# File Creation Optimization for Metadata-Intensive Application in File Systems

Limin Xiao[1,2], Qiaoling Zhong[1,2](✉), Zhisheng Huo[1,2], Ang Li[1,2], Li Ruan[1,2], Kelong Liu[3], Yuanyuan Zang[3], and Zheqi Lu[3]

[1] State Key Laboratory of Software Development Environment, Beihang University, Beijing, China
qiaoling.0605@163.com

[2] School of Computer Science and Engineering, Beihang University, Beijing 100191, China
{xiaolm,ruanli}@buaa.edu.cn

[3] Space Star Technology Co., Ltd, Beijing 100086, China

**Abstract.** There are many steps among file creation, including creating metadata files in metadata servers, creating data files in data servers, creating a directory entry and adding it in the parent directory. The above steps are generic methods in distributed file system; however, it cannot achieve good performance in the metadata-intensive application where many clients create files at the same time, such as checkpointing, gene biological computing, high energy physics experiments. In this article, we present a method for file creation, called multi-stage file submission for metadata, which is used to optimize file creation in the metadata-intensive situation. This method is designed to make full use of the metadata servers' locality and decrease I/O operations. What we do is to make some changes among file creation for metadata and metafile storage. The procedure of file creation is based on Parallel Virtual File System version 2.8.2 (PVFS2) and we test the method in a simulation. The result shows that the throughout reaches to 14.06 kops, contrast to the original 0.92 kops, in the situation of sixteen clients and eight metadata servers. Of course, this method is used in metadata-intensive creation application.

**Keywords:** File system · File creation · Multi-stage file submission for metadata · I/O locality · Throughout

## 1 Introduction

Nowadays demand for high I/O throughout for large scale storage system is continuing to be imperative [15]. Many previous researches focus on improving the scale and performance on the data operations that read and write large amounts of file data by striping data across many servers or disks [6,10]. Many distributed file systems, such as PVFS2 [12], Lustre [10], Ceph [18], Hadoop Distributed File

System (HDFS) [16], separate metadata from data and storage them on different servers, which are metadata servers and data servers. The operations of metadata are much more than data operation, such as file lookup, file creation and file search. There are many involved metadata operations among a file operation. Researches [7,8,14] show that metadata access and modification operation make up to about seventy percent of file system I/O operations. In large scale file system, we cannot get the expected performance through deploying more servers and adapting more sophisticated hardware for the metadata wall [1]. It's the key to make metadata effective accessed in the large file system.

The metadata access characteristic of scientific computing and business computing application is intensive. Checkpointing is indispensable fault tolerance tool, commonly used by high-throughput applications. Checkpoint is massively parallel application for thousands of computing nodes [3]. File creation is one kind of intensive-metadata access operation [13]. We must optimize file creation to meet the metadata-intensive situation.

In this article, we present our method to optimize file creation for metadata-intensive application, called multi-stage file submission for metadata. The rest of paper is the following. In Sect. 2, we analyze related work; In section Sect. 3, we present basic file creation procedure in distributed metadata file system. In Sect. 4, we would present our method to scale file creation performance, called multi-stage file creation, which would mainly contain file creation protocol and metadata storage method. In Sect. 5, we give our experiment result to give evidence of the performance. At last, we give our conclusion.

## 2 Releted Work

In distributed file system, file creation consists of interaction protocol and metadata storage. The file creation interaction protocol charges with the message passing or data flow between client and server in the procedure of file creation.

There are many steps to accomplish file creation, including creating data file, creating metadata file, and adding a directory entry in the parent directory. Devulapalli et al. [5] designed alternative method based on distributed metadata file creation protocol in PVFS2, which contains compound operation, handling selection strategies and leased handles, leading to decrease the interaction between metadata servers and data servers and hidden the delay of parallel operation. Carns et al. [4] also designed a method to avoid several clients to send many independent file creation requests to decrease overhead, which is based on collective communication protocol among servers, simplifying the file creation consistent problem and improving file creation performance. Yi et al. [20] proposed a new protocol, Cx, in which the affected servers Concurrently eXecute the sub-operations of a cross-server file operation, and respond immediately to a client. From the above descriptions, the current researches focus on eliminating metadata accessing bottleneck of single file creation. When multiple clients create files, they will contend system resources. This leads to locality miss and degrades performance.

