Performance-Aware Data Placement in Hybrid Parallel File Systems

Shuibing He[1 2], Xian-He Sun[2], Bo Feng[2], Kun Feng[2]

[1] School of Computer, Wuhan University, Wuhan, Hubei, China
[2] Department of Computer Science, Illinois Institute of Technology, Chicago, IL, USA
{she11, sun, bfeng5, kfeng1}@iit.edu

Abstract. Hybrid parallel file systems (PFS), which consist of both HDD and SSD servers, provide a promising solution for data-intensive applications. In this study, we propose a performance-aware data placement (PADP) strategy to enable efficient data layout in hybrid PFSs. The basic idea of PADP is to dispatch data on different file servers with adaptive varied-size file stripes based on the server storage performance. By using an effective data access cost model and a linear programming optimization method, the appropriate stripe sizes for each file server are determined effectively. We have implemented PADP within OrangeFS, a widely used parallel file system in HPC domain. Experimental results of representative benchmark show that PADP can significantly improve the I/O performance of hybrid PFSs.

Keywords: Parallel I/O System; Parallel File system; Solid State Drive

1 Introduction

Data I/O access is a key performance bottleneck of modern computer systems. To tackle this problem, parallel file systems (PFS) have been proposed to speed up large-scale data accesses. In many PFSs (e.g., OrangeFS [1] and Lustre [2]), a file usually is distributed across multiple file servers with a fixed-size stripe. When serving a client request concurrently by multiple nodes, the I/O bandwidth is significantly aggregated and improved. However, while PFSs favor large requests, they fail to perform well when serving clusters of small requests, especially random requests. Therefore, how to optimize PFS performance with different I/O patterns is still a challenging task the high performance computing (HPC) community is facing.

At the same time, newly emerged storage technologies, such as flash-based solid state drives (SSD), provide a new opportunity for I/O system design. Compared to traditional HDDs, SSDs have higher storage density, lower power consumption, a smaller thermal footprint and orders of magnitude higher performance [3]. However, it is not practical to replace HDDs completely with SSDs in a large-scale HPC system for two reasons. First, building PFSs solely based on SSDs may be too expensive for most systems. Second, HDDs have several

advantages that satisfy the demands of HPC well, such as large capacity, attractive cost per storage unit, and decent peak bandwidth for large requests. Therefore, hybrid PFSs that consists of HDD-based file servers (HServer) and SSD-based file servers (SServer), provide a promising solution for data-intensive applications [4, 5]. This kind of situation is especially common in low-cost clusters, where cost is a critical issue and old components have to be used as much as possible.

Although hybrid PFSs have brought new opportunities to many application domains, there are many technical challenges yet to be solved before it is widely adopted. One of the major issues is, with the limited SServer resources, how to maximize the overall I/O system performance. Optimizing file system data placement (layout) is one of the most effective way to reach this goal. Researchers have made significant efforts to get an optimal data placement via adjusting the file stripe size [6, 7], or file stripe distribution method [8]. All these techniques introduce an optimal data layout based on application's data access patterns and guarantee the load-balance of all file servers. However, these approaches often focus on homogeneous PFSs, there is little consideration toward the performance difference among heterogeneous file servers in hybrid PFSs. For instance, for the file stripe size adjustment problem addressed in [6–9], the solutions are based on the assumption that all of the file servers are based on traditional HDDs.

However, in hybrid PFSs, the performance of each kind of file servers (i.e., HServer and SServer) varies significantly. A high-speed SServer can finish processing and storing data in a local SSD of the server faster than the relatively low-speed HServer. Traditional PFSs usually place a large file across multiple file servers with a fixed-size stripe. These data placement schemes are suitable for homogeneous environments, because they are able to provide concurrent I/O accesses and good load balance among multiple file servers. When applied to hybrid PFSs where file servers are not identical fixed-size stripe placement schemes may lead to severe load-imbalance among file servers and can significantly degrade the I/O performance. As high-speed SServers spend plenty of time on waiting the slower HServers during the I/O service, the potential of the hybrid PFSs is not fully utilized.

