Dynamic Stripe Management Mechanism in Distributed File Systems

Jianwei Liao[1,2], Guoqiang Xiao[1], Xiaoyan Liu[1], and Lingyu Zhu[1]

[1] College of Computer and Information Science, Southwest University of China, Beibei, Chongqing, P.R. China, 400715
[2] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, Jiangsu, P.R. China, 210023

Abstract. This paper presents a novel mechanism to dynamically re-size and re-distribute stripes on the storage servers in distributed file systems. To put this mechanism to work, the information about logical I/O access on the client side is piggybacked to physical I/O access on the storage server side, for building the relationship between the logical I/O access and physical I/O access. Moreover, this newly presented mechanism supports varying size of stripes on the storage servers to obtain finer concurrency granularity on accessing to data stripes. As a result, the mapping relationship can be utilized to direct stripe re-sizing and re-distributing on the storage servers dynamically for better system performance. Experimental results show that this stripe management mechanism can reduce I/O response time and boost I/O data throughput significantly for applications with complicated access patterns.

Keywords: Distributed/parallel file systems, Re-sizing and re-distributing stripes, Varying stripe size, I/O optimization.

1 Introduction

The progresses in computation, storage and communication technologies firmly speedup the development of complicated data processing applications that need to deal with big data in distributed computing environments. According to the EMC-IDC Digital Universe 2020 study, the amount of data created, replicated, and consumed in China may grow 24-fold over 2012 and 2020 [1]. Thus, one particularly difficult challenge in this context is to find the right approach to store and manage such huge amounts of data in a distributed or parallel computing environment. The traditional centralized client/server model file systems have been proven to be a barrier to scalable performance in distributed computing systems [4]. Therefore, the file system deployed in a distributed computing environment is called a distributed file system, which is always employed to be a backend storage system to offer I/O services for various sorts of data-intensive applications. Actually, the distributed file system leverages multiple distributed I/O devices by striping file data across the I/O nodes, and uses high aggregate bandwidth to meet the growing I/O requirements of distributed scientific applications. In other words, a distributed file system is responsible for distributing

C.-H. Hsu et al. (Eds.): NPC 2014, LNCS 8707, pp. 497–509, 2014.

files on top of the involved storage devices, as well as managing the created files and their attributes [5] and [6].

In general, the method describing the mapping from logical files to a physical layout of bytes on storage servers is called file data distribution function or stripe distribution function. The generally adopted stripe distribution function is able to divide one-dimensional logical files into a set of non-overlapping chunks of data, which are called stripes. To be specific, files are supposed to be separated into many stripes, and then stored on the I/O nodes with certain distribution methods. Normally, the stripes are stored in a round robin manner on data files on the storage nodes [8], certain advanced distributed file systems, such as GPFS [9], Lustre [10] and Google file system [11] employ this kind of stripe distribution mechanism. It is well-known that data striping performance is also influenced by the application's I/O behavior, however, stripe distribution does not adjust, even though the distribution goes against application's access modes [12]. For instance, the requirements of continuous media file servers differ from the requirements of scientific applications that needs to process multi-dimensional data but the traditional distributed file systems treat them without any distinction [13]. From certain previous studies, it seems that the static distribution mechanism and the fixed stripe size configuration may perform poorly in dealing with a substantial quantity of multi-dimensional data, which may be read/written concurrently by a large number of clients [14].

