I/O Optimizations Based on Workload Characteristics for Parallel File Systems

Bing Wei, Limin Xiao, Bingyu Zhou, Guangjun Qin, Baicheng Yan, Zhisheng Huo

I/O Optimizations Based on Workload Characteristics for Parallel File Systems

Bing Wei[1,2], Limin Xiao[1,2]✉, Bingyu Zhou[1,2], Guangjun Qin[1,2]✉, Baicheng Yan[1,2], and Zhisheng Huo[1,2]

[1] State Key Laboratory of Software Development Environment, Beihang University, Beijing, China
[2] School of Computer Science and Engineering, Beihang University, Beijing, China
{weibing,xiaolm,qingj}@buaa.edu.cn

Abstract. Parallel file systems usually provide a unified storage solution, which fails to meet specific application needs. In this paper, we propose an extended file handle scheme to address this problem. It allows the file systems to specify optimizations for individual file or directory based on workload characteristics. One case study shows that our proposed approach improves the aggregate throughput of large files and small files by up to 5% and 30%, respectively. To further improve the access performance of small files in parallel file systems, we also propose a new metadata-based small file optimization method. The experimental results show that the aggregate throughput of small files can be effectively improved through our method.

Keywords: Parallel file systems · Workload characteristics · Extended file handle · Small file optimizations.

1 Introduction

Applications with different workload characteristics usually have different access requirements for storage resources. The unified storage solution of parallel file systems fails to meet specific application needs. Many approaches [2–4] have been proposed to address this issue. However, these approaches cannot meet the following three requirements at the same time: (1) flexible management of I/O optimizations; (2) dynamical selection of I/O optimizations; (3) adaptive adjustment of I/O optimizations at runtime. In this paper, we propose an extended file handle (EFH) scheme to meet the above-mentioned requirements. The serving process of an I/O request can be customized with the EFH; hence, the corresponding optimization information can be achieved. To further improve the access performance of small files, we describe performance trade-off between small file load and metadata load based on the metadata-based method [5]. The steady trade-off model and the burst load trade-off model are established to determine the small file threshold. Small files are migrated across file system servers based on load condition, thereby improving the access performance of small files while avoiding overload on metadata servers.

The rest of this paper is organized as follows: Section 2 describes the design of extended file handle. Section 3 presents the small file optimization method. Section 4 presents the experimental results and discussions. Section 5 presents the conclusions.

2 Design of Extended File Handle

We describe the definition of the EFH model in this section. An example of extended file handle structure is shown in Fig. 1, an EFH consists of five elements, including logical file handle, real file

handle, version, optimization indices, and handle types. The logical file handle is used to uniquely identify a file. It is assigned by using a simple random distribution method when creating a file. The real file handle is the unique identifier of a file in the file system.

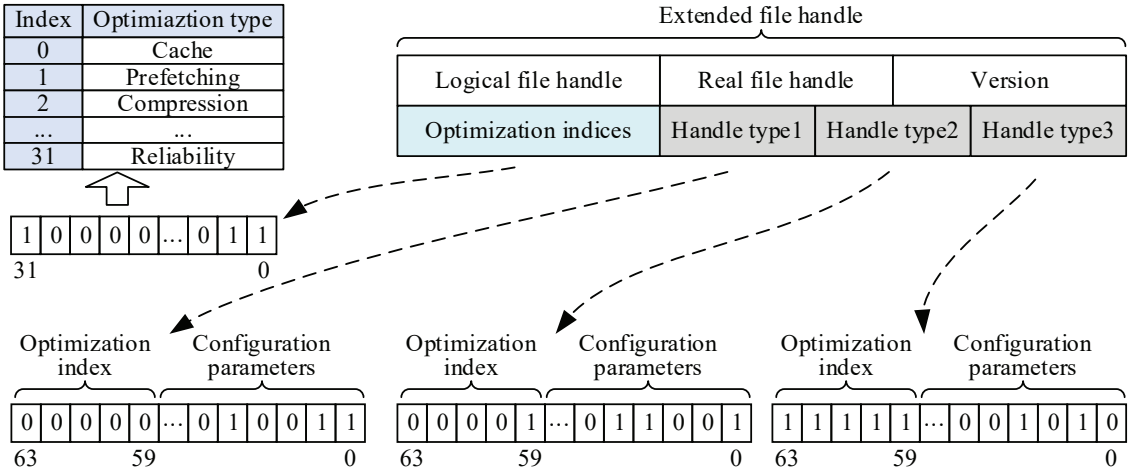

Fig. 1. An example of extended file handle structure.

The EFH version number is used for consistency maintenance. The 32-bit optimization index element indicates which optimization type is enabled. Each bit corresponds to an optimization type. If the bit is set to 1, then the corresponding optimization type is enabled; otherwise, it is not enabled. As a result, the I/O optimizations can be managed in fine grain. The handle types are used to record the customized configuration parameters for corresponding optimization type. The high 5-bit of a handle type records the index of optimization type that is corresponding to the handle type and the low 59-bit of a handle type records the corresponding configuration parameters. The EFH is stored in the directory entry that is stored on the metadata servers. Multi-type optimization information is managed with small memory overhead.

We abstract the processing of an I/O request across file system servers as a file I/O path. A proper file I/O path is selected based on the extended file handle. The process of selecting I/O path consists of four modules: EFH buffer, EFH parser, decision maker, and I/O path set. The recently used EFHs are cached in the EFH buffer. The main job of the EFH parser is to parse the EFH and transfer the parsed information to decision maker. The decision maker selects the proper I/O path based on the EFH parsed information to serve the I/O request. The I/O path set contains all the available I/O paths on the server. When changing the optimizations, enabling or updating a handle type may involve updating data between client-side and server-side. The version number of an EFH is incremented by 1 if updating successfully.

3 Small File Optimization Method

The steady trade-off model determines the small file threshold based on the long-term running status of system. The information of unused space capacity and the load of the metadata server

is periodically collected to calculate the global threshold (Gl_t), which is used to determine the threshold for a specific file and can be calculated by the following equation:

$$Gl_t = \begin{cases} Gl_{pre-t} - xGl_{pre-t} & Ca_{unused} \geq Ca_t \\ Max(Gl_{max}, Gl_{pre-t} + yGl_{pre-t}) & Ba_{io} < Ba_{low-t}, Ca_{unused} < Ca_t \\ Gl_{pre-t} - zGl_{pre-t} & Ba_{io} \geq Ba_{low-t}, Ca_{unused} < Ca_t \end{cases} \quad (1)$$

Ca_{unused} is the ratio of unused space capacity to the total space capacity. Ca_t is the threshold of unused space capacity. Gl_{pre-t} is the global threshold of the previous moment. Parameters x, y, and z are empirical adjustment parameters. Ba_{io} is the ratio of the current I/O bandwidth to the maximum I/O bandwidth. Ba_{high-t} and Ba_{low-t} are the high and low load threshold, respectively. Gl_{max} is the given maximum global threshold.

The migration frequency of a file is used to avoid frequent migrations of small files. The target threshold ($F_{target-t}$) for a file is the larger one between Gl_{max} and the fine-adjusted threshold. It can be calculated by the following equation:

$$F_{target-t} = Max(Gl_{max}, \frac{(\theta + Fre_m)Gl_t}{\theta}) \quad (2)$$

Fre_m is the migration frequency and θ is the empirical adjustment parameter. Once receiving the access request of a small file that is stored on a metadata sever, the target threshold for the file is calculated by equation 1 and equation 2. If the file size exceeds the target threshold, the file will be migrated to other servers. Reversely, if a file stored on a data server is truncated to a size below the target threshold, the file will be migrated to a metadata server.

The burst trade-off model determines the small file threshold in the burst load situation. The exponential smoothing method (ESM) calculates prediction value by the following equation:

$$E(t) = \lambda V(t-1) + (1-\lambda)E(t-1) \quad (3)$$

E(t) and *E(t-1)* are the prediction values for the moment t and *t-1*, respectively. λ is the smoothing parameter. *V(t-1)* is the observed value for the moment *t-1*. The prediction load can be easily calculated by equation 3. However, the prediction accuracy is low because of lacking of the consideration of the current I/O request status. A burst load sensing model (BLS-ESM) based on ESM is proposed to improve the prediction accuracy.

The I/O scheduler in the metadata server is used to determine the execution order of the I/O requests that are sent from the clients, and the requests that cannot be served at the current moment are blocked in the queue. $S_{t-2,t-1}$ is the amount of requested data that is served in the queue between moment *t-2* and *t-1*. $S'_{t-2,t-1}$ is the total amount of data that is blocked in the queue between moment *t-2* and *t-1*. The probability of burst load at the moment t can be calculated by the following equation:

$$R_{i-1} = \frac{S'_{t-2,t-1}}{S_{t-2,t-1}} \quad (4)$$

The larger the R_{i-1}, the greater the possibility of a burst load, and vice versa. Therefore, the predicted value at the moment t can be calculated by the following equation:

$$Gl_t = \begin{cases} (R_{i-1} - 1 + \lambda)V(t-1) + (1-\lambda)E(t-1) & R_{i-1} \notin (\mu, \nu) \\ \lambda V(t-1) + (1-\lambda)E(t-1) & R_{i-1} \in (\mu, \nu) \end{cases} \quad (5)$$

In the above equation, μ represents the low threshold of the burst load and ν represents the high threshold of the burst load. BLS-ESM is used to calculate the small file load prediction value at next moment for the metadata server.

4 Evaluation

Our experiments were conducted on a 5-node cluster of machines. Each machine was configured with two 20-core 2.2 GHz Intel Xeon 4114 CPUs, 128 GB of memory, two 7.2 K RPM 4 TB disks, and the Centos7 operating system. Each machine was configured with 5 virtual machines, which had the same configuration. The network was 1-Gigabit Ethernet. Our proposed approaches were conducted in PVFS [1].

4.1 Case Study: Directory Hint Optimization

We used traces *pweb* [6] and *pgrep* [6] to test data I/O performance for the three approaches, including default PVFS, PVFS-EFH (EFH), and directory hint (DH) [7]. Fig. 2 shows the aggregate throughput of the three above-mentioned approaches when replaying the two traces. EFH improves the aggregate throughput over PVFS in terms of small files for the two trace by up to 11% and 30%, respectively. Meanwhile, EFH improves the aggregate throughput over PVFS in terms of large files by up to 5% for *pweb* and has no significant impact on large files for *pgrep*.

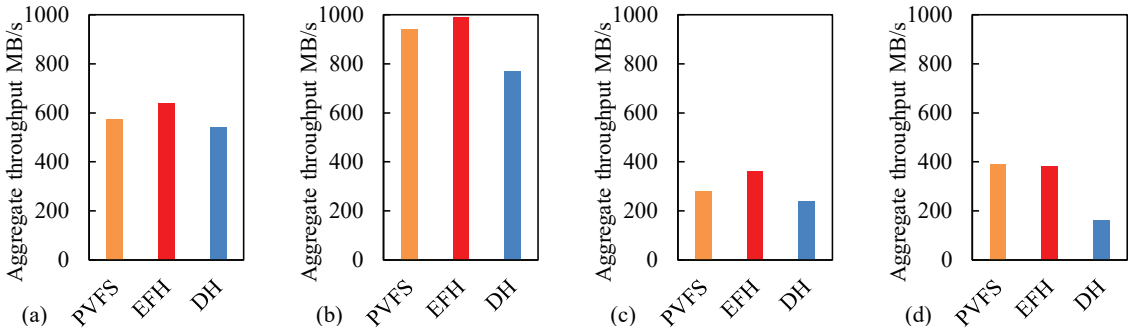

Fig. 2. The aggregate throughput of data I/O: (a) small files of pweb; (b) large files of pweb; (c) small files of pgrep; (d) large files of pgrep.