When we talk about metadata storage, the current method is to optimize the data storage of metadata server to improve metadata write performance. Stender et al. [17] presented BabuDB, a database, which stores file system metadata relying on LSM-tree-like index structures, which offers better scalability and performance than equivalent ext4 and Berkeley DB-based metadata server implementations. Ali et al. [2] presented two metadata management schemes, both of which remove the need for a separate metadata server and replace it with object-based storage. All of above storage methods can be concluded to separate metadata and data for storage, leading the metadata access to be small I/O and discrete and influencing the metadata write performance.

In massive parallel file creation, the current file creation protocol cannot enough meet the locality in metadata-intensive situation, degrading write metadata performance, because of metadata access interleaving. The storage method of metadata separates different file metadata among different location. This makes the disk I/O scheduler choose the best storage location in massive file creation situation and issues many unordered metadata access requests.

## 3 Basic File Creation Protocol

Now we firstly introduce the architecture of distributed metadata file system, and then we give the detail of file creation of distributed file system. The following Fig. 1 shows the architecture of distributed metadata file system.

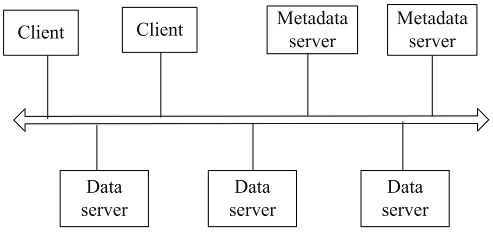

**Fig. 1.** Architecture of distributed file system

**Table 1.** Components of a file in distributed file system

| Entity | Description |
|---|---|
| Multiple data files | A large file consists of multiple data file |
| Metadata file | Owner, group, timestamp etc. |
| An entry in parent directory | Directory entry in parent directory |
| Additional attributes | A large file consists of multiple data file |

In distributed metadata file system, there are many components in a file, which is identified by a 64-bits long integer, called handle. Generally, a single file strips across several data servers and owns multiple entities on data servers and metadata servers. Table 1 shows components of a file in distributed metadata file system [12].

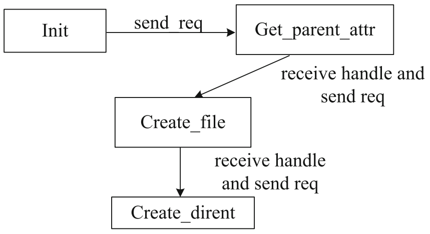

**Fig. 2.** File creation state machine on clients

In order to create a file, a client must get parent attribute to know the location of parent directory. A client can get the parent directory handle by lookup method, and then send a request to metadata servers to create a file. After creating a metafile, client sends request to metadata server and adds a directory entry in the parent directory. In the Fig. 2, we show the interaction between client and metadata servers.

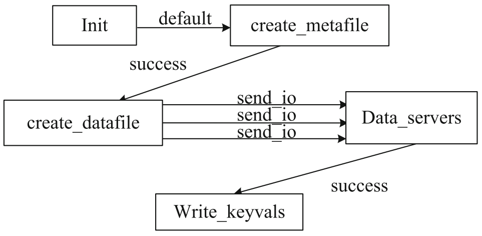

**Fig. 3.** File creation state machine on metadata servers

From the metadata server, if receiving a request from client, it starts to create a metafile and allocates a handle for this metafile. Then metadata server sends I/O requests to data servers to create data files for the metafile. At last, metadata server writes down the information of data files in the metafile (Fig. 3).

## 4 Design Alternatives

This section talks about our design method. There are several types of metadata access operation among file creation, including getting information of parent directory, file metadata creation, directory entry creation and adding it in parent directory. In massively parallel file creation situation, current file creation cannot take full advantage of the locality in the metadata servers, impacting the performance of metadata access. At the same time, there are many duplicated metadata access and discrete metadata operation. In order to obtain high performance, we must decrease duplicated metadata access and translate discrete metadata operation into serial metadata operation. Figure 4 shows the architecture of our method, called multi-stage file submission for metadata.

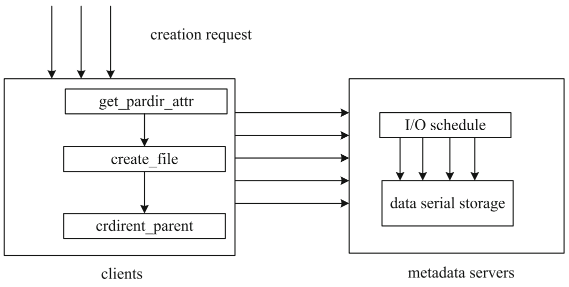

**Fig. 4.** Multi-stage file submission for metadata method

### 4.1 Client

In order to take advantage of the locality, we want to accommodate sub operations of file creation. The followings are the steps:

1. If a client sends a file creation request to server, a client must get enough information about the process to distinct different sub operation of file creation from different process.
2. In the client, we apply a monitor on file creation process, and then adjust file creation to adapt multi-stage file submission for metadata, separating file creation into multiple file creation sub operation. We set a timer on the sub operation queue, such as get_parentdir queue, create_file queue. Therefore, if a request is send from a client to servers, it would not be transmitted immediately, only after a time interval.
3. In the last, the processes in the clients add sub operation of file creation into the relative queue and wait the request to be completed.

