Scaling Distributed File Systems via Correlation-based Metadata Prefetching

Youxu Chen(student), Yinlong Xu(co-advisor) and Cheng Li(co-advisor)

University of Science and Technology of China

1 Problem and Motivations

Distributed file systems (DFS) such as GFS[5], HDFS[14], Ceph[16] have become key data storage components to building scalable Internet services. The typical DFS architecture consists of three components, namely, metadata servers(MDS), object-based storage devices(OSD) and clients. Clients first send metadata requests directly to MDS and later on contact OSD for accessing raw data. The interaction with MDS may involve a sequence of operations to perform a lookup to locate the path info, obtain the metadata info (i.e., directory entry and inode) and require a lease to be able to manipulate the target file. A survey highlights that the number of IOs to MDS is significant and accounts for 50% in the overall IOs to the whole DFS [12]. As a result, the performance of the metadata server is crucial to scale DFS[17, 10, 18, 3, 15, 11, 1, 2, 13].

In web services, loading one web page may lead to loading multiple scripts and images, as that page contains various links referring to those required resources. For instance, we analyse the access log [19] of web servers and find out that 23% requests are correlated. The metadata server design does not respect the existences of correlations, and thus resulting in performance degradations: (a) too much traffic that could be grouped together flows to metadata server in a separating manner and may overwhelm it; (b) the sequential access also prolongs the page loading time, which is the primary concern of service providers. It is worth noting that these correlated access patterns also appear in scenarios other than web services, such as code browsing, scientific computations and so forth.

To keep MDS away from being the bottleneck, in this paper, we propose a correlations-based metadata prefetching to significantly reduce IO traffic to MDS and consequently to scale distributed file systems. To achieve this, we must overcome the following challenges:(1) extracting correlations with low false negatives where a correlation we predict never exists; (2) fast adapting to real time changes of correlations as the system is running; (3) efficiently prefetching correlated files.

2 Background and Related Work

Some previous works focus on exploiting correlated access patterns to scale storage systems. For instance, C-Miner[9] explores block correlations, and DiskSeen[4] analyzes temporal and spatial relationships of disk access. Unlike them, we instead target file correlations, which contain richer semantic. To find file correlations, Kroeger et al.[8] introduces an extended partitioned context modeling to explore correlations by building an access pattern trie. Gu et al.[6] proposes a weighted-graph-based grouping method to predict and prefetch the future access sequence. Hua et al.[7] organizes files into semantic-related groups and designs a semantic-aware caching method to offer low latency queries.

Most of them are built on a hypothesis that if two files were frequent accessed closely in the past, they will be accessed together with high probability in the future. Unfortunately, all of these approaches discard a type of correlations — data correlation, which are important and are taken into account by us. Besides, almost every piece of the above works runs the correlation extraction offline, and thus resulting in inaccuracies as correlations will be evolving. To address these problems, we plan to integrate online and offline methods to adapt to real-time changes.

3 Our Approach and Novelty

3.1 Defining Correlations

Informally, we say that file A and file B are correlated if A refers to B or they are accessed closely. This definition breaks correlations into two categories, namely data correlation and access correlation. We propose to associate the metadata of each file with a list of key value pairs where *key* is the unique inode number of one of its correlated files, and *value* consists of two double numbers, namely V_D and V_A, where the former and latter represents the likelihood of having data correlation or being accessed together. The V_D or V_A larger, the corresponding correlation stronger.

3.2 Extracting and Updating Correlations

The major challenges of exploring data correlations to scale distributed file systems is to detect these correlations and adapt them to dynamic file changes. To fulfill the goal of not impacting the performance of normal metadata IOs, we propose a hybrid solution, which can extract and update correlations when adding new files, removing existing files and changing file content, and additionally support deferred processing of such tasks.

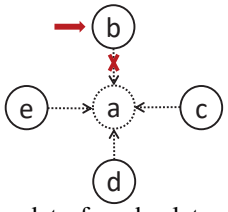

Figure 1: Lazy update for obsolete correlations when deleting files.

