Improve Throughput of Storage Cluster Interconnected with a TCP/IP Network Using Intelligent Server Grouping

Xuechen Zhang, Guiquan Liu, Song Jiang

Improve Throughput of Storage Cluster Interconnected with a TCP/IP Network Using Intelligent Server Grouping

Xuechen Zhang[1] Guiquan Liu[2] Song Jiang[1]

[1] Wayne State University,
[2] University of Science and Technology of China
xczhang@wayne.edu, gqliu@ustc.edu.cn, sjiang@eng.wayne.edu

Abstract. Cluster-based storage systems connected with TCP/IP networks are expected to achieve a high throughput by striping files across multiple storage servers. However, for the storage system interconnected with the TCP/IP network, several critical issues, like Incast effect and data access interference, invalidate the assumption that higher access parallelism always results in increased I/O throughput. To address this issue, we propose a new file striping strategy, named as storage server grouping (SSG), which changes file striping pattern across the storage servers based on the analysis of file popularity and impact of the number of storage servers on the clients' perceived performance (I/O speedup) to reduce the interference of data accesses to popular files and avoid dramatic reduction of system throughput caused by the Incast effect. Our experimental evaluation shows that SSG can improve I/O throughput by 22.1% on average.
Keywords: the Incast Effect, Interference, File Striping, Lustre.

1 Introduction

Cluster-based storage systems are adopted as an increasingly important platform for supporting applications that demand high I/O performance for large-scale data access [2]. The building blocks of many such systems are a number of small and less capable storage servers which are usually connected with commodity low-cost and high-performance Gigabit TCP/IP-Ethernet networks [1]. Figure 1 illustrates a typical cluster-based storage system. The TCP/IP network usually has sufficiently high link bandwidth (more than 1Gbps) but the switch has very limited size of buffer to store and transfer data.

In the storage system managed by parallel file systems, such as Lustre [3], GPFS [4], and PVFS [5], the data accessed by one request from a client can be distributed on multiple storage servers. Accordingly, a client actually disassembles such a request into several small sub-requests, each to a storage server for a piece of data stored on the server. The servers then concurrently access the pieces of data for each sub-request. In this way, file striping helps achieving high data access parallelism, which is expected to improve I/O throughput. Parallel file systems usually hide the details on file placement and access protocol. However, many of them, such as Lustre, provide interfaces allowing the owner of files to set some critical data striping parameters, such as striping unit size, striping factor (the number of storage servers to store files), and striping index (the first storage server for the striping).

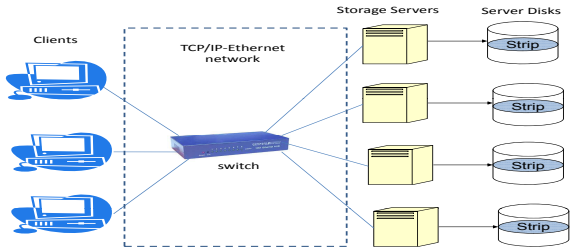

Fig. 1. A typical cluster-based storage system is composed of clients, TCP/IP-Ethernet network including a switch, and a number of storage servers managed by a parallel file system. In the system, a file is divided into several strips, each stored on a storage server according to a set of pre-defined configuration parameters.

We conducted experiments to study performance of such systems. Our key observations are 1) high disk interference on storage servers can be incurred by clients' excessive exploitation of their data access parallelism without coordination; 2) the limited switch buffer size might cause TCP throughput to collapse; and 3) data access to popular files intensifies the disk head contention among clients on I/O servers. These observations make the conventional approaches less effective, which focus on striping data over many servers and balancing I/O load over the servers [6] in order to explore the parallelism of data accesses for a high I/O throughput.

To address the issue, we propose SSG (Storage Server Grouping) as a framework to automatically generate file striping parameters in an on-line manner. SSG uses the proposed I/O speedup model to find the optimal number of storage servers before a file is striped across storage servers. The I/O speedup model is trained by using relative machine learning technique [11] to correlate the number of storage servers with I/O performance of a workload. SSG keeps tracking file popularity and intelligently separates files into different server groups by setting the striping index, reducing data access interference on each group. SSG also periodically tunes the file striping parameters based on the I/O workload characteristics profiled on line. We have implemented the SSG scheme on top of Lustre parallel file system. Our experimental results show that SSG can improve system-wide I/O throughput by up to 38.6% and 22.1% on average.

The remainder of this paper is organized as follows. Section 2 presents our experimental observations. Section 3 discusses related work. Section 4 presents the design of the SSG scheme in detail. Section 5 delves into experimental evaluation, and section 6 concludes the paper.

2 Experimental Observations

To study the performance of TCP/IP network based storage cluster, we set up an experimental platform consisting of sixteen nodes, eight configured as compute nodes, and the other eight as I/O nodes, managed by the Lustre parallel file system. File data was striped over the I/O nodes. We used the default Lustre file striping setting, whose file striping size is 64KB, striping factor is 8, and striping index is 0. (More details of the experimental platform are described in Section 5.) We used the Iozone benchmark [7]

for three types of reads: the sequential read, the random read, and the backward read in the throughput mode, which we can specify the number of active threads in its running.

2.1 The Incast Effect

Increasing the number of I/O servers of a storage cluster connected with TCP/IP networks does not necessarily improve the effective network bandwidth. As we know, the Ethernet switch has only very limited built-in memory. For example, the widely used HP Procurve 2848 Ethernet switch has only 16MB memory. When the incoming packets arrive too fast to be buffered in the cache, some of them would have to be discarded due to the limited buffer size, which will cause TCP timeout and re-transmission of the packets from the servers. In this scenario, the servers cannot serve the next I/O request from clients until the timeout is detected and all the discarded packets are re-transmitted. The worst scenario, in which network delay is seriously increased, network links are idle, and system throughput is collapsed, is called the Incast effect [1, 9].

To observe the impact of Incast effect on the I/O throughput, we run the Iozone benchmark for the three types of reads: sequential read, random read, and backward read on a client with 512KB request size. An Iozone thread on the client accesses a file which is striped over different numbers of I/O servers, ranging from 1 to 8. Figure 2 shows that the I/O throughput does not increase with the increase of the number of I/O servers. The throughput even becomes lower by 15% when the number of I/O servers increases from 6 to 8, which is a clear indication of the Incast effect.

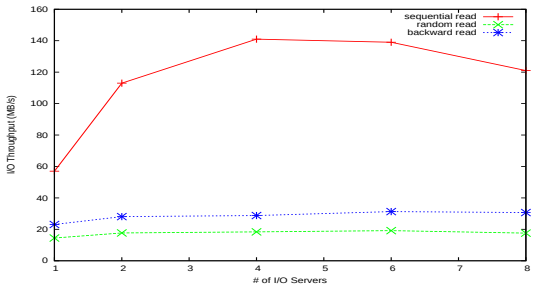

Fig. 2. System-wide I/O throughput observed at the application level when running the Iozone benchmark on a single client as the number of I/O servers increases. Request size of these tests is 512KB. Incast effