Exploiting FastDFS Client-based Small File Merging

Haimeng Chen, Hua Zhang

State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, 100876, China

Abstract. How to store masses of small files is a generally acknowledged problem in industry and academia. Small file merging is the most successful strategy to solve this problem, which has been supported by many distributed filesystems, such as FastDFS. However, in FastDFS, our experiments indicate that excessively wide striping causes performance degradation, and pre-allocated space causes data loss or error. Therefore, this paper presents a client-based small file merging scheme based on the characteristics of online web application. This scheme makes full use of the resources of the web server and optimizes the small file merging with the annealing algorithm to improve the throughput. Experimental results show that the write performance are improved by 52.03% and query performance by 27.69% compared with the original FastDFS small file merging scheme.

Keywords: Masses of small files; small file merging; Client-based.

1. Introduction

The data are exploding rapidly in the background of Internet technology, cloud computing, big data and so on. There may be mass of small file mixed with large files, such as application stores, network videos. How to efficiently store mass small files is facing a huge challenge.

Known as the LOSF problem, large amount of small file storage has attracted great attention. The main methods against to the LOSF problem include changing the directory indexing mechanism [1, 2], adding Cache to store hot files [3, 4], and small file merging storage and so on. Small file merging is the most successful strategy, which has been used in Haystack [5], TFS [6], FastDFS [7], and so on. The purpose of small file merging is that multiple logical files share a same physical file.

In addition to realizing the small files merging, the filesystem need to improve the performace in different ways for different purposes. For example, some filesystems improve the performance by optimizing the indexing scheme [8, 9] and some by prefetching index files according to relevance and locality [10, 11]. These methods service well in their application. However, they can't avoid managing the index files which are always stored in memory in order to access quickly, and therefore significantly limited by memory resources, memory swapping.

In consideration of the above-mentioned problems from index files, FastDFS has the better implementation in small file merging without index information. Nonetheless, FastDFS leads to other problems. Excessively wide striping causes performance degradation, because each small file requires a full TCP/IP roundtrip [12]. And pre-allocated space causes data loss or error, because Trunk Server may fail and restart.

Aiming to solve problems mentioned above, we propose a client-based small file merging according to the characteristics of data from online service websites. Small files can be merged early in the Client, which reduces the number of network connections between the Client and the Tracker Server, the Client and the Storage server, and reduces the management complexity of the Storage Server, thus improves the service performance of the file system. The experimental results show that the write performance and the query performance are improved by 52.03% and 27.69%, respectively compared with the original FastDFS small file merging scheme.

The overall structure of the paper is as follows: Section 2 will describe the architecture of Client-based small file merging and its key design principles. In the 3rd section, we describe our experiments and the performance results. We discuss future work and conclusion in Section 4.

2. Small File Merging Based-on Client

This part will introduce the Client-based small file merging from two aspects: the overall architecture and the Client's operations. The overall architecture includes the key design principles, and the Client's operations include file writing, file reading and file deletion on the Client.

2.1 the Architecture

FastDFS is particularly suitable for online services that always need to handle many medium or small files, such as album sites, video sites. Fig. 1 shows the basic structure of web site application working with FastDFS: the users visit the web site through the browser and the web server provides services. In addition, FastDFS expands the web server's storage business that stores files in Storage Cluster. As the consequence, the web server acts as not only the web server providing services but also the filesystem's Client. What's more, the returned FileIDs are always stored in the web server through database.