In this paper, we propose a dynamic stripe management mechanism for distributed file systems, which enables varying stripe sizes, and supports stripe re-sizing and re-distributing on the storage servers. As a result, the distributed file systems can adjust stripe sizes and distribute stripes dynamically on the basis of both applications' access patterns and their corresponding disk access patterns, to yield better I/O performance. This mechanism makes the following two contributions:

1. *Piggybacking applications' access information to disk access patterns.* Applications' logical access information reveals the applications' behavior on the client side, but only the stripe access information on the storage servers shows the real disk operations. In this stripe management mechanism, the logical access information is supposed to be piggybacked with client I/O requests for benefiting to mapping logical access to stripe access, but client file systems do not need to keeping logs for logical access. The mapping relationship can definitely do good to conduct I/O optimization strategies on the storage server sides in the distributed file systems.
2. *Re-sizing and re-distributing stripes dynamically on the storage servers.* Except for supporting varying size of stripes, the newly proposed mechanism is able to perform dynamic stripe re-sizing and re-distributing on the storage servers, according to the mapping relationship between logical access information and physical access information. This indicates that it can boost I/O data throughput, as well as reduce I/O response time through conducting relevant I/O optimization strategies according to both logical and physical access patterns.

The following paper is organized as follows: Section 2 describes certain background knowledge and related work that aims to improve I/O performance in distributed file systems by employing different I/O optimization strategies. We will demonstrate design details of the mechanism of dynamic stripe management on the storage servers in Section 3. The evaluation methodology and relevant results are illustrated in Section 4. Finally, we conclude this paper in Section 5.

2 Related Work

For the purpose of yielding attractive I/O performance in the distributed file systems, much current work focuses on I/O optimization strategies for better I/O performance by resorting to keeping and analyzing either logical I/O traces or disk I/O traces. This section discusses some typical approaches, which are mainly sorted as the following two categories:

I/O Optimization by using either logical I/O access information or physical I/O access information. T. Madhyastha et al. [15] presented two approaches to reveal various file access patterns and then employ these access patterns to carry out the appropriate caching and prefetching optimization for file systems. The main idea for characterizing access patterns is to use neural networks for short time scales and Hidden Markov models for long time scales. The project IOSig+ allows users to classify the I/O access patterns of an application in two steps: 1) obtain the trace of all the I/O operations of the application from the view point of clients; 2) through the offline analysis on the trace to yield the I/O Signature. Therefore, by using the I/O Signatures, which is the information about logical I/O access patterns, certain optimization on I/O systems, such as data pre-fetching, I/O scheduling, and cost model based data access optimization can be conducted [14]. Besides, J. He et al. [23] have explored and classified patterns of I/O within applications, thereby allowing powerful I/O optimization strategies including pattern-aware prefetching to enhance I/O performance.

There are also many studies about the analysis of access patterns on disk I/O traces. Z. Li and Y. Zhou first investigated the block correlation in the storage servers by employing data mine techniques, to benefit to I/O optimization in servers [16] and [17]. S. Narayan and J. Chandy [18] researched disk I/O traffics under different workloads and different file systems, and they declared the modeling information about physical I/O operations can contribute to I/O optimization tactics for better system performance [19]. In [20], an automatic locality-improving storage has been presented, which automatically reorganizes selected disk blocks based on the dynamic reference stream to boost effective storage performance. After that, *DiskSeen* has been presented that supports to perform prefetching directly at the level of disk layout [21]. H. Song et al [22] have presented a server-side I/O collection mechanism to coordinate file servers for serving one application at a time to decrease the completion time.

Intelligently setting stripe size in file systems. H. Simitci [24] proposed an adaptive mechanism to set the size of striping unit according to the system's

state. To be specific, the size of stripes can be determined on the basis of some parameters including the request rate, the request size, the network flow, and disk speed. M. Medina et al. [13] proposed a self-tuning approach for automatically determining and refining the file system's striping parameters based on application access patterns. In other words, this technique relies on the monitoring of application I/O requests including their size, type, duration and inter-arrival times etc., and then a proper analytic model is used to decide file striping parameters to improve overall file system performance. Therefore, the self-tuning file systems usually operate correspondingly according to the principle that the behavior of the file system must change to match the application. B. Dong et al. [25] have proposed an analytic model to evaluate the performance of highly concurrent data access, and then they have described how to apply this model to determine the stripe size of a file. However, this adaptive disk striping approach does not allow change the size of file stripe dynamically and various sizes of stripes belonging to the file.

Besides, Triantafillou and Faloutsos [26] presented the mechanism of overlay striping, which is a novel data distribution scheme, it stores several copies of a file prior to its use, leveraging a number of different stripe widths. As a result, the relevant replica with the most beneficial stripe width will be accessed. N. Ali et al. have presented a fault-tolerant mechanism to distribute the parity computation for generalized Cartesian data distributions on the storage servers. Actually, in [12], we have proposed a self-tuning storage system that supports stripe movement among storage servers on the fly. But it requires the client file system to record the logical I/O events, and the stripe size is fixed all the time.

It is true that logical access patterns on the application side may affect the I/O performance on the storage server side, that is the reason certain file systems enable self-tuning functionality for determining stripe size and stripe location, according to logical access patterns. On the other hand, only the physical access patterns can disclose the disk behaviors corresponding to logical access. However, the fact is that none of the mentioned techniques and tools support the optimization strategy of supporting dynamic re-sizing and re-distributing stripes by analyzing both logical I/O access patterns and their corresponding physical access patterns in the distributed file systems. Therefore, our work addresses that it is able to build the connection between logical I/O access and physical I/O access; then help the storage servers to re-size and re-distribute the stripes, as well as enable varying size of stripes, for better I/O performance.

3 Dynamic Re-sizing and Re-distributing stripes

In Section 2, we depicted that a major part of I/O tracing approaches proposed by other researchers focus on the logical I/O access occurred on the client file system side, which might be useful for affirming application's I/O access patterns [14]. Nevertheless, without relevant information about physical I/O access, it is difficult to build the connection between the applications and the distributed file system for enhancing the I/O performance significantly through I/O

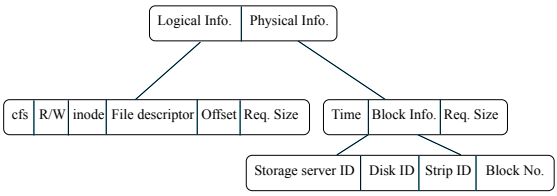

Fig. 1. Logged information about logical access and the corresponding physical access

optimization on the storage servers. Therefore, this section describes the details of the way to support varying size of stripes, and then enable dynamic re-sizing and re-distributing stripes on the storage servers to advance I/O system's performance.

3.1 Piggybacking Logical Access Information to Servers

In this newly presented stripe management mechanism, understanding the mapping relationship between logical access and physical access is a critical precondition to perform I/O optimization. Thus, for storage servers, it is necessary to know the information about client file systems and applications. Although we have proposed a mapping mechanism in our previous work [12], it requires the client file systems to keep the track of logical access information, and then send the tracing logs to the server side. To reduce the overhead resulted by client logging in our previous work, we leverage a piggybacking mechanism, to transfer related information from the client node to the storage servers for contributing to construct the mapping relationship between logical access and physical access. To put it from another angle, the client file system is responsible for keeping extra information about the application, client file system and the logical access information; after that, it piggybacks the extra information with relevant I/O request, and sends them to the corresponding storage server. On the other hand, the storage server is supposed to parse the request to separate piggybacked information and the real I/O request. Apart from forwarding the I/O request to the low level file system, the storage server has to record the disk I/O access with the information about the corresponding logical I/O access.

Briefly speaking, when sending a logical I/O request to the storage server, the client file system piggybacks information about the client file systems and the application. In this way, the storage servers can record disk I/O events with associated client information, which plays a critical role for modeling I/O access relationship, and then directing stripe optimization operations on the storage servers dynamically.

3.2 Mapping Access Patterns

As mentioned before, the client information is piggybacked to the storage servers, then the storage servers are possible to record the disk I/O operations accompanying with the information about relevant logical I/O events. Figure 1 demonstrates the structure of each piece of logged information, which is stored on the relevant storage server. The information about logical access includes *inode* information, *file descriptor*, *offset* and *requested size*. On the other hand, the information about the relevant physical access contains *storage server ID*, *stripe ID*, *block ID* and *requested size*.