In this paper, we propose a performance-aware data placement scheme (PADP) to optimize the data placement in hybrid PFSs. Compared to traditional placement schemes, PADP distributes file data on different file servers with appropriate varied-size stripes according to their storage performance. For heterogeneous system it is not easy to determine the appropriate stripe sizes for different file servers. There are three main reasons. First, the performance of file server can be impacted significantly by I/O patterns. Second, even under the same I/O patterns, the performance between HServer and SServer can be different due to their distinct storage media characteristics. Third, besides the storage cost, the overall I/O performance is a function of the underlying network, which should be considered when evaluating the data access performance. The proposed performance-aware scheme takes the above three challenges into consideration in the data placement on hybrid PFSs, and can minimize the overall I/O access

time from a client point of view. In addition, it can be extended to systems with other kinds of heterogeneous file servers, system configurations, and I/O patterns.

Specifically, we make the following contributions.

- We develop an analytical model to evaluate the overall I/O completion time of each data access in hybrid PFSs.
- Based on the cost model, we use a linear programing method to determine the optimal stripe size for each file server.
- We propose a performance-aware data placement scheme with the optimal stripe sizes to improve the hybrid file system performance.
- We implemented a prototype of PADP under OrangeFS, and evaluated its performance with IOR benchmark. Extensive experimental results show that PADP can significantly improve the I/O throughput of hybrid parallel file systems.

The rest of this paper is organized as follows. Section 2 discusses the related work. The design and implementation of PADP is described in section 3. Section 4 presents the performance evaluation with commonly used benchmark. Finally, conclusions are summarized in section 5.

2 Related work

Optimizing data placement of parallel file system is an effective approach to improve I/O performance. Parallel file systems usually provide several data placement policies for different I/O workloads [8], such as simple stripe, two dimensional stripe, and variable stripe. Data partition [10, 11] and replication [8, 12] techniques are also widely used to optimize data placement on file servers consistent with I/O workloads. Because data accesses for some scientific applications usually show several regular patterns [13], some data placement optimization techniques rely on the prior knowledge of data access patterns [9].

For applications that access I/O systems non-uniformly, simple stripe placement schemes are not able to obtain high performance. Segment-level placement scheme logically divides a file into several segments such that an optimal stripe size is assigned for each segment with non-uniform access patterns [6]. Server-level adaptive placement strategies adopt different stripe sizes on different file servers to improve the overall I/O performance of parallel file systems [7]. However, this work is not suitable for systems built on heterogeneous file servers. AdaptRaid addresses the load imbalance issue in heterogeneous disk array by optimizing data distribution with adaptive number of blocks [14]. However, it aims to reduce I/O latency rather than improving I/O bandwidth, and needs not to consider the network cost in data accesses.

Because SSDs exhibit obvious performance benefits over traditional HDDs, they are commonly integrated into parallel file system to improve I/O performance. Currently, most SSDs serve as a cache to traditional HDDs [15, 16] or persistent storage of file data [17,18]. Most of these techniques, however, are done

on a single file server. Our previous work CARL [4] selectively places file regions with high access costs onto the SSD-based file servers at the I/O middleware level.

All the aforementioned data placement techniques are effective in improving the performance of parallel file systems. However, there is little effort devoted on data placement in a hybrid parallel file system configured with HServers and SServers.

3 Design and Implementation

3.1 The Basic Idea of PADP

The proposed data placement scheme, PADP, aims to optimize the file data placement on heterogeneous file servers with varied-size stripes based on their storage performance. Figure 1 shows the idea of PADP. Similar to traditional data placement method, PADP dispatches the file data across file servers in a round-robin fashion, but the high-performance SServers are expected to store and process larger file stripes compared with low-performance HServers, so that all the file servers can complete processing their I/O requests within about the same time.

As we have mentioned previously, determining the appropriate stripe sizes on heterogeneous file servers is not an easy task due to three reasons. First, the file server performance can be impacted significantly by I/O patterns, such as request size, I/O operation (read or write), number of processes, etc. Second, the server performance is also related with their storage media characteristics. HServer and SServer have different performance behaviors even under the same I/O patterns. Finally, besides the storage cost, the overall I/O performance is a function of the underlying network, which should be considered when evaluating the data access performance. In order to address these issues, we first propose an analytical model to evaluate the access time of file requests. Then we use a linear programming method to determine the appropriate stripe size for each file server. Finally we describe the performance-aware data layout scheme.