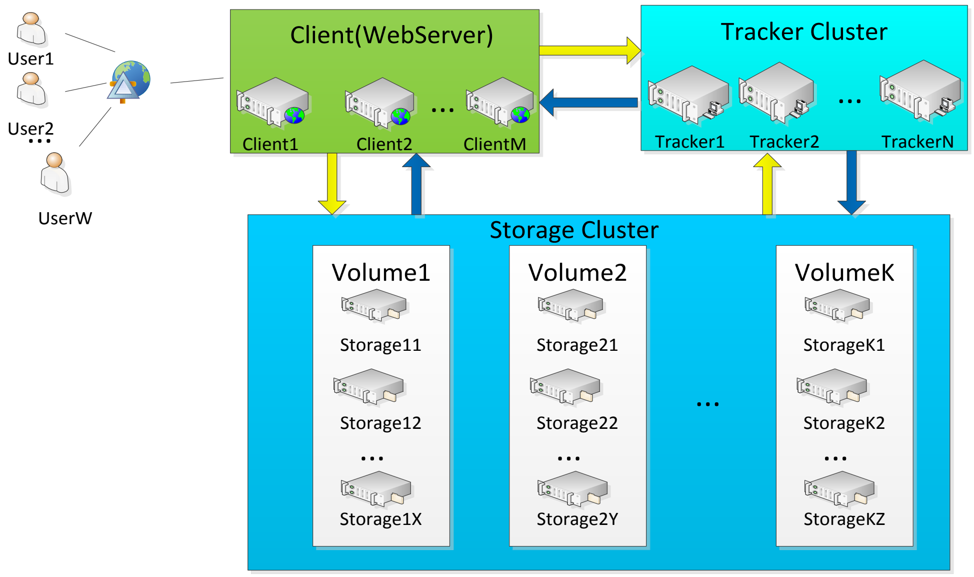

Fig. 1 Web application's structure with FastDFS

In the above scenario, the web server receives a large number of files from different users, which may contain many small files that need to be stored through file merging. If we implement the small file merging in the Client, there is no need to allocate the Trunk Server, pre-allocate spaces, and maintain free balance trees and so on. On one hand, this can reduce the complexity of writing, and then improve storage efficiency. On the other hand, it can also avoid data errors or data loss caused by pre-allocated space. Furthermore, the multiple small file merged into a large file can not only reduce the number of network connections between Client and Tracker, Client and Storage, but also reduce the small file access disk I/O times, which can improve the throughput.

2.2 the Client's operations

Client-based small file merging solution is to improve the performance of original FastDFS by altering the operations in Client. The following are the detail descriptions on changes while file writing, file reading and file deletion.

Fig. 2 shows the small file writing process on Client. The first step is to store the file and its metadata. The file data should be stored in Files Container, a space prepared for files on Client disk and metadatas stored in memory in the form of a hash table File Map. For web users, the file storage process has been completed and the file is already in a readable state which can concluded in the file reading process described later. The second step is to do small file merging. Merge Task is a timed merging program that dynamically changes the merging operation interval based on the file flow rate into the Files Container to ensure that there are enough samples to achieve the optimal merging. The optimal merging here means that the size of the final merged file is closest to the value of the memory block (default 64M), which is implemented by the simulated annealing algorithm [13]. The client merges the small files into large files marked with TaskId and records the merge order in memory. In the rest steps, the Client writes large file to Storage Server and receives the fileID of large file. Then, calculates every the small file's fileID with the returned fileID of large file and record of the merging order. At last, stores the fileIDs.

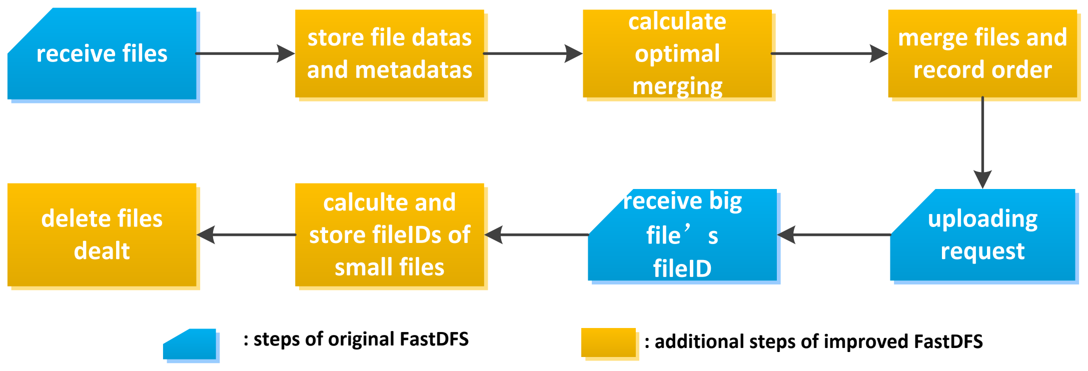
receive files
store file datas and metadatas
calculate optimal merging
merge files and record order
uploading request
receive big file's fileID
calculte and store fileIDs of small files
delete files dealt
: steps of original FastDFS
: additional steps of improved FastDFS

Fig. 2 File writing operations on Client

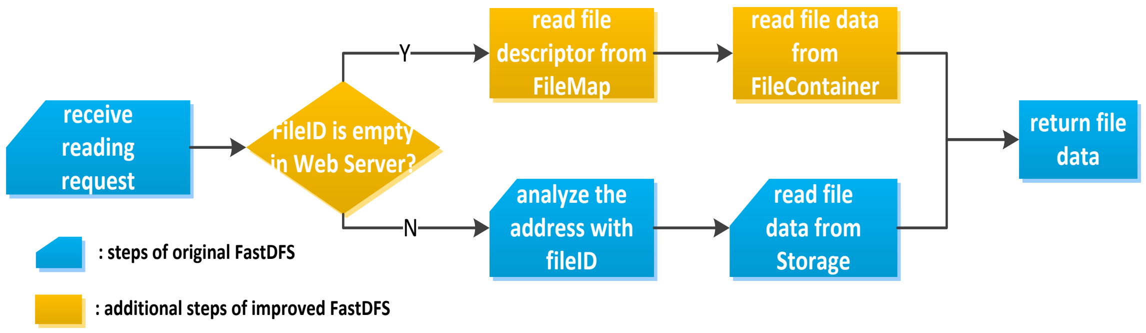
receive reading request
FileID is empty in Web Server?
Y
read file descriptor from FileMap
read file data from FileContainer
N
analyze the address with fileID
read file data from Storage
return file data
: steps of original FastDFS
: additional steps of improved FastDFS