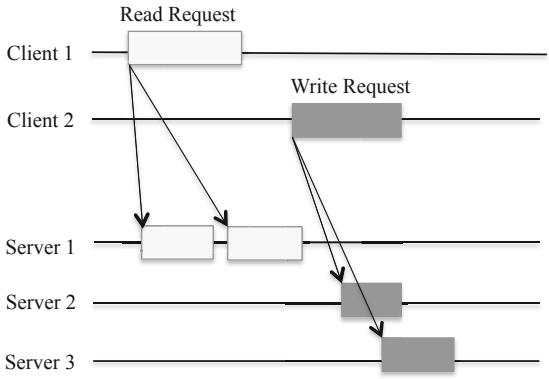

Fig. 2. Mapping Example of logical I/Os and physical I/Os

After analyzing the recorded logs for a series of I/O operations, we can easily to obtain the relations between logical access and physical access. Figure 2 illustrates an example case about I/O visualization of both kinds of I/O access information. In the figure, the *Read request* from *Client 1* is reflected to *Server 1*, so that the two relevant stripes on that server will be accessed sequentially, which may damage access concurrency. In addition, the *Write request* from *Client 2* is separately mapped to *Server 2* and *Server 3*. From the visualization illustration between logical access and physical access, it is not difficult to issue I/O optimization on the storage servers after understanding the shortcomings of the current stripes distribution. For example, the *Write request* from *Client 2* is mapped to *Server 2* and *Server 3*, which indicates that the two stripes should be updated, as well as the replicas corresponding to these two stripes. If this *Write request* is mapped to only one stripe (merging the involved two stripes), the number of replica synchronization can be reduced to a half. That is why we have done the work to support dynamically performing stripe optimization according to the access patterns.

3.3 Re-sizing and Re-distributing Functions

This paper presents a stripe management mechanism that allows varying size of stripes on storage servers on the basis of analysis of the mapping of access patterns. In other words, this newly introduced mechanism makes it possible that the stripes are possible to be re-sized and truncated dynamically for some reason. Figure 3 (a) and (b) illustrate two cases about re-distributing an existing stripe and creating a new stripe (i.e. truncating an existing stripe to generate a new stripe) respectively.

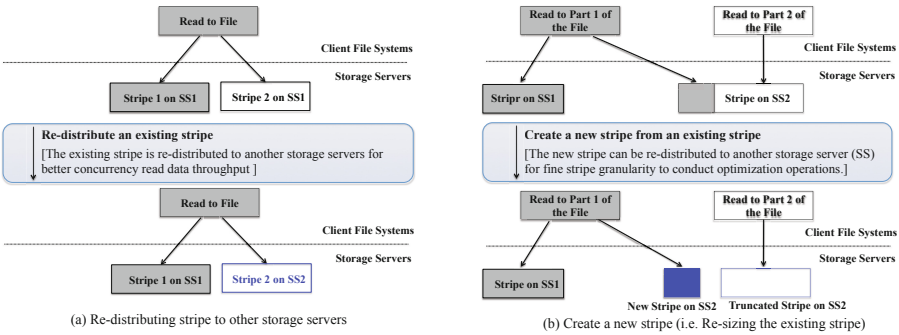

Fig. 3. Re-distributing and Re-sizing stripes on the storage servers

Let us take Figure 3(b) as an instance, the first read operation (i.e. *Read to Part 1 of the File*) is reflected to 2 stripes that stored in two different storage servers (i.e. *SS 1* and *SS 2*), the reason for this situation is due to application's specification and default round-robin stripe distribution function. In this case, the first read request does damage to the second read request (i.e. *Read to Part 2 of the File*) that might be issued by another client file system in our example. Because the first read needs to lock the whole stripe stored on *SS 2*, that means the second write or read request should wait until obtain the lock to that stripe, even though it does not read the contents that requested by the first read request. To overcome this problem, this paper introduces a novel stripe management approach, which supports varying size of stripe units, and enables dynamic re-sizing, re-distributing the existing stripes. Therefore, the conflicted stripe stored on the *SS2* can be divided into 2 stripes, and the first read request does not need to lock the stripe file, which is requested by the second read request. Without doubt, after re-sizing and re-distributing operations, the metadata server should be notified to update relevant metadata, e.g. stripe sizes and stripe locations.

4 Experiments and Evaluation

4.1 Experimental Setup

Experimental Platform. One cluster and two LANs are used for conducting the experiments, one active metadata server, 4 storage servers are deployed on the 5 nodes of the cluster. Moreover, for emulating a distributed computing environment, 6 client file systems are installed on a LAN that is connected with the cluster by a 1 GigE Ethernet; another 6 client file systems are installed on another LAN but with same node specifications, which is connected with the cluster by a 100M Ethernet, and both LANs equipped with MPICH2-1.4.1. Table 1 show the specifications of nodes on both of them.

Table 1. Specification of Nodes on the Cluster and the LANs

	Cluster	**LANs**
CPU	2xIntel(R) E5410 2.33G	Intel(R) E5800 3.20G
Memory	1x4GB 1066MHz/DDR3	4GB DDR3-SDRAM
Disk	6x114GB 7200rpm SATA	500GB 7200rpm SATA
Network	Intel 82598EB, 10GbE	1000Mb or 100 Mb
OS	Ubuntu 13.10	Debian 6.0.4

Evaluation Counterparts. To demonstrate the effectiveness of our proposed dynamic stripe management scheme, we have employed the conventional distributed file system and the self-tuning storage system, as comparison counterparts in our experiments:

- *Dynamic Re-sizing and Re-distributing Storage (D2RS).* The proposed mechanism has been implemented and applied in a prototype distributed file system, which enables varying stripe sizes and dynamic stripe re-sizing.
- *Conventional Self-Tuning Storage (CSTS).* We implemented a self-tuning storage system in our previous work [12], which supports certain preliminary optimized I/O strategies, such as stripe migration on storage servers, but without piggybacking mechanism and supporting for varying size stripes. As a matter of fact, this work is the most related scheme of our proposed *D2RS*.
- *Conventional Storage System (CSS).* The storage servers are responsible for all I/O operations normally, and the data stripes are distributed with the normal round-robin pattern. That indicates no I/O tracing and no dynamic stripe re-sizing and stripe re-distributing functionality.

Benchmarks. We selected two benchmarks to evaluate our proposed stripe management approach and its comparison counterparts.

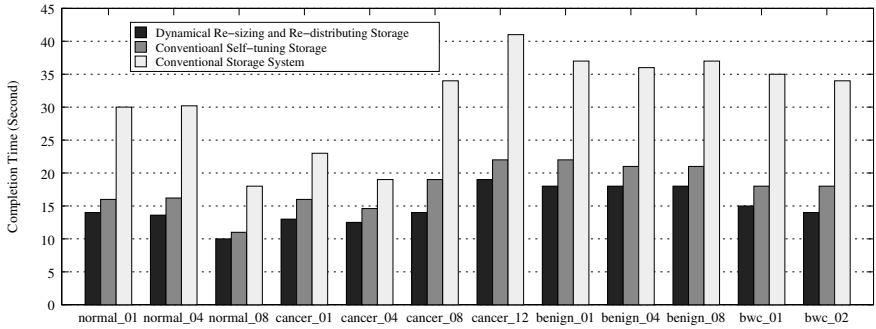

Fig. 4. Completion Times for Running *DDSM* Sampled Volumes

– *Digital Database for Screening Mammography (DDSM)*, which is a database of digitized film-screen mammograms with associated ground truth and other information. The purpose of this resource is to provide a large set of mammograms in a digital format that may be used by researchers to emulate medical image processing [2].
– *MADbench2*, which is an I/O benchmark derived from a real world application analyzing massive cosmic microwave background radiation in the sky from noisy pixelated datasets from satellites [3]. Since MADbench2 performs large, contiguous mixed read and write patterns, it has become a popular and often used benchmark in the parallel I/O community.