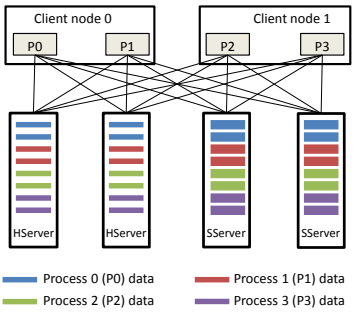


Fig. 1. Performance-aware data placement with varied-size stripes

3.2 Data Access Cost Analysis

Table 1. Parameters in cost analysis model

Symbol	Meaning
p	Number of client processes
c	Number of processes on one I/O client node
m	Number of HServers
n	Number of SSServers
h	Stripe size on HServer
s	Stripe size on SServer
r	Data size of one request
e	Cost of single network connection establishing
t	Network transmission cost of one unit of data
α_h	Startup time of one I/O operation on HServer
β_h	HDD transfer time per unit data
α_s	Startup time of one I/O operation on SServer
β_s	SSD transfer time per unit data

The cost is defined as the overall I/O time of each data access in hybrid PFSs. Table 1 lists the related parameters. Compared with previous work [6], this model is designed for heterogeneous environments. Please note that parameters for different types of requests on different storage media are differentiated when measuring the I/O time. The startup and transfer time are different between HServer and SServer. Generally, α_S is far smaller than α_H, and β_S is far greater than β_H because SSDs have no mechanical components. In addition, both α_H and α_S can be different between random and sequential operations, as we discuss in our experiments. Finally, while β_H is the same for reads and writes, β_S is different for them because writes on SSDs lead to background activities like garbage collection and wear leveling.

Before introducing the details of the model, we make following reasonable assumptions. First, all client nodes are separated from file servers in the system, which implies every data access involves network transmission. Second, the application-level parallel operations in each node are handled serially at the hardware layer, such as multiple network connections and storage accesses on file servers. Third, each I/O request involves all file servers, so that all servers can contribute to the aggregated I/O bandwidth. Assuming the stripe size of HServer and SServer is h and s respectively, the size of the data access is r, then

$$m \times h + n \times s = r \tag{1}$$

The data access cost mainly includes two parts: the network transmission time, T_{NET}, and the storage access time, T_{STOR}. Generally, T_{NET} consists of T_E and T_X. T_E is the network connection for data transmission, T_X is the data transfer time on network. T_{STOR} consists of T_S and T_T, the former is the startup time, and the latter is the actual data operation (i.e., read/write) time on storage media. Thus the cost of one data access can be described as follows.

$$T = T_E + T_X + T_S + T_T \tag{2}$$

T_E is determined by the number of establishing connections to each file server. As each file server is accessed by p processes and the p network connections have to be established serially, $T_E = pe$. T_X is determined by the amount of data accessed on each file server. For HServer, $T_X = pht$; for SServer, $T_X = pst$. In a parallel environment, the overall network transfer time is the maximum of all servers, thus $T_X = max\{pht, pst\} = pst$. On the other hand, each client node needs to establish network connections and transfer their data from all file servers serially, thus $T_E = c(m+n)e$ and $T_X = crt$. As network connections are affected by both file servers and client nodes, the network establish time T_E and network transfer time T_X are chosen in a prudential way when they are different at client nodes and file servers. If the number of network connections on client nodes is larger than that of file servers ($c(m+n) > p$), the number of client connections $c(m+n)$ is used. That is

$$T_E + T_X = \begin{cases} c(m+n)e + \max\{crt, pst\}, & c > \frac{p}{m+n} \\ pe + \max\{crt, pst\}, & \text{otherwise} \end{cases} \tag{3}$$

The startup time T_S and data transfer time T_T of each file server is only determined by the number of sequential I/O operations, namely the number of client processes assigned on that server. For HServer, $T_S = p\alpha_h, T_T = ph\beta_h$; for SServer, $T_S = p\alpha_s, T_T = ps\beta_s$. In a parallel environment, the storage cost T_{STOR} is determined by the maximal storage cost of all servers. Thus

$$T_S + T_T = p \times \max\{\alpha_h + h\beta_h, \alpha_s + s\beta_s\} \tag{4}$$

Based on Equation 3 and 4, the overall cost values of each data access are shown in Figure 2. This cost model provides a detailed analysis of completion time for data accesses in hybrid PFSs. Although there are several parameters in the model, for most applications, the runtime variables such as c, p, m and n are fixed for each run. In general, for a given system, e, t, α and β can be regarded as constants.

3.3 Determining the Optimal Stripe Sizes for Each File Server

Figure 2 shows that the data access cost T can be significantly impacted by the file server stripe sizes h and s. In other words, data placements with different stripe sizes lead to substantially variable access cost. In order to get the optimal I/O performance, the proposed data placement will find suitable stripe sizes

Condition	Network cost T_{NET}		Storage cost T_{STOR}
	Establish T_E	Transfer T_X	Startup T_S + I/O T_T
$p \leq c(m+n)$	$c(m+n)e$	$\max\{crt, pst\}$	$p * \max\{\alpha_h + h\beta_h, \alpha_s + s\beta_s\}$
$p > c(m+n)$	pe	$\max\{crt, pst\}$	$p * \max\{\alpha_h + h\beta_h, \alpha_s + s\beta_s\}$

Fig. 2. Cost formulas for hybrid PFSs

to minimize the data access cost in hybrid PFSs. Thus, the optimization problem can be described as minimizing function F described in Equation 5 while satisfying the size constraints described in Equation 1.

$$F = \max\{crt, pst\} + p \times \max\{\alpha_h + h\beta_h, \alpha_s + s\beta_s\} \tag{5}$$

According to the member values in the two maximum functions in Equation 5, such problem can be translated into four linear programming (LP) problems with two unknown variables representing the stripe size h and s. The final problem is to choose the values of h and s so as to minimize F as below.

Case 1:

$$\text{Minimize} \qquad F = crt + ph\beta_h \tag{6}$$

$$\text{s.t.} \begin{cases} mh + ns & = r \\ ps & \leq cr \\ \alpha_s + s\beta_s & \leq \alpha_h + h\beta_h \end{cases} \tag{7}$$

Case 2:

$$\text{Minimize} \qquad F = crt + ps\beta_s \tag{8}$$

$$\text{s.t.} \begin{cases} mh + ns & = r \\ ps & \leq cr \\ \alpha_h + h\beta_h & \leq \alpha_s + s\beta_s \end{cases} \tag{9}$$

Case 3:

$$\text{Minimize} \qquad F = pst + ph\beta_h \tag{10}$$

$$\text{s.t.} \begin{cases} mh + ns & = r \\ cr & \leq ps \\ \alpha_s + s\beta_s & \leq \alpha_h + h\beta_h \end{cases} \tag{11}$$

Case 4:

$$\text{Minimize} \qquad F = pst + ps\beta_s \tag{12}$$

$$\text{s.t.} \begin{cases} mh + ns & = r \\ cr & \leq ps \\ \alpha_h + h\beta_h & \leq \alpha_s + s\beta_s \end{cases} \tag{13}$$

The final stripe sizes of h and s are determined by the case where the objective function F achieve the smallest value among the four cases. Please note that the optimal h can be zero, which means placing file data only on the underlying SServers leads to better performance. As the linear program is expressed with two unknown variables, the search space is very small and solving the program requires acceptable time cost.

3.4 Performance-aware Data Placement Scheme

Based on the optimal stripe sizes h and s, PADP is able to achieve the optimal file data placement for data-intensive applications. This approach requires a prior knowledge of data access patterns of applications. As described in [9, 10], many HPC applications access their files with either regular data access patterns or predictable behaviors. For example, numerous tools were developed to trace I/O requests for these applications [13]. These applications are often executed on a computer cluster many times, and the file access patterns are generally independent of the data values stored. The request patterns can be learned from previous runs. Figure 3 shows the procedure of the optimal data placement