Fig. 3 File reading operations on Client

Fig. 3 shows the small file reading process on Client. When the Client receives a request to read a file, the first step is to determine the FileID. If FileID is empty, the file is still stored on the web server (Client). The Client program reads the file metadata from the hash table FileMap to obtain the file access handle, and then accesses the file in the File Container to return the contents. If a FileID exists, the file address is parsed from the FileID and the corresponding file is read from the FastDFS storage system just like the original steps.

File deletion is simple and direct. The Client receives the request of the deleted file, reads the file according to the above-mentioned step of reading, and then deletes the file and the corresponding metadata information. It should be noted that, in such scenario, the file operations are mainly write and read, as mentioned in Haystack [7], there are few deletion operations. Therefore, there is only a simple implementation of the deletion operation in this paper and it is left to be further studied in deletion operation.

3. Evaluation

In this section, we design an experiment to test our small file merging solution. First, we set up a test environment as shown in Fig. 1, FastDFS Client and Web server are set up on the same machine. The number of Tracker server and Storage server is two. Each server is Ubuntu14.4 system, 8G memory, 1T hard drive capacity.

There are two test groups, to test write efficiency and query efficiency. File operations include writing (uploading), reading (downloading), deleting (because the deletion is not the focus of this paper, so there is no test here). We test about 500,000 files, the file size ranges from a few KB to about 100MB.

Our write experiments are performed on three conditions: the first, the original FastDFS without small file merging; the second, the FastDFS with original small file merging solution; the third, the improved FastDFS with Client-based Small File merging solution. We use writing speed to indicate write performance. Table 1 lists the first set of data results, and Fig. 4 shows a more intuitive results.

Table 1 Writing speed

concurrence	1	5	20	50	100	200
no merging (M/s)	4.46	3.89	2.92	2.48	2.34	2.11
original merging(M/s)	6.38	5.74	5.09	4.92	4.84	4.83
Client-based merging(M/s)	7.72	7.56	7.51	7.43	7.37	7.33

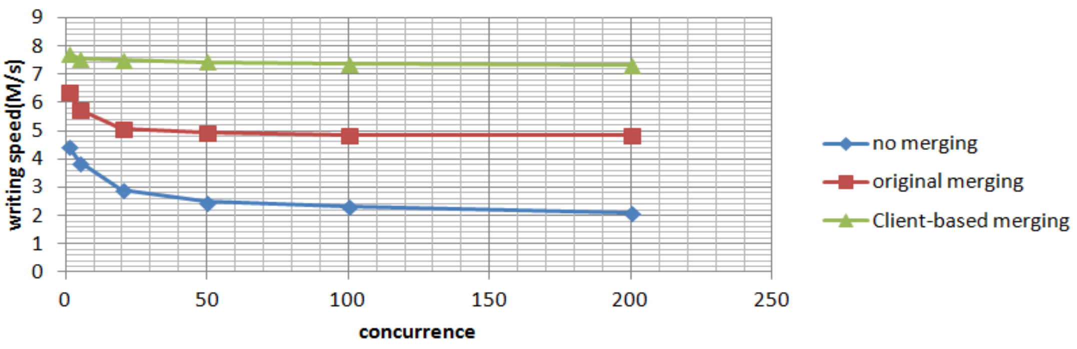

Fig. 4 Writing speed

Through the results, the speed is enhanced twice after the original FastDFS opens merging function, because merging function let multiple small files share the same trunk, and reduce the overall disk I/O times. And it is increased by another 52.03% with the Client-based solution which has avoided excessive network connections and transmissions.

For read operation, we also test in three conditions mentioned above. We use Query Rate Per Second (QPS) to indicate read performance. Table 2 lists the second set of data results and shows more intuitive results in Fig. 5.

