wall. In: Proceedings of the Sixth Workshop on Parallel Data Storage, pp. 13–18. ACM (2011)
4. Anderson, E.: Capture, conversion, and analysis of an intense NFS workload. In: FAST, vol. 9, pp. 139–152 (2009)
5. Beaver, D., Kumar, S., Li, H.C., Sobel, J., Vajgel, P., et al.: Finding a needle in haystack: Facebook's photo storage. In: OSDI, vol. 10, pp. 1–8 (2010)

6. Braam, P.: The lustre storage architecture. arXiv preprint arXiv:1903.01955 (2019)
7. Braam, P., Callahan, M., Schwan, P., et al.: The intermezzo file system. In: Proceedings of the 3rd of the Perl Conference, O'Reilly Open Source Convention (1999)
8. Carns, P., Lang, S., Ross, R., Vilayannur, M., Kunkel, J., Ludwig, T.: Small-file access in parallel file systems. In: 2009 IEEE International Symposium on Parallel & Distributed Processing, IPDPS 2009, pp. 1–11. IEEE (2009)
9. Corbett, P.F., Feitelson, D.G.: The Vesta parallel file system. ACM Trans. Comput. Syst. (TOCS) **14**(3), 225–264 (1996)
10. Ghemawat, S., Gobioff, H., Leung, S.T.: The Google file system. In: ACM SIGOPS Operating Systems Review, vol. 37, pp. 29–43. ACM (2003)
11. Harter, T., et al.: Analysis of HDFS under HBase: a Facebook messages case study. In: FAST, vol. 14, p. 12 (2014)
12. Hertel, C.R.: Implementing CIFS: The Common Internet File System. Prentice Hall Professional, Upper Saddle River (2004)
13. Leung, A.W., Pasupathy, S., Goodson, G.R., Miller, E.L.: Measurement and analysis of large-scale network file system workloads. In: USENIX Annual Technical Conference, vol. 1, pp. 2–5 (2008)
14. Li, S., Lu, Y., Shu, J., Hu, Y., Li, T.: LocoFS: a loosely-coupled metadata service for distributed file systems. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2017, Denver, CO, USA, 12–17 November 2017, pp. 4:1–4:12 (2017)
15. Miller, E.L., Katz, R.H.: RAMA: an easy-to-use, high-performance parallel file system. Parallel Comput. **23**(4–5), 419–446 (1997)
16. Morris, J.H., Satyanarayanan, M., Conner, M.H., Howard, J.H., Rosenthal, D.S., Smith, F.D.: Andrew: a distributed personal computing environment. Commun. ACM **29**(3), 184–201 (1986)
17. Ousterhout, J.K., Cherenson, A.R., Douglis, F., Nelson, M.N., Welch, B.B.: The sprite network operating system. Computer **21**(2), 23–36 (1988)
18. Pawlowski, B., Juszczak, C., Staubach, P., Smith, C., Lebel, D., Hitz, D.: NFS version 3: Design and implementation. In: USENIX Summer, Boston, MA, pp. 137–152 (1994)
19. Rodeh, O., Teperman, A.: zFS-a scalable distributed file system using object disks. In: 2003 Proceedings of 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST 2003), pp. 207–218. IEEE (2003)
20. Roselli, D.S., Lorch, J.R., Anderson, T.E., et al.: A comparison of file system workloads. In: USENIX Annual Technical Conference, General Track, pp. 41–54 (2000)
21. Satyanarayanan, M.: Coda: a highly available file system for a distributed workstation environment. In: Proceedings of the Second Workshop on Workstation Operating Systems, pp. 114–116. IEEE (1989)
22. Schwan, P., et al.: Lustre: building a file system for 1000-node clusters. In: Proceedings of the 2003 Linux Symposium, vol. 2003, pp. 380–386 (2003)
23. Shen, Z., Shu, J., Lee, P.P.: Reconsidering single failure recovery in clustered file systems. In: 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 323–334. IEEE (2016)
24. Shvachko, K., Kuang, H., Radia, S., Chansler, R., et al.: The hadoop distributed file system. In: MSST, vol. 10, pp. 1–10 (2010)
25. Tarasov, V.: Filebench (2018). https://github.com/filebench/filebench
26. Thomson, A., Abadi, D.J.: CalvinFS: consistent WAN replication and scalable metadata management for distributed file systems. In: 13th USENIX Conference on File and Storage Technologies (FAST 2015), pp. 1–14 (2015)

27. Weil, S.A., Brandt, S.A., Miller, E.L., Long, D.D., Maltzahn, C.: Ceph: a scalable, high-performance distributed file system. In: Proceedings of the 7th Symposium on Operating Systems Design and Implementation, pp. 307–320. USENIX Association (2006)
28. Weil, S.A., Pollack, K.T., Brandt, S.A., Miller, E.L.: Dynamic metadata management for petabyte-scale file systems. In: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, p. 4. IEEE Computer Society (2004)
29. Xiao, L., Ren, K., Zheng, Q., Gibson, G.A.: ShardFS vs. indexFS: replication vs. caching strategies for distributed metadata management in cloud storage systems. In: Proceedings of the Sixth ACM Symposium on Cloud Computing, pp. 236–249. ACM (2015)
30. Xing, J., Xiong, J., Sun, N., Ma, J.: Adaptive and scalable metadata management to support a trillion files. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, p. 26. ACM (2009)
31. Xue, L., Brandt, S.A., Miller, E.L., Long, D.D.: Efficient metadata management in large distributed file systems. In: Twentieth IEEE/Eleventh NASA Goddard Conference on Mass Storage Systems and Technologies (2003)
32. Zhang, S., Catanese, H., Wang, A.I.A.: The composite-file file system: decoupling the one-to-one mapping of files and metadata for better performance. In: FAST, pp. 15–22 (2016)