

Small Files Problem in Parallel File System

Xiuqiao Li, Bin Dong, Limin Xiao, Li Ruan, Yi Ding

State Key Laboratory of Software Development Environment

School of Computer Science and Engineering, Beihang University

Beijing, China

{xiuqiaoli,bdong,georgedingyi}@cse.buaa.edu.cn,{xiaolm,ruanli}@buaa.edu.cn

Abstract—Parallel file systems suffer performance degradation of small files generated by many workloads. We provide the systematic analysis of the small files problem from aspects of both parallel file system architecture and workload characteristics. Based on the analysis, we discuss the deficiencies of current optimizations and propose several open questions and possible research directions to address the problem.

Keywords—parallel file system; small files problem; access pattern; optimization

I. INTRODUCTION

Parallel file system is one of the most important layers in parallel I/O stack of large computing systems. In typical design of parallel I/O system, parallel file system hides the diversity of underlying storage system and acts as the portal for applications to utilize backend storage. However, most of modern parallel file systems are employed a one-size-fit-all solution to serve application I/O [1]. Hence, it hinders the opportunity of tuning performance for specific workloads. One of the scenarios is the requirements of performing I/O on large amount of small files in a parallel file system. As current parallel file systems are mainly designed for large file accesses, both the architecture and I/O processes are introducing overheads for small files.

Traditional research on I/O access patterns in large computing systems reflects the workloads of scientific applications are dominated with large file I/O, while the small files have little contribution to overall I/O. This is due to the small files are mainly composed of parameter files or human readable output results [2]. However, recent studies on scientific workloads exhibit significant changes compared with previous results. One of the changes is the increase of the small files proportion in scientific workloads. Results of some studies that can draw this conclusion are as follows.

- The study in 2007 at Pacific Northwest National Laboratory showed 58% of the 12 million files on a file system were under 64 KB [3].
- Wang et al. [4] studied the distribution by file sizes in Lustre file system of cluster located at LLNL. The results showed that 30% of files in thirty-two file servers are smaller than 1 MB during the science runs phase.
- Bonfield et al. [5] stated in their study that up to 30

million files averaging 190 KB generated by sequencing the human genome.

On the other hand, many computing systems are serving more applications from different fields (such as home, enterprise and scratch storage) rather than scientific-only field. Furthermore, this trend becomes inevitable with the development of new computing paradigms, including grid and cloud computing. Hence, parallel file systems are expected to experience more different I/O access patterns. For example, user home directories usually have tons of small files and access pattern is diverse and random. Cho et al. [6] studied the files in scratch storage under two parallel file systems for and observed that over 55% of files were smaller than 1 megabyte in one system, while this proportion is over 83% in another system.

Serving for small files in above environments presents new challenges for parallel file systems. Although some studies proposed some methods to improve small file performance, the small files problem is still not well understood for applications from different fields. In this paper, we gave the systematic analysis of small files problem in parallel file system and discussed possible optimizations and open questions. The analysis covers the aspects of both parallel file system and workloads characteristic. On one hand, we analyzed possible bottlenecks for small files introduced by the typical parallel file system architecture. On the other hand, we summarized the access patterns of five small files workloads in different dimensions. Based on the analysis, we concluded the state-of-art optimizations for small files and recognized the gaps between these methods and the small files problem.

Our study is funded by a project of national natural science foundation of China, which aims to devise new methods to optimize small file performance in parallel file system. As the first step in this direction, we recognized the problems need to be addressed and discussed possible methods for these problems in this paper.

II. THE PARALLEL FILE SYSTEM

Parallel file system provides a single system image (SSI) from user's view, which can be mounted on multiple data clients simultaneously. Files are striped over selected data servers with the purpose to achieve high throughput by

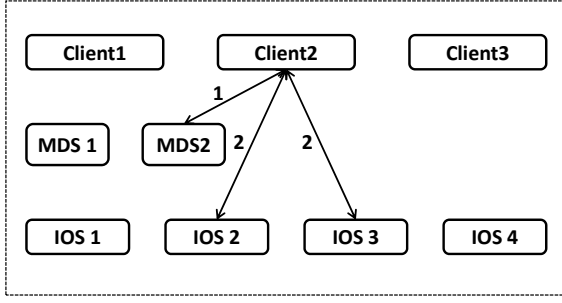


Figure 1. Parallel file system architecture

increasing parallelism. In order to support file striping, modern file systems, such as PVFS2 [7] and Lustre [8], exploit a similar client/server architecture, which divides file system nodes into three roles: client, metadata server (MDS) and I/O server (IOS). Figure 1 shows the typical architecture of parallel file systems.