Table 2 Query Rate Per Second

concurrence	20	50	100	200
no merging(fetches/s)	42.58	40.32	34.67	30.43
original merging(fetches/s)	95.40	93.88	85.65	78.36
Client-based merging (fetches/s)	115.26	114.62	112.67	108.58

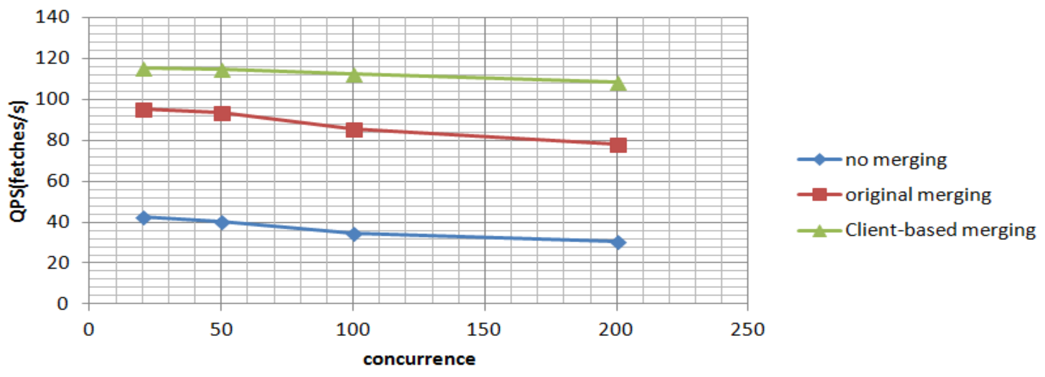

Fig. 5 Query Rate Per Second

Through the results, we can find that small file merging can improve the efficiency of querying. There just needs to access the disk file in one time once they are in the same trunk when accessing multiple files. Therefore, the original merging storage per second query rate is about 1.4 times than no merging. In Client-based resolution, files can be read from Clients directly if hit on Client, so it is improved by another 27.69%.

In summary, this paper has presented a Client-based small file merging solution in dealing with mass small files storage, and it has a better write and read performance than the original system.

4. Conclusion

In this paper, we have proposed a Client-based merged storage solution for solving the problem that the throughput of the distributed file system is limited. Experiments show that it greatly improves the FastDFS throughput. In addition, our merging storage scheme does not need pre-allocated space, so it is theoretically possible to avoid the problem of data loss or file error caused by pre-allocated space. The reproduction and demonstration of pre-allocated problem is on its way to our future work.

Acknowledgments

This work is supported by NSFC (Grant Nos. 61300181, 61502044), the Fundamental Research Funds for the Central Universities (Grant No. 2015RC23).

References

[1] Chinner D, Higdon J. Exploring high bandwidth filesystems on large systems[C]//Proc. of 2006 Linux Symposium. Ottawa, Canada: [sn]. 2006: 177-191.

[2] Prabhakaran V, Arpaci-Dusseau A C, Arpaci-Dusseau R H. Analysis and Evolution of Journaling File Systems[C]//USENIX Annual Technical Conference, General Track. 2005: 105-120.

[3] Lensing P, Meister D, and Brinkmann A. hashfs: Applying hashing to optimize file systems for small file reads [C]//Storage Network Architecture and Parallel I/Os (SNAPI), 2010 International Workshop on. IEEE, 2010: 33-42.

[4] Bairavasundaram L N, Sivathanu M, Arpaci-Dusseau A C, et al. X-ray: A non-invasive exclusive caching mechanism for raids [C]//Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on. IEEE, 2004: 176-187.

[5] Beaver D, Kumar S, Li H C, et al. Finding a Needle in Haystack: Facebook's Photo Storage [C]//OSDI. 2010, 10: 1-8.

[6] Information on:http://tfs.taobao.org/

[7] Information on: https://www.oschina.net/question/tag/fastdfs

[8] Liu X, Han J, Zhong Y, et al. Implementing WebGIS on Hadoop: A case study of improving small file I/O performance on HDFS [C]//2009 IEEE International Conference on Cluster Computing and Workshops. IEEE, 2009: 1-8.

[9] Dong B, Qiu J, Zheng Q, et al. A novel approach to improving the efficiency of storing and accessing small files on hadoop: a case study by PowerPoint files [C]//Services Computing (SCC), 2010 IEEE International Conference on. IEEE, 2010: 65-72.

[10] Chandrasekar S, Dakshinamurthy R, Seshakumar P G, et al. A novel indexing scheme for efficient handling of small files in hadoop distributed file system [C]//Computer Communication and Informatics (ICCCI), 2013 International Conference on. IEEE, 2013: 1-8.

[11] Gohil P, Panchal B. Efficient ways to improve the performance of HDFS for small files [J]. Computer Engineering and Intelligent Systems, 2014, 5(1): 45-49.

[12] Kreps J, Narkhede N, Rao J. Kafka: A distributed messaging system for log processing [C] // Proceedings of the NetDB. 2011: 1-7.

[13] Lyden S, Haque M E. A simulated annealing global maximum power point tracking approach for PV modules under partial shading conditions [J]. IEEE Transactions on Power Electronics, 2016, 31(6): 4171-4181.