Figure 5 shows how clients handle file creation. When a file creation request from a process send to metadata servers, it would be separated into several sub operations, which are added into relative queue, such as create_file queue, get_parentdir queue. In order to make use of the locality, we add a timer on the queue. After a time interval, a job request would be send to the metadata servers to get service. In this way, we expect to make use of the locality of metadata servers in massively parallel file creation application.

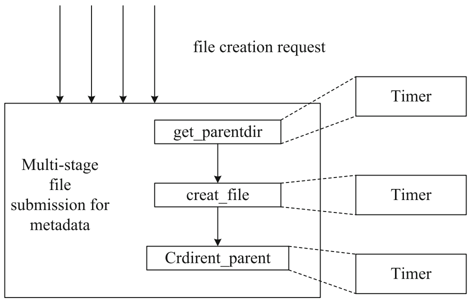

**Fig. 5.** Multi-stage files submission for metadata on clients

### 4.2 Metadata Servers

In the situation of metadata-intensive file creation, we adapt metadata I/O schedule algorithm and metadata storage to improve metadata write performance. The metadata I/O schedule algorithm is used to schedule the metadata I/O and aggregate the same directory file metadata creation requests into one request, and then only this one request would be send to underlying file system to complete file creation. The followings are the steps:

1. Based on the metadata request information, such as metadata file creation, metafile modification, etc., we build several queues about different metafile operation.
2. After receiving a metadata request from client, metadata server daemon process add it to the relative queue, such as creation queue. In our method, we add metafile creation requests in the same directory into the same creation queue.
3. On the metadata servers, we set a timer interval, which is as an aggregation time. At the expiration, the metafile creation requests are aggregated into one metafile creation request and send the underlying I/O operation.

Figures 6 and 7 show the procedure how metadata servers deal with file creation. Based on the metadata request information, we put metadata file creation request into creation queue and aggregate several file creation requests in the same directory into one file creation request after an aggregation time interval. In this way, we decrease the number of disk I/O, because the disk I/O is a key factor for improving performance.

## 5 Experiments and Results

In this section, we would present our experiment environment, including hardware, software and workload trace. After that, we present our result to get our expectation.

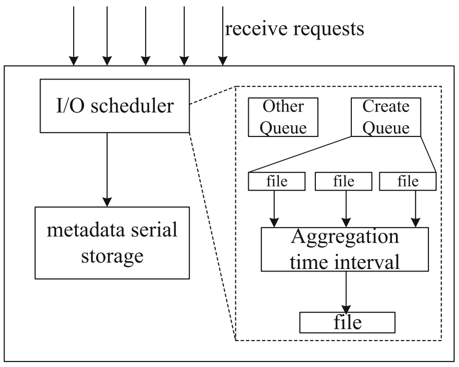

**Fig. 6.** Architecture of metadata servers on multi-stage files submission for metadata

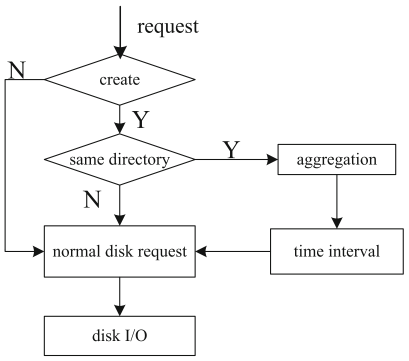

**Fig. 7.** Metadata servers file creation on multi-stages file submission for metadata

### 5.1 Experiment Environment

In our pass study, we has developed a simulation about distributed metadata file system, called DMFSsim [19], which is derived from PFSsim. PFSsim [9] is a simulation of PVFS2, which can effectively simulate the procedure of interaction between clients and servers, including network simulation, and runs in the OMNet++, an open source simulation tool [11]. Because PFSsim cannot support multiple metadata servers, we have developed another simulation DMFSsim, which is proved to be high efficiency and useful. In our experiment, we use DMFSsim to test our method and represent the results. In DMFSsim, all the nodes are connected with a high speed network with the average latency of 0.2 ms and the bandwidth of 1 Gbps. We evaluated our design by running simulations on a server with AMD Quad-core processor, 8 GB RAM, and 1 TB Seagate 7200 RPM hardware driver.

## 5.2 Workload and Results

Because we want to prove our method would be effective, we take two steps to verify our method.

Firstly, we show that only one client would run effectively after adapting this method in distributed metadata file system. In the experiment, we simulate one client and eight metadata servers. On the part of client, it runs a trace file to simulate an application, which runs in distributed metadata file system. In this simulation, we create a trace file, which stands for file creation workload in the same directory and is created by a script. In the trace file, we suppose that a client would create one thousand files continuously. At the same time, we set several request intervals to make simulation more real.

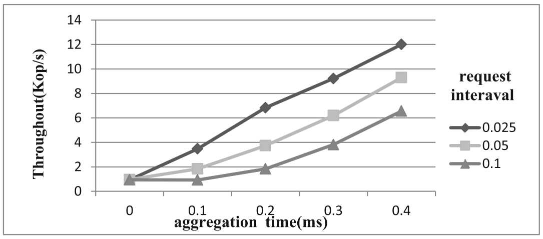

**Fig. 8.** Throughout for a client file creation

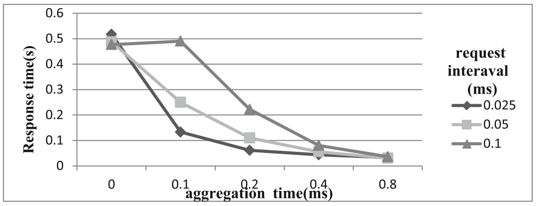

**Fig. 9.** Mean response time for a client file creation

The Figs. 8 and 9 show the file creation result for one client and eight metadata servers. From the above, if we handle file creation by the original method in distributed metadata file system, the throughout is about 0.95 kops for different request intervals. If a client send file creation by multi-stage file submission for metadata and metadata servers aggregate several metadata files into one metadata file, the throughout increases. Increasing of aggregation time, the throughout increases. For example, if request interval is 0.1 ms and aggregation time is 0.4 ms, the throughout can get up to 6.57 kops, which about times six throughout of the original method. Even there are different request intervals, throughout increases. The result shows that if there is less request interval, throughout can get more improvement. At the same time, response time is important for client.

From the experiment result, it shows that the original response time is about 0.5 s. If request interval is 0.1 ms, the response time increases at first and then decreases with the aggregation time increasing. The response time can get up to 0.03 s when the aggregation time is 0.8 ms. Although there may be more time to pay on a file creation request, the mean response time can decrease for that metadata servers can take full advantage of the locality and decrease the number of I/O.

Secondly, in order to prove that we can scale to multiple clients, we simulate sixteen clients and eight metadata servers. In this situation, we also make every client create one thousand file in the same directory.

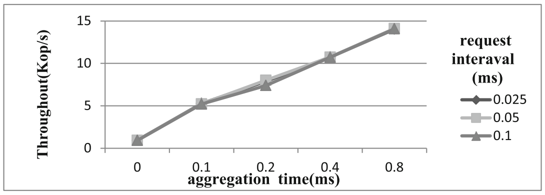

**Fig. 10.** Throughout for several clients files creation

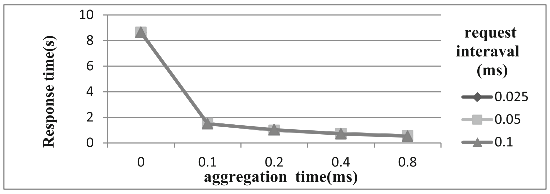

**Fig. 11.** Mean response time for several clients files creation

The Figs. 10 and 11 show the experiment result for sixteen clients and eight metadata servers. As expected, the throughout is about 0.92 kops in the old way. There will be an improvement for file creation by using multi-stage files submission for metadata method. No matter how long is the request interval, the throughout increases and even gets up to 14.06 kops with 0.8 ms aggregation time. At the same time, response time is also important for service. From the Fig. 11, the mean response time decreases. The mean response time is about 8.64 s when using the original method. By this method, the mean response time decreases to 0.52 s. Of course, we just test file creation operation for distributed metadata file system. The result shows that this method can scale to more clients, not only one client for distributed metadata servers.

## 6 Conclusion

In this paper, we firstly analyze the file creation protocol for distributed metadata file system, and then present our method, called multi-stage file submission for metadata. Of course, our method is just for intensive-metadata creation situation, which is more high performance. Because distributed metadata file system is more complex, we hope that this way could help distributed metadata file system design more useful for specific situation, especially for intensive-metadata creation.

**Acknowledgments.** The works described in this paper are supported by the fund of the State Key Laboratory of Software Development Environment under Grant No. SKLSDE-2014ZX-05, the National Natural Science Foundation of China under Grant No. 61370059 and No. 61232009, the Fundamental Research Funds for the Central Universities under Grant No.YWF-14-JSJXY-14, Beijing Natural Science Foundation under Grant No. 4122042, the Open Research Fund of The Academy of Satellite Application under grant NO. 2014-CXJJ-DSJ-04.

## References

1. 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)
2. Ali, N., Devulapalli, A., Dalessandro, D., Wyckoff, P., Sadayappan, P.: Revisiting the metadata architecture of parallel file systems. In: 3rd Petascale Data Storage Workshop, 2008. PDSW 2008, pp. 1–9. IEEE (2008)
3. Bent, J., Gibson, G., Grider, G., McClelland, B., Nowoczynski, P., Nunez, J., Polte, M., Wingate, M.: PLFS: a checkpoint filesystem for parallel applications. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, p. 21. ACM (2009)
4. Carns, P.H., Settlemyer, B.W., Ligon III, W.B.: Using server-to-server communication in parallel file systems to simplify consistency and improve performance. In: Proceedings of the 2008 ACM/IEEE conference on Supercomputing, p. 6. IEEE Press (2008)
5. Devulapalli, A., Ohio, P.: File creation strategies in a distributed metadata file system. In: IEEE International Parallel and Distributed Processing Symposium, 2007, IPDPS 2007, pp. 1–10. IEEE (2007)
6. Ghemawat, S., Gobioff, H., Leung, S.T.: The google file system. ACM SIGOPS Oper. Syst. Rev. **37**, 29–43 (2003)
7. Gu, P., Wang, J., Zhu, Y., Jiang, H., Shang, P.: A novel weighted-graph-based grouping algorithm for metadata prefetching. IEEE Trans. Comput. **59**(1), 1–15 (2010)
8. Leung, A.W., Pasupathy, S., Goodson, G.R., Miller, E.L.: Measurement and analysis of large-scale network file system workloads. USENIX Ann. Tech. Conf. **1**(2), 5.2 (2008)
9. Liu, Y., Figueiredo, R., Clavijo, D., Xu, Y., Zhao, M.: Towards simulation of parallel file system scheduling algorithms with PFSSIM. In: Proceedings of the 7th IEEE International Workshop on Storage Network Architectures and Parallel I/O, May 2011
10. Lustre: Lustre. http://lustre.org/. Accessed 08 March 2015
11. OMNeT++: Omnet++ discrete event simulator - home. http://www.omnetpp.org/. Accessed 08 March 2015

12. ParallelVirtualFileSystemVersion2: Parallel virtual file system, version 2. http://www.pvfs.org/. Accessed 08 March 2015
13. Patil, S.V., Gibson, G.A., Lang, S., Polte, M.: Giga+: scalable directories for shared file systems. In: Proceedings of the 2nd International Workshop on Petascale Data Storage: Held in Conjunction with Supercomputing 2007, pp. 26–29. ACM (2007)
14. 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)
15. Ross, R., Felix, E., Loewe, B., Ward, L., Nunez, J., Bent, J., Salmon, E., Grider, G.: High end computing revitalization task force (hecrtf), inter agency working group (heciwg) file systems and i/o research guidance workshop 2006 (2006)
16. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The hadoop distributed file system. In: 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), pp. 1–10. IEEE (2010)
17. Stender, J., Kolbeck, B., Hogqvist, M., Hupfeld, F.: BabuDB: fast and efficient file system metadata storage. In: 2010 International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI), pp. 51–58. IEEE (2010)
18. 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)
19. Wu, Q.M., Xie, K., Zhu, M.F., Xiao, L.M., Ruan, L.: DMFSsim: a distributed metadata file system simulator. Trans. Tech. Publ. Appl. Mech. Mater. **241**, 1556–1561 (2013)
20. Yi, L., Shu, J., Ou, J., Zhao, Y.: Cx: concurrent execution for the cross-server operations in a distributed file system. In: 2012 IEEE International Conference on Cluster Computing (CLUSTER), pp. 99–107. IEEE (2012)