A. Analysis of file I/O in parallel file system

The metadata servers maintain the directory hierarchy and file attributes for parallel file system. The storage of metadata is logically decoupled from data storage, which striped over multiple data servers. Hence, when clients received I/O requests from interface layer, file system needs to first fetch the file distribution information from the metadata server and then contact data servers to read or write data. Hence, the I/O processes in parallel file system have significant differences compared with local file system. The total cost T_{total} of an I/O request can be described as follows:

$$T_{total} = T_{meta} + \max(T_{io,1}, T_{io,2}, \dots, T_{io,n}), \quad (1)$$

where T_{meta} is the time of accessing metadata composed of one network round and the cost of reading metadata on MDS and $T_{io,i}$ represents the time of accessing the i th IOS, which given by:

$$T_{io,i} = N_{round} \bullet T_{net-round} + T_{datafile-io}; \quad (2)$$

where $T_{net-round}$ is the cost of network round-trips and $T_{datafile-io}$ is the I/O cost on the IOS node.

Compared with requests of local file system, I/O performance on large files can be improved by accessing multiple data objects in parallel. This is because the I/O request gains more in parallelism than the overhead introduced by accessing metadata and file striping. Following equation describes the cost of accessing a large file T_{large_total} and the comparison of accessing the same file in local file system, which represents as T_{io} .

$$T_{large_total} = T_{meta} + \max(T_{io,1}, T_{io,2}, \dots, T_{io,n}) \ll T_{io}. \quad (3)$$

B. Bottlenecks for small files

Differ from large files, however, the parallelism cannot substantially improve the throughput of small file I/O, and even degrade the performance on the contrary. Following equation gives the total cost T_{small_total} of accessing a small file,

$$T_{small_total} = T_{meta} + \max(T_{io,1}, T_{io,2}, \dots, T_{io,n}). \quad (4)$$

1) *Metadata accesses*: File metadata in parallel file system is composed of common file attributes and the file distribution information, such as the location of data objects and its distribution layout. Therefore, the metadata involves in most of I/O operations as file system needs the distribution information to locate the file data. However, the distributed architecture introduces overhead for the clients to access metadata. First, as the metadata is stored on a separate server, clients need one more network round-trip before issuing requests to access file data. For small files, this overhead is amplified since the gains from accessing file objects are limited. Second, many parallel file systems adopt multiple MDSs for load balancing purpose. However, the distribution of files introduces overheads for lookup files that stored in deep directories. This is due to the file system may need more round-trips to locate such files. Third, concurrent metadata operations exhibit many small file accesses in underlying storage, making the MDSs hot spots, especially with metadata-intensive workloads. The above overheads will significantly increase the cost of T_{meta} and degrade the I/O performance eventually.

2) *Data accesses*: With the data I/O parallelism, the I/O processes are split into multiple overlapped processes on different servers. However, as we can see from 3, there are overheads to setup the process of parallel I/O. Performance can only be speedup when the gains surpass the costs of these overheads. For small files, since the total data amount for each request is smaller than the large file I/O, the extra round-trips before the parallel I/O process are too expensive and unnecessary when only one IOS is involved in the I/O process, which should be the common case in practice. On the other hand, the small files are split into much smaller datafiles on different IOSs. Accessing small datafiles makes the further performance degradation on each IOS.

III. WORKLOAD CHARACTERISTICS

This section presents the characteristics of several small file workloads in parallel file system. We first define a classification of typical access patterns for small files and then analysis different workloads using the classification.