Online There are two major cases in which we have to extract or update correlations, namely, files got changed, and frequencies of co-accessing evolves.

(1) Content changes When clients update files data, we trigger a **semantic analysis procedure** which updates data correlations at runtime according to the content changes. To do so, we face the following challenges:

Challenge 1 The semantic analysis procedure has to handle the diverse application-specific data files. For example, scripts, images or other files are referred by web files using `href` or `src`. Source code files refer to the required header files by `include` or `import`. We plan to generalize this and make the procedure understand the formats of various applications.

Challenge 2 When a file is blindly overwritten, it is easy to find correlations pointed by new data, but an additional read to the old data is needed to remove the corresponding old correlations. To avoid this, we introduce a lazy update method, where we first mark which parts of the file were overwritten and rebuild correlations in background to keep them consistent w.r.t data changes when a subsequent read arrives.

Challenge 3 When a file is deleted, the correlations pointing to it become obsolete and should be removed. As Fig. 1 depicts, file `a` is deleted, and the metadata of the other four files must evict correlations to `a`. However, this task needs to perform a comprehensive search in MDS to figure out which files correlate with `a`, as the correlation relation is not symmetric. To reduce the overhead, we defer this update and remove obsolete correlations when MDS tells that the correlated files do not exist.

(2) Access pattern evolves To react to online changes in access patterns, we adopt a sliding window method to compute real time frequencies of file access correlations and update the access value field (Section 3.1) accordingly. To obtain frequencies, we keep track of a set of sequences of file access, where each sequence is connected to a client. The window with a given fixed size starts at the beginning of every sequence and slides towards the tail one element per time. The files in that window are considered correlated and recorded as *ordered pairs* as $< fileA, fileB >$. We aggregate the number of occurrences of each ordered pair.

Offline To avoid introducing significant overhead at runtime, we also propose a offline method to exploring correlations among files when the system is idle or under loaded. We record the access patterns for every client and find the correlations by analyzing access traces. The correlations will be then propagated to the metadata server in a light-weight manner.

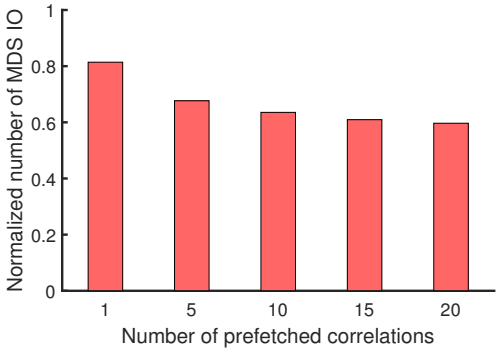

Figure 2: Normalized IO traffic to MDS vs. number of prefetched correlations

3.3 Prefetching Metadata

When correlations are in place, prefetching is straightforward to obtain the metadata of most correlated files whose values are sorted. In addition, we make two optimizations. First, in order to reduce the traffic from MDS to client and increase the client cache hit rate, we put an upper bound on the number of correlated items that can be fetched at a time. Second, to reduce the lookup and permission check overhead, we also batch the requests to obtain the ancestors information (directory and inode) of a given file when executing prefetching.

3.4 Implementation and Evaluation

We have built a prototype system based on Ceph, which demonstrates all key ideas presented above. We expect that our design will reduce IO traffic to MDS with moderate memory footprint, improves the client cache hit ratio, and makes the enhanced Ceph more scalable via correlations-based metadata prefetching.

Microbenchmark results We focus our attention on the reduction in metadata IOs in the preliminary experiments. We created a microbenchmark which contains 100000 files in a single directory, each of which has a random number of correlations w.r.t other files. In each experiment, we varied the number of prefetched correlations upon each request sent to MDS. Fig. 2 summarizes the comparison between the original Ceph and our enhanced version. Compared with Ceph, our correlation prefetching scheme could reduce up to 40% IO traffic to MDS when prefetching 20 correlations per time. This improvement can be explained by the fact that prefetched metadata resided in the clients' cache and a significant amount of subsequent accesses were likely served from the cache. This result is also supported by an increase of 10% in the cache hit ratio at the client side.

References

[1] C. L. Abad, H. Luu, N. Roberts, K. Lee, Y. Lu, and R. H. Campbell. Metadata Traces and Workload Models for Evaluating Big Storage Systems. In *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, pages 125–132. IEEE Computer Society, 2012.

[2] S. R. Alam, H. N. El-harake, K. Howard, N. Stringfellow, and F. Verzelloni. Parallel i/o and the metadata wall. In *In Proceedings of the 6th Parallel Data Storage Workshop (PDSW11)*. Citeseer, 2011.

[3] D. Beaver, S. Kumar, H. C. Li, J. Sobel, P. Vajgel, et al. Finding a Needle in Haystack: Facebook's Photo Storage. In *OSDI*, volume 10, pages 1–8, 2010.

[4] X. Ding, S. Jiang, F. Chen, K. Davis, and X. Zhang. DiskSeen: Exploiting Disk Layout and Access History to Enhance I/O Prefetch. In *USENIX Annual Technical Conference*, volume 7, pages 261–274, 2007.

[5] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. In *ACM SIGOPS operating systems review*, volume 37, pages 29–43. ACM, 2003.

[6] P. Gu, J. Wang, Y. Zhu, H. Jiang, and P. Shang. A Novel Weighted-Graph-Based Grouping Algorithm for Metadata Prefetching. *IEEE Transactions on Computers*, 59(1):1–15, 2010.

[7] Y. Hua, H. Jiang, Y. Zhu, D. Feng, and L. Tian. SmartStore: A New Metadata Organization Paradigm with Semantic-Awareness for Next-Generation File Systems. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, page 10. ACM, 2009.

[8] T. M. Kroeger and D. D. Long. Design and Implementation of a Predictive File Prefetching Algorithm. In *USENIX Annual Technical Conference, General Track*, pages 105–118, 2001.

[9] Z. Li, Z. Chen, S. M. Srinivasan, and Y. Zhou. C-Miner: Mining Block Correlations in Storage Systems. In *FAST*, volume 4, pages 173–186, 2004.

[10] S. Patil and G. A. Gibson. Scale and Concurrency of GIGA+: File System Directories with Millions of Files. In *FAST*, volume 11, pages 13–13, 2011.

[11] K. Ren, Q. Zheng, S. Patil, and G. Gibson. Indexfs: Scaling File System Metadata Performance with Stateless Caching and Bulk Insertion. In *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 237–248. IEEE, 2014.

[12] D. S. Roselli, J. R. Lorch, T. E. Anderson, et al. A Comparison of File System Workloads. In *USENIX Annual Technical Conference, general track*, pages 41–54, 2000.

[13] M. A. Sevilla, N. Watkins, C. Maltzahn, I. Nassi, S. A. Brandt, S. A. Weil, G. Farnum, and S. Fineberg. Mantle: a Programmable Metadata Load Balancer for the Ceph File System. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 21. ACM, 2015.

[14] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The Hadoop Distributed File System. In *2010 IEEE 26th symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10. IEEE, 2010.

[15] S. Sinnamohideen, R. R. Sambasivan, J. Hendricks, L. Liu, and G. R. Ganger. A Transparently-Scalable Metadata Service for the Ursa Minor Storage System. In *USENIX Annual Technical Conference*, 2010.

[16] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn. Ceph: A Scalable, High-Performance Distributed File System. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, pages 307–320. USENIX Association, 2006.

[17] S. A. Weil, K. T. Pollack, S. A. Brandt, and E. L. Miller. Dynamic Metadata Management for Petabyte-Scale File Systems. In *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, page 4. IEEE Computer Society, 2004.

[18] J. Xing, J. Xiong, N. Sun, and J. Ma. Adaptive and Scalable Metadata Management to Support a Trillion Files. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, page 26. ACM, 2009.

[19] S. Zhang, H. Catanese, and A.-I. A. Wang. The Composite-file File System: Decoupling the One-to-One Mapping of Files and Metadata for Better Performance. In *FAST*, pages 15–22, 2016.