4.2 Testing Small File Optimization Methods

We used IOR [8] benchmark to test the performance of small file optimization methods. Fig. 3 shows the aggregate throughput of the original metadata-based method (OMB) [5] and our method under single metadata server. When increasing the number of client processes from 2 to 20, the metadata performance degradations for OMB and our method are 62% and 11%, respectively; the small file performance improvement for OMB and our method are 150% and 196%, respectively.

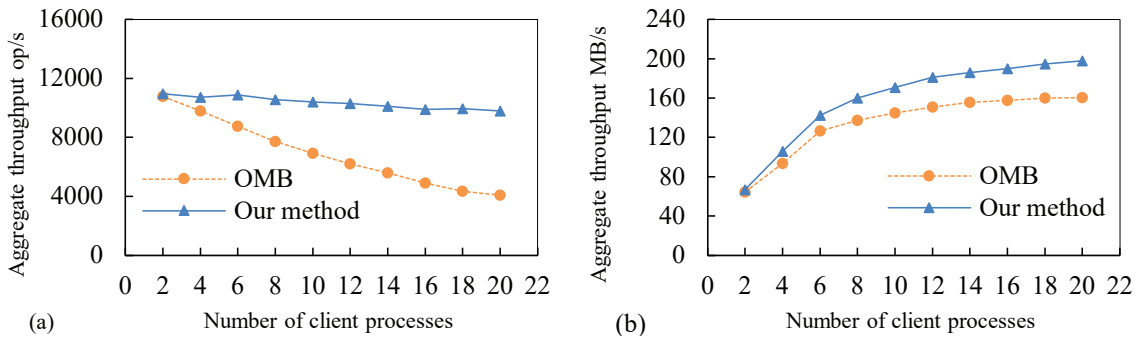

Fig. 3. The aggregate throughput: (a) metadata; (b) small files.

5 Conclusion

To meet the various requirements of multiple applications on storage resources, we propose an extended file handle scheme, which allows parallel file systems to specify customized optimizations for each file or directory based on workload characteristics. Our approach enables fine-grained management of selecting I/O optimizations for serving multiple workloads. We propose an adaptive optimization method to further improve small file performance. Performance trade-off between small file load and metadata load is achieved by our proposed method.

Acknowledgment

This work was supported by the National key R&D Program of China under Grant NO. 2017YFB1010000, the National Natural Science Foundation of China under Grant No. 61772053, the Science Challenge Project, No. TZ2016002, and the fund of the State Key Laboratory of Software Development Environment under Grant No. SKLSDE-2017ZX-10.

References

1. Ross, R, B., Thakur, R.: PVFS: A parallel file system for Linux clusters. Proceedings of the 4th annual Linux showcase and conference, pp. 391-430. (2000).
2. Isaila, F.: Collective I/O tuning using analytical and machine learning models. 2015 IEEE In-ternational Conference on Cluster Computing. pp. 128-137. IEEE, (2015).
3. Zhang, S., Catanese, H.: The composite-file file system: decoupling the one-to-one mapping of files and metadata for better performance. 14th USENIX Conference on File and Storage Technologies. pp. 15-22. (2016).
4. Byna, S., Chen, Y.: Parallel I/O prefetching using MPI file caching and I/O signatures. Pro-ceedings of the 2008 ACM/IEEE conference on Supercomputing. pp. 44. IEEE, (2008).
5. Carns, P., Lang, S.: Small-le access in parallel le systems. IEEE IPDPS 2009. pp. 1-11. (2009).
6. Uysal, M., Acharya, A.: Requirements of I/O systems for parallel machines: An application-driven study. (1998).
7. Kuhn, M., Kunkel, J, M.: Dynamic le system semantics to enable metadata optimizations in PVFS. Concurr. Comput. : Pract. Exper 21(14), 1775-1788 (2009).
8. LNCS Homepage, https://sourceforge.net/projects/ior-sio. Last accessed 16 May 2019