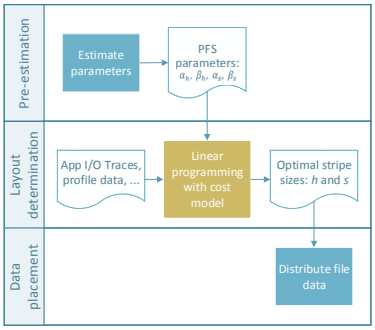


Fig. 3. The procedure of the performance-aware data placement scheme

scheme. Basically, the proposed data placement scheme consists of three phases: pre-estimation, layout determination, and data placement. In the pre-estimation phase, the related parameters in the cost model are estimated. As described previously, the network parameters, such as e and t, the storage parameters, such as α_h, β_h, α_s, β_s, and the system configuration parameters, such as m and n , can be regarded as constants. In the layout determination phase, the cost model and the linear programing method are used to calculate the optimal file stripe sizes h and s for HServers and SServers. In this phase, the applications access patterns, such as c, p, r are used as inputs. Determining the optimal stripe sizes is relatively fast since it requires solving only a small two-variable linear programming problem. In the data placement phase, the optimized file stripes can be used for the file data distribution for the applications, either by creating

new files for later runs of the applications, or adjusting the file layout by file copy operations in the existing parallel file systems.

3.5 Implementation

We have implemented a prototype of the performance-aware data placement scheme in OrangeFS.

Pre-estimation We use one file server in the parallel file system to test the startup time α and data transfer time β for HServers and SServers with sequential/random and read/write patterns. Please note that the parameters can vary with different I/O patterns. In addition, we use a pair of nodes (one client node and one file server) to estimate network parameters, the network connection establishing time e and network transfer time t. We repeat the tests with thousands of times (the number is configurable), and then calculate their average values, which are used as the parameter values.

Optimal Data Distribution Once obtaining the optimal stripe sizes for HServers and SServers, we use them to distribute file data among available file servers for better I/O performance. The OrangeFS file system supports an API for implementing specific variable stripe distribution by default. The variable stripe distribution is similar to simple stripe, except that the stripe size can be configured to be different on different file servers. In OrangeFS, parallel files can either be accessed by the direct PVFS2 interface or the POSIX interface. When using the direct PVFS2 interface, we utilize the "pvfs2-xattr" command to set the data distribution of directories where the application files are located. In addition, when a new file is created, we use the "pvfs2-touch" command with the "-l" option to specify the order of the file servers, so that the file stripe size h and s can be configured for the corresponding HServers and SServers accordingly.

3.6 Discussion

One concern of PADP is that it can potentially lead to more storage space consumption for SServers, which might perhaps be an unwanted feature by users. Fortunately, most file systems do not make full use of the storage space in the underlying devices. In practical system, this issue is not frequently encountered if the SSD space is enough. In the worst case, with the possibility of an SServer running out of its space, we design a data migration method to balance the storage space by moving data from SServers to HServers, so that the available remaining space on SServers can be guaranteed for new coming requests. This problem can also be addressed by using the hybrid PFS to store performance-critical data (e.g. frequently accessed data) and the PFS only on HServers to store the rest of the data.

4 Performance Evaluation

In this section, we evaluate the performance of the proposed data placement scheme through several benchmark-driven experiments.

4.1 Experimental Setup

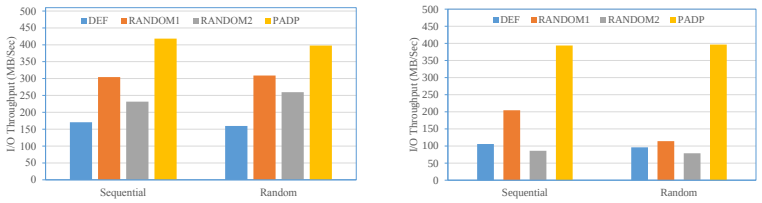
We conducted the experiments on a 65-node SUN Fire Linux cluster, where each node has two AMD Opteron(tm) processors, 8GB memory and a 250GB HDD. 16 nodes are equipped with additional OCZ-REVODRIVE 100GB SSD. All nodes are equipped with Gigabit Ethernet interconnection. The parallel file system is OrangeFS 2.8.6. Among the available nodes, we select eight nodes as client computing nodes, eight nodes as HServers, and eight nodes as SServers. By default, the hybrid OrangeFS file system is built on six HServers and two SServers.

We compare three data placement schemes: the default scheme (DEF), the random scheme (RANDOM), and the proposed PADP scheme. In DEF, the file data is placed across all file servers with a fixed-size stripe of 64KB; in RANDOM, the file stripe sizes are randomly selected. To be simple, the stripe size pair $< h, s >$ is used in the following sections, which means the stipe sizes on HServers and SServers are h and s respectively. The popular benchmark IOR [19] is used to test the performance.

4.2 IOR Benchmark

Performance with Different Read Write Modes Unless otherwise specified, the IOR benchmark runs with 8 processes, each of which performs I/O operations in individual mode on a 10GB shared file. The request size is kept to 512KB. Figure 4 demonstrates the I/O performance of IOR with sequential and random I/O access mode under the three data placement schemes. In the figure, the randomly selected stripe size pair is <32KB, 96KB> in RANDOM1, and <96KB, 32KB> in RANDOM2. For PADP, the optimal stripe sizes for sequential and random read, sequential and random write, are <28KB, 100KB>, <20KB, 108KB>, <24KB, 104KB>, and <36KB, 92KB> respectively. From the results we can observe that PADP has the best performance of all schemes. By using the optimal stripe sizes for HServers and SServers, PADP can improve read performance by up to 149.2% over DEF with all I/O access modes, and write performance by up to 271.8%. Compared with RANDOM1 and RANDOM2, PADP can improve the read performance by up to 80.6% and write performance by up to 357.1% for all I/O access modes. This shows that the idea of PADP works well and the stripe size determining formula of PADP is effective.

In order to give a detailed explanation, Figure 5 plots the I/O time of each file server during a 10-second IOR execution period when IOR performs sequential read operations under the three schemes. The I/O time is normalized to that of the minimum I/O time of all file servers. Among the eight file servers, server 0 to 5 are HServers, and the rest are SServers. From Figure 5, we can observe



(a) Read throughput (b) Write throughput

Fig. 4. Throughputs of IOR under different placement schemes with different I/O modes

that the I/O loads of HServers and SServers are severely skewed under scheme DEF and RANDOM. In contrast, the optimal data placement scheme PADP can significantly eliminate the load imbalance among file servers. Thus, PADP improves the file system performance.

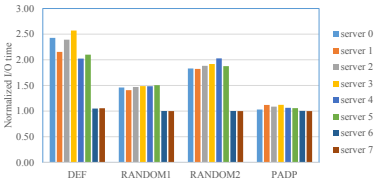
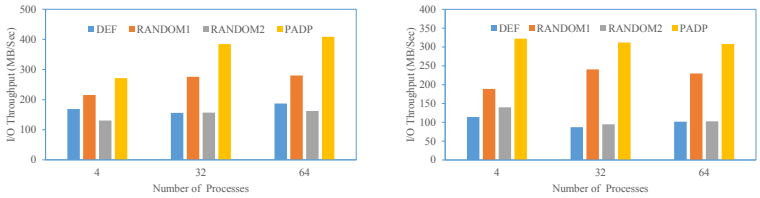


Fig. 5. I/O time on each node under different data placement schemes

Performance with Different Number of Processes The I/O performance is also evaluated with different number of processes. The IOR benchmark is executed under the random access mode with 4, 32 and 64 processes. In this test, RANDOM1 and RANDOM2 use the same stripe size configuration as previous test. As show in Figure 6, the results are similar to the previous test. PADP has the best performance among the three schemes. Compared with DEF, PADP improves the read performance by 60.8 %, 146.3%, and 118.4%z respectively with 4, 32 and 64 processes, and write performance by 182.3%, 257.8 %, and 202.7%. Compared with RANDOM, PADP can brings a read performance improvement by up to 107.9%, 145.2%, and 151.6% respectively with 4, 32 and 64 processes, and write performance improvement by up to 130.3%, 228.8%, and 200.3%. These results show that PADP has very good scalability with the number of I/O processes.