A. Classification of file access patterns

Understanding file access patterns is the preliminary work to tune performance for small files. Based on the previous studies, we classify the file access patterns for small files in the following dimensions.

1) *File size distribution*: The file size in workloads depends on the program styles of specific applications. For example, some checkpoint applications choose shared file patterns for program convenience, resulting a large file shared by thousands of processes [9]. On the contrary, physics applications tend to create for every atomic particle and produce lots of small files [10]. Recognize the file size distribution of a workload can help administrators make decisions to tradeoff I/O performance.

2) *Commonality*: This factor determines the data similarity of files in a workload. The higher commonality, the more potential savings in storage space can be achieved by exploiting content-addressed storage methods [11].

3) *File lifetime*: Files can be created for storing temporary or permanent data. The live time of a file can guide the administrator to choose a proper storage device to store the file. For instance, the temporary file can be stored in memory or flash-based devices to avoid unnecessary disk overhead.

4) *File open pattern*: Application developers determine the access modes for file by specifying parameters in file open call. Common modes include read-only (R), write-only (W), read/write (RW) and append-only (A). This pattern exhibits the semantics of accessing files.

5) *Request size*: Requests of different applications has various request size. Even the size changes during the applications are running. Small request sizes will cause many discrete and small accesses on IOSs, degrading I/O performance. Large request size results larger, sequential disk accesses which have better performance.

6) *File sharing*: This factor reflects the access behaviors on files among different processes. Weak sharing pattern means the file data has strong locality and optimizations like caching or prefetching can be employed to improve performance with little overhead to maintain consistency. On the contrary, strong sharing pattern needs more efforts to keep file data consistency among concurrent processes.

7) *Request frequency*: The requests of an application can be arrived with burst or sparse pattern. The degree of burstiness generates different pressure on storage servers. For example, many checkpoint applications create large amount of files at the same time. The burstiness of metadata operations significantly affects the performance of MDS.

8) *Directory pattern*: Files are maintained and constructed in directories. The directory size and directory depth are the two important factors which influence the performance of directory operations, such as listing directory, and lookup files.

B. Workloads characteristics

We conclude and analysis the characteristics of four typical workloads in parallel file system according to the dimensions discussed above. The analysis focuses on the access patterns of small files in different workloads. Table I gives the comparison of workload characteristics for five

typical workloads. To be noticed, even for applications that belongs to the same workload, the access patterns may be different on some dimensions. The results in Table I are concluded based on a large broad of previous studies on general characteristics for different workloads [12]–[16].

1) *Scientific workload*: Most of large computing systems are mainly served for scientific applications from various fields. The access patterns are studies by many researchers [2], [4], [12], [13]. According to these studies, the proportion of small files are significant larger than large files, while almost all of the I/O data are transferred by requests on large files. Besides, the small files are usually created at burst and exhibit metadata-intensive pattern. Since scientific applications involve hundreds or thousands processes, the metadata accesses produce high pressure for metadata server. In addition, large amount of small files will increase the complexity of metadata management and the capacity of metadata storage. For data I/O, many scientific applications generate small unaligned accesses on small files, significantly impacting file performance. For directory pattern, scientific applications often place many files in a single directory, requiring efficient methods to manage large directories [9], [10].

2) *Home workload*: Many computing systems currently provide support for storing files related to users in their home directory. The user files are mainly composed of large portion of small files and deep directories. In this case, it need parallel file system to provide efficient search methods to reduce latency of iterating large amount of small files. The workload shows high locality and lower sharing pattern for I/O requests. This is due to the small chance of accessing files of other users for possible security constrains. Therefore, file systems should provide optimizations for small files to benefit from the weak consistency exhibits by the workload.

3) *Enterprise workload*: This workload can be encountered from file systems served for large corporate or engineering organizations. Leung et al. [14] studied two large CIFS file systems employed in the NetApp corporate headquarters. According to their study, the workload consists of over 50% small files, while most of the I/O comes from large files. This pattern is similar with the one in scientific workload, although the data amount of enterprise workload is smaller by many orders of magnitude. Similar with home workload, files are infrequently being shared by multiple clients. There is no studies on the commonality and directory pattern of this workload.