4.2 Experimental Results: Benefits and Overhead

In this section, we are expected to unveil the overhead brought by the scheme of dynamic re-sizing and re-distributing, as well as the benefits brought by this newly presented stripe management scheme. Thus, the following two sub-sections explore both positive and negative aspects of the proposed scheme respectively.

Improvement on I/O Performance We employed the aforementioned two application benchmarks to measure I/O responsiveness and data throughput respectively to show the merits brought by our proposed mechanism. First, we executed *DDSM* on the three storage systems, and recorded the time required for executing all sub-benchmarks in *DDSM*. The relevant results are reported in Figure 4, and it is not difficult to know that all sub-benchmarks of *DDSM* completed with the shortest times while it run on *D2RS*, because the computation times of the sub-benchmarks are the same, but *D2RS* caused the least times for I/O processing. For instance, when the sub-benchmark is *cancer_08*, *D2RS* yielded more than 30% execution accelerating, compared with *CSTS*, which means *D2RS* have better I/O responsiveness while processing multi-dimensional datasets.

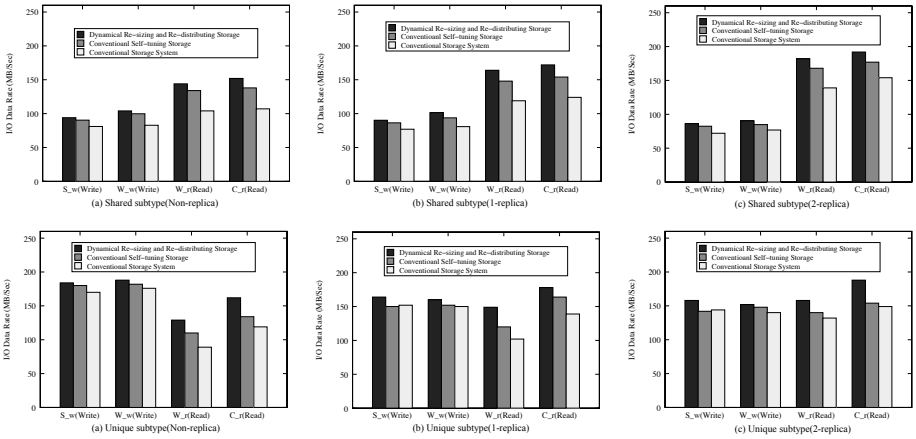

Fig. 5. MADbench2 Experimental Results (MPI, SYNC, 18KPIX, 16BIN)

Moreover, for the purpose of checking the improvement on data throughput by adopting our proposed mechanism, we run *MADbench2* benchmark, and set various number of replicas for each stripe to measure read/write data throughput. Actually, in *MADbench2*, the function S only writes, the function W both reads and writes, the function C only reads; so that, the sub-benchmarks are denoted as S_w, W_w, W_r, C_r to show the different I/O operations in the different functions. Figures 5 shows the experimental results of executing MADbench2 benchmark with *Shared* and *Unique* subtype when adopting different configuration of replicas. In all sub-figures, X axis shows the names of sub-benchmarks in *MADBench2*, while Y axis indicates I/O data rate, and the higher one is better. From the results shown in the figure, we can safely make a conclusion, compared with other two comparison counterparts, *D2RS* could potentially result in better overall data throughput. Especially, while the number of replicas is becoming larger, the improvement on data throughput is more attractive. This is because re-sizing stripes and creating new independent stripes may reduce the update synchronization overhead caused by write operations.

Overhead on Client and Storage Servers. After disclosing the positive effects brought by the newly proposed stripe management mechanism, it might be interesting to unveil the negative aspects on I/O performance caused by this mechanism. Table 2 shows the execution time and space overheads for performing stripe re-sizing and re-distributing dynamically on the storage servers, as well as keeping the track of physical I/O access. The results show that this newly proposed mechanism can effectively and practically guide re-sizing and re-distributing stripes on the storage servers for different workloads with acceptable overhead on CPU time and disk space. Because *MADbench2* is a typical I/O benchmark to test I/O performance of storage systems, more than 7.4%

time overhead on trace analyzing and re-sizing stripes on the storage servers. But, for the compute-intensive *DDSM* application, our proposed mechanism consumed not much time to yield preferable system performance. Namely, while the workload is *DDSM*, the time required for keeping disk traces and performing dynamic stripe management is around 3.7% of total processing time. This trend indicates that a major part of processing time can be used to tackle I/O processing; therefore, we can understand that this server-side, dynamic stripe management technique is practical for storage systems in distributed computing environments.

Table 2. Overhead on Dynamic Stripe Management Mechanism

Benchmarks	Consumed time (%)	Space for traces (MB)
DDSM (Overall)	3.7	588.4
Madbench (Shared, Non-replica)	7.4	332.7
Madbench (Unique, Non-replica)	8.6	443.8

Besides, we also recorded the disk space utilized to save the physical I/O traces, which are used to analyze access patterns for conducting potential optimization. The relevant results are reported in the table as well. It is clear that the space used for storing I/O traces is not so much, in contrast to the space used for storing the input and output data required by the benchmarks. For instance, *DDSM* benchmark deal with more than 60 GB data, but it uses less than 600 MB space for saving I/O trace data.

5 Concluding Remarks

This paper presents a novel stripe management technique in distributed file systems, in which the stripe size is varying from each other, and the data stripes can be re-sized and re-distributed dynamically according to the access patterns of target applications. The evaluation experiments have illustrated the effectiveness of this newly proposed mechanism, and the attractive experimental results demonstrated that our introduced stripe management mechanism is practical for storage systems in distributed computing environments. As a matter of fact, we have implemented this approach into a prototype distributed file system to verify the feasibility of the idea presented in this paper, it can be not only applied to other traditional distributed file systems, but also parallel file systems, such as the Lustre file system, the Google file system, the GPFS file system, or their extensions, as well.

Acknowledgment. This work was supported partially by "National Natural Science Foundation of China (No. 61303038"), "Natural Science Foundation Project of CQ CSTC (No. CSTC2013JCYJA40050)", and "the Opening Project of State Key Laboratory for Novel Software Technology (No. KFKT2014B17)".

References

1. Gantz, J., Reinsel, D.: The digital universe in 2020: Big Data, Bigger Digital Shadows, Biggest Growth in the Far East, United States (2013), http://www.emc.com/collateral/analyst-reports/idc-digital-universe-united-states.pdf (accessed on October 3, 2013)
2. Digital database for screening mammography, http://marathon.csee.usf.edu/Mammography/Database.html (accessed on December 12, 2011)
3. MADbench2. borrill/MADbench2/, http://crd.lbl.gov/
4. Weil, S.A., Pollack, K.T., Brandt, S.A., Miller, E.L.: Dynamic metadata management for petabyte-scale file systems. In: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, SC 2014, pp. 4–15. IEEE Computer Society, Washington, DC (2004)
5. Nieuwejaar, N., Kotz, D.: The galley parallel file system. Parallel Computing 23(4-5), 447–476 (1997)
6. Kunkel, J., Ludwig, T.: Performance evaluation of the pvfs2 architecture. In: Proceedings of 15th EUROMICRO International Conference on Parallel, Distributed and Network-Based Processing (PDP 200), pp. 509–516 (2007)
7. Liao, J., Ishikawa, Y.: Partial replication of metadata to achieve high metadata availability in parallel file systems. In: Proceedings of the 41st International Conference on Parallel Processing, ICPP 2012, pp. 168–177 (2012)
8. Latham, R., Miller, N., Ross, R., Carns, P.: A Next- Generation Parallel File System for Linux Clusters. Linux World 2(1) (2004)
9. Schmuck, F., Haskin, R.: Gpfs: A shared-disk file system for large computing clusters. In: Proceedings of the 1st USENIX Conference on File and Storage Technologies, FAST 2002. USENIX Association, Berkeley (2002)
10. Schwan, P.: Lustre: Building a file system for 1,000-node clusters. In: Proceedings of the Linux Symposium, p. 9 (2003)
11. Ghemawat, S., Gobioff, H., Leung, T.: The Google file system. ACM SIGOPS Operating Systems Review 37(5), 29–43 (2003)
12. Liao, J.: Self-tuning optimization on storage servers in parallel file system. Journal of Circuits, Systems and Computers 30(4), 21 pages (2014)
13. Medina, M.: A self-tuning disk striping system for parallel input/output. Dissertation. University of Illinois at Urbana-Champaign, USA (2007)
14. Byna, S., Chen, Y., Sun, X.-H., Thakur, R., Gropp, W.: Parallel i/o prefetching using mpi file caching and i/o signatures. In: SC 2008, pp. 44:1-44:12 (2008)
15. Madhyastha, T.: Automatic Classification of Input/Output Acess Patterns. Dissertation, Champaign, IL, USA (1997)
16. Li, Z., Chen, Z., Srinivasan, S., Zhou, Y.: C-Miner: Mining Block Correlations in Storage Systems. In: Proceedings of the 3rd Conference on File and Storage Technologies, FAST 2004 (2004)
17. Li, Z., Chen, Z., Zhou, Y.: Mining Block Correlations to Improve Storage Performance. ACM Transactions on Storage 1(1), 213–245 (2005)
18. Narayan, S., Chandy, J.: Trace Based Analysis of File System Effects on Disk I/O. In: Proceedings of 2004 International Symposium on Performance Evaluation of Computer and Telecommunication Systems, SPECTS 2004 (2004)
19. Narayan, S.: File System Optimization Using Block Reorganization Techniques. Master of Science Thesis, University of Connecticut (2004)

20. Hsu, W., Smith, A., Young, H.: The automatic improvement of locality in storage systems. ACM Trans. Comput. Syst. 23(4), 424–473 (2005)
21. Jiang, S., Ding, X., Xu, Y., Davis, K.: A Prefetching Scheme Exploiting both Data Layout and Access History on Disk. ACM Transaction on Storage 9(3), Article 10, 23 p. (2013)
22. Song, H., Yin, Y., Sun, X., Thakur, R., Lang, S.: Server-side I/O coordination for parallel file systems. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2011). ACM (2011)
23. He, J., Bent, J., Torres, A., Sun, X., et al.: I/O Acceleration with Pattern Detection. In: Proceedings of the 22nd International ACM Symposium on High Performance Parallel and Distributed Computing (HPDC 2013), pp. 26-35 (2013)
24. Simitci, H.: Adaptive Disk Striping for Parallel Input/Output. Dissertation, Champaign (2000)
25. Dong, B., Li, X., Xiao, L., et al.: A New File-Specific Stripe Size Selection Method for Highly Concurrent Data Access. In: Proceedings of 2012 ACM/IEEE 13th International Conference on Grid Computing (GRID), pp. 22-30 (2012)
26. Triantafillou, P., Faloutsos, C.: Overlay striping and optimal parallel I/O for modern applications. Parallel Computing 24(1), 21–43 (1998) Special Issue on Applications: Parallel Data Servers and Applications (1998)