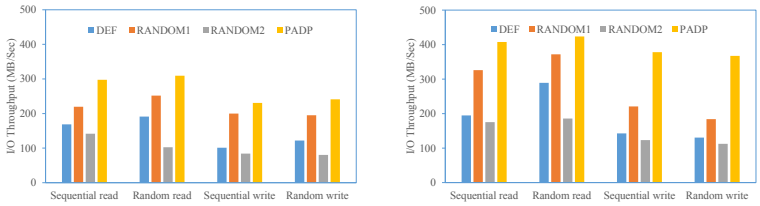
Performance with Different Request Sizes Figure 7 demonstrates the I/O performance of IOR with request size of 128KB and 2048KB. The number of processes is fixed to 16, and IOR issues random requests. Figure 7(a) shows



(a) Read throughput (b) Write throughput

Fig. 6. Throughputs of IOR with varied number of processes

the result for the relatively small requests. We can observe that PADP can improve the read performance by up to 76.3%, and write performance by up to 127.9% in comparison with the default data placement scheme DEF. Compared with RANDOM, PADP also has better performance: the read performance is increased by up to 201.7 %, and write performance is increased by up to 199.4%. When the request size is 128KB, it is worth noting that the optimal stripe sizes in PADP are <0KB, 32KB> for all I/O modes. This implies that distributing the file only on the four SServers leads to the highest I/O performance if the requests are relatively small. For larger size 2048KB, PADP distributes the file across both HServers and SServers to achieve optimal performance, as shown in Figure 7(b). This is because all servers are working cooperatively and this can lead to better I/O performance for large requests. These results show that PADP has a good scalability for different request sizes.

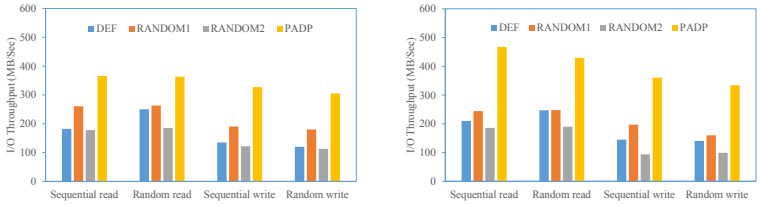


(a) Request size is 128K (b) Request size is 2048K

Fig. 7. Throughputs of IOR with varied request sizes

Performance with Different Server Configurations The I/O performance is examined with varied ratios of SServers to HServers. The OrangeFS is built using HServers and SServers with the ratios of 5:3, and 3:5. Figure 8 shows the average I/O bandwidth with different file server configurations. As it can be seen from the results, PADP can improve I/O throughput for both data read and write. When the ratio is 5:3, the randomly selected stripe sizes of RANDOM1 and RANDOM2 are <34KB, 114KB> and <82KB, 34KB> respectively. Compared with DEP, the read performance improves by up to 100.9% , and write

performance improves by up to 154.1%. We can observe that PADP can increase the read performance by up to 105.9%, and write performance by up to 169.6% the RANDOM scheme. When the ratio is 3:5, the randomly selected stripe sizes of RANDOM1 and RANDOM2 are <29KB, 85KB> and <89KB, 49KB> respectively. We can observe that PADP has the similar behavior. In the experiments, read and write performance improved as the number of SServers increased. This is because the I/O performance of SServers is efficiently utilized by PADP. By using the optimal stripe sizes determined by the linear programing method in this paper, PADP can significantly improve the hybrid file system performance with all file server configurations.



(a) 5HServers : 3SServers (b) 3HServers : 5SServers

Fig. 8. Throughputs of IOR with varied file server configurations

5 Conclusions

In this study, we have proposed a performance-aware data placement (PADP) scheme, which distributes data across HDD and SSD file servers with adaptive stripe sizes based on their storage performance. We have presented the proposed PADP data placement optimization scheme and implemented it in the OrangeFS file system. Essentially, PADP provides a better matching of data access characteristics of an application with the storage capabilities in the file servers of the underlying heterogeneous file system. Experimental results of representative benchmark show that PADP can significantly improve the file system performance.