4) *Web workload*: Many cluster systems are served as Web servers to satisfy large amount of accesses. The workload has high access locality and 10% of files accessed account for 90% of the requests [4]. Moreover, the requests is dominated by read accesses and most of accesses can hit in cache maintained by the Web server. Therefore, the problem of this workload is how to effectively store high volumes

Table I
ACCESS PATTERNS OF SMALL FILES WORKLOADS

Workload	Small files	Commonality	File lifetime	File open pattern	File sharing	Request size	Request frequency	Directory pattern
Scientific	Many	low	Various	RW>R>W	high write	Small/Large	Burst	Flat, large
Home	Most	NA ¹	Long	R	Low	Small	NA	Deep, small
Enterprise	Many	NA	Long	RW>R>W	Low	Small	Sparse	NA
Web	Many	High	Long	R	high read	Small	NA	NA
Cloud	Many	NA	NA	R, most A	Low	Large	Burst	NA

1. NA represents the pattern is unknown

of data rather than accelerating accesses. Since the Web data exhibits high commonality, content addressable storage methods can be employed to increase storage efficiency.

5) *Cloud workload*: With the advent of cloud computing, file systems are expected more frequently to experience this kind of workload. Early studies of cloud file systems focus on increasing the computing ability by exploiting techniques like Map/Reduce, big table. However, recently studies show that some applications generate many small files due to their access patterns [16], [17]. Small files cannot benefit from the performance advantages of file system. Moreover, the management of large amount of small files increase the burden of namenode (metadata server) [17].

IV. OPTIMIZATIONS AND DISCUSSION

A. Current optimizations

Recently, many works focus on the small files problem in parallel file system. We divide these studies into two different categories: architecture optimizations and workload optimizations. The first category targets to optimize the file operation of small files from architecture view.

Limiting file stripe count: As we discussed earlier, unnecessary stripes degrade the I/O performance for small files. Some file systems provided hints to let user adjust the stripe count for files. However, manually setting hints for files increased the burden of administrator. Further, it was unpractical to decide the optimum stripe count for a file. Metadata stuffing was the common small files optimizations in local file systems. Shaikh et al. [18] presented an implementation of file stuffing on metadata server. The problem was that stuffing increases space pressure of metadata server as more and more small files are stuffed. Carns et al. [19] implemented their stuffing method in PVFS2. Compared with the method in [18], their method only worked when the metadata server acts as data server at the same time. However, only files that smaller than stripe size can benefit from their optimization.

Distinguish I/O paths: Some methods distributed I/O requests across different path according to file size or request size. EMC Celerra HighRoad [20] and pNFS [21] were the two typical systems of this kind. The former one enabled seamless integration of NAS and SAN into one unified storage network, and dynamically chosen LAN or SAN routes for optimum performance. The latter one was

a parallel extension of NFS that provides direct storage access to other parallel file systems while preserving NFS to aggregate small I/O accesses. Hildebrand studied a pNFS implementation with PVFS2 in [16]. The benefits were exploiting and utilizing good performance of different storage ways. However, these methods did not optimize the storage manner of small files. Therefore, the file data were still spited into small stripes among multiple data servers.

Bridging the gaps between clients and servers: Hendricks et al. [22] studied the overhead due to the decoupling of clients and servers in parallel file system. They presented metadata prefetching method to mask the network round-trip. Since the metadata is very small, the optimization can bring significant performance gains.

The second category tends to optimize small files according to workload characteristics.

Directory-based optimization: Kuhn et al. [23] observed that not all applications require complete metadata for small file workloads. They implemented a special directory hint *no_metadata* to label the directory which contains small files. Then, for these small files, there was no need to create the metadata and only one datafile was created to store file data. Thus, clients can get the metadata and file size with one round message exchange. In the method, the small files terms to files with a size of smaller than 1 megabyte. This method shows significant metadata performance improvements for small files but the directory restriction limits its applying scope to directory level.

Metadata compression: Mackey et al. [17] studied the small files problem with single metadata server in the context of HDFS. They proposed a harball compress method to store small files in HDFS efficiently and improved the space utilization for metadata. However, the metadata read performance was hurt due to extra steps of lookup file in harball.

B. Open questions

Although current optimizations for small files can improve file performance on some aspects, the small files problem is still not well addressed. In this section, we discuss some open questions recognized from our analysis on parallel file system architecture and workload characteristics.

1) *How to efficiently and comprehensively trace and analyze the access patterns of small files in typical workloads?*

How can quantitatively represent and utilize the access patterns of the small files workload?: As we discussed in Section III, current studies on workload characteristics cannot cover all aspects of small files access patterns. The details of access patterns remain unknown by the community, especially for new workloads, such as cloud workload. On the other hand, there is no study on the performance effect introduced by tracing requests. For small files, the trace overhead may be too large to be acceptable.

The workload characteristics is various among different workloads, even in the same workload. It would be beneficial to devise signatures for small files access patterns to be recognized and guide optimizations selections at runtime.

2) *How to collect the provenance of small files? How can we benefit from the provenance?:* Provenance contains the semantic information of the files, such as who created the file, or what the relation with other files [24]. The provenance of files is not available from current studies on workload characteristics. Understanding this information can help the administrators to tune performance for certain files or accelerate query operations. However, there is no consensus on how to collect this information and how to use it to optimize performance.

3) *How to reduce the overhead introduced by content addressable storage?:* The content addressable storage (CAS) can exploit the commonality of files to significantly save the storage space. However, the method generates many small chunks and small requests in storage. Furthermore, the get/put interface introduces the query overhead, especially when in large file systems.

4) *How to manage large amount of small files under different directory patterns?:* Based on our analysis, different workloads exhibit various directory patterns, which require specific optimizations to tune performance. For directories with large depth, efficient lookup structures need to be adopted to improve performance. For large, flat directories, the optimizations should focus on increasing the metadata scalability and aggregate performance.

5) *How to tradeoff between metadata compression ratio and performance?:* For metadata management, the optimizations should tradeoff the compression ratio and performance. In order to mask the overhead of lookup file and decompression, efficient prefetching method need to be devised and utilize fast storage devices, such as SSD, to bridge the gaps. Also, the cost model needs to be setup and evaluate the prefetching effect.

V. CONCLUSION

In this paper, we studied the small files problem in parallel file system. We first detailed the I/O path for small files accesses and analyzed the architecture bottlenecks for small files in typical design of parallel file system. Our analysis covers the performance of both metadata and data I/O for small files. Second, we defined a classification of access

patterns for small files and then concluded the patterns of small files in five workloads using the classification. Then, we surveyed and discussed the deficiencies of small files optimizations proposed by previous studies. Finally, based on the analysis, we presented several open questions for small files problem in parallel file system and discussed possible research directions. In future, we plan to study new methods to address these problems recognized in this paper.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China under Grant No.60973007, the Fundamental Research Funds for the Central Universities under Grant No. YWF-10-02-05, the Doctoral Fund of Ministry of Education of China under Grant No. 20101102110018, and the fund of the State Key Laboratory of Software Development Environment under Grant No. SKLSDE-2009ZX-01.

REFERENCES

- [1] S. Al-Kiswany, A. Gharaibeh, and M. Ripeanu, "The case for a versatile storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, pp. 10–14, March 2010.
- [2] E. L. Miller, "Input/output behavior of supercomputing application," Berkeley, CA, USA, Tech. Rep., 1992.
- [3] Petascale Data Storage Institute. NERSC file system statistics. World Wide Web electronic publication. [Online]. Available: <http://pdsi.nersc.gov/filesystem.htm>
- [4] F. Wang, Q. Xin, B. Hong, S. A. Brandt, E. L. Miller, D. D. E. Long, and T. T. Mclarty, "File System Workload Analysis for Large Scale Scientific Computing Applications," in *In Proceedings of the 21st IEEE / 12th NASA Goddard Conference on Mass Storage Systems and Technologies*, 2004, pp. 139–152.
- [5] J. K. Bonfield and R. Staden, "ZTR: a new format for DNA sequence trace data," *Bioinformatics*, vol. 18, no. 1, pp. 3–10, 2002.
- [6] H. Cho, S. Kim, and S. Lee, "Analysis of long-term file system activities on cluster systems," *WORLD ACADEMY OF SCIENCE, ENGINEERING AND TECHNOLOGY*, vol. 60, no. 1, pp. 159–164, 2009.
- [7] "The Parallel Virtual File System," 2010. [Online]. Available: <http://www.pvfs.org>
- [8] "Lustre File System," 2010. [Online]. Available: <http://www.lustre.org>
- [9] J. Bent, G. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte, and M. Wingate, "PLFS: a checkpoint filesystem for parallel applications," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ser. SC '09. New York, NY, USA: ACM, 2009, pp. 1–12.

- [10] S. V. Patil, G. A. Gibson, S. Lang, and M. Polte, "GIGA+: scalable directories for shared file systems," in *Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing '07*, ser. PDSW '07. New York, NY, USA: ACM, 2007, pp. 26–29.
- [11] P. Nath, B. Urgaonkar, and A. Sivasubramaniam, "Evaluating the usefulness of content addressable storage for high-performance data intensive applications," in *Proceedings of the 17th international symposium on High performance distributed computing*, ser. HPDC '08. New York, NY, USA: ACM, 2008, pp. 35–44.
- [12] N. Nieuwejaar, D. Kotz, A. Purakayastha, C. S. Ellis, and M. L. Best, "File-Access characteristics of parallel scientific workloads," *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, pp. 1075–1089, October 1996.
- [13] P. H. Carns, R. Latham, R. B. Ross, K. Iskra, S. Lang, and K. Riley, "24/7 Characterization of petascale I/O workloads," in *CLUSTER*. IEEE, 2009, pp. 1–10.
- [14] A. W. Leung, S. Pasupathy, G. Goodson, and E. L. Miller, "Measurement and analysis of large-scale network file system workloads," in *USENIX 2008 Annual Technical Conference on Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2008, pp. 213–226.
- [15] A. Mahanti, C. Williamson, and L. Wu, "Workload characterization of a large systems conference web server," in *Proceedings of the 2009 Seventh Annual Communication Networks and Services Research Conference*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 55–64.
- [16] X. Liu, J. Han, Y. Zhong, C. Han, and X. He, "Implementing WebGIS on Hadoop: A case study of improving small file I/O performance on HDFS," in *Proceedings of the 2009 IEEE International Conference on Cluster Computing, August 31 - September 4, 2009, New Orleans, Louisiana, USA*. IEEE, 2009, pp. 1–8.
- [17] G. Mackey, S. Sehrish, and J. Wang, "Improving metadata management for small files in HDFS," in *CLUSTER*, 2009, pp. 1–4.
- [18] F. Shaikh and M. Chainani. A case for small file packing in parallel virtual file system. [Online]. Available: <http://www.andrew.cmu.edu/user/mchainan/FinalPaper.pdf>
- [19] P. Carns, S. Lang, R. Ross, M. Vilayannur, J. Kunkel, and T. Ludwig, "Small-file access in parallel file systems," *Parallel and Distributed Processing Symposium, International*, vol. 0, pp. 1–11, 2009.
- [20] "EMC Celerra HighRoad Whitepaper," 2001. [Online]. Available: <http://www.emc.com>
- [21] D. Hildebrand, L. Ward, and P. Honeyman, "Large files, small writes, and pNFS," in *Proceedings of the 20th annual international conference on Supercomputing*, ser. ICS '06. New York, NY, USA: ACM, 2006, pp. 116–124.
- [22] J. Hendricks, R. R. Sambasivan, S. Sinnamohideen, and G. R. Ganger. (2006) Improving small file performance in object-based storage.
- [23] M. Kuhn, J. M. Kunkel, and T. Ludwig, "Dynamic file system semantics to enable metadata optimizations in PVFS," *Concurr. Comput. : Pract. Exper.*, vol. 21, pp. 1775–1788, September 2009.
- [24] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer, "Provenance-aware storage systems," in *Proceedings of the annual conference on USENIX '06 Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2006, pp. 4–4.