References

1. "Orange File System," http://www.orangefs.org/.
2. S. Microsystems, "Lustre File System: High-performance Storage Architecture and Scalable Cluster File System," Tech. Rep. Lustre File System White Paper, 2007.
3. F. Chen, D. A. Koufaty, and X. Zhang, "Understanding Intrinsic Characteristics and System Implications of Flash Memory Based Solid State Drives," in *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems*, 2009, pp. 181–192.

4. S. He, X.-H. Sun, B. Feng, X. Huang, and K. Feng, "A Cost-Aware Region-Level Data Placement Scheme for Hybrid Parallel I/O Systems," in *Proceedings of the IEEE International Conference on Cluster Computing*, 2013.
5. S. He, X.-H. Sun, and B. Feng, "S4D-Cache: Smart Selective SSD Cache for Parallel I/O Systems," in *Proceedings of the International Conference on Distributed Computing Systems*, 2014.
6. H. Song, Y. Yin, X.-H. Sun, R. Thakur, and S. Lang, "A Segment-Level Adaptive Data Layout Scheme for Improved Load Balance in Parallel File Systems," in *Proceedings of the 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2011, pp. 414–423.
7. H. Song, H. Jin, J. He, X.-H. Sun, and R. Thakur, "A Server-Level Adaptive Data Layout Strategy for Parallel File Systems," in *Proceedings of the IEEE 26th International Parallel and Distributed Processing Symposium Workshops and PhD Forum*, 2012, pp. 2095–2103.
8. H. Song, Y. Yin, Y. Chen, and X.-H. Sun, "A Cost-Intelligent Application-Specific Data Layout Scheme for Parallel File Systems," in *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, 2011, pp. 37–48.
9. Y. Yin, J. Li, J. He, X.-H. Sun, and R. Thakur, "Pattern-Direct and Layout-Aware Replication Scheme for Parallel I/O Systems," in *Proceedings of 27th IEEE International Parallel and Distributed Processing Symposium*, 2013.
10. Y. Wang and D. Kaeli, "Profile-Guided I/O Partitioning," in *Proceedings of the 17th Annual International Conference on Supercomputing.* San Francisco, CA, USA: ACM, 2003, pp. 252–260.
11. S. Rubin, R. Bodik, and T. Chilimbi, "An Efficient Profile-Analysis Framework for Data-Layout Optimizations," *ACM SIGPLAN Notices*, vol. 37, no. 1, pp. 140–153, 2002.
12. H. Huang, W. Hung, and K. G. Shin, "FS2: Dynamic Data Replication in Free Disk Space for Improving Disk Performance and Energy Consumption," in *Proceedings of the 20th ACM Symposium on Operating Systems Principles.* New York, NY, USA: ACM, 2005, pp. 263–276.
13. Y. Yin, S. Byna, H. Song, X.-H. Sun, and R. Thakur, "Boosting Application-Specific Parallel I/O Optimization Using IOSIG," in *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2012, pp. 196–203.
14. T. Cortes and J. Labarta, "Taking Advantage of Heterogeneity in Disk Arrays," *Journal of Parallel and Distributed Computing*, vol. 63, no. 4, pp. 448–464, 2003.
15. T. Pritchett and M. Thottethodi, "SieveStore: a Highly-Selective, Ensemble-level Disk Cache for Cost-Performance," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, 2010, pp. 163–174.
16. X. Zhang, K. Liu, K. Davis, and S. Jiang, "iBridge: Improving Unaligned Parallel File Access with Solid-State Drives," in *Proceedings of 27th IEEE International Parallel and Distributed Processing Symposium*, 2013.
17. Q. Yang and J. Ren, "I-CASH: Intelligently Coupled Array of SSD and HDD," in *Proceedings of the IEEE 17th International Symposium on High Performance-Computer Architecture*, 2011, pp. 278–289.
18. F. Chen, D. A. Koufaty, and X. Zhang, "Hystor: Making the Best Use of Solid State Drives in High Performance Storage Systems," in *Proceedings of the international conference on Supercomputing*, 2011, pp. 22–32.
19. "Interleaved Or Random (IOR) Benchmarks," http://sourceforge.net/projects/ior-sio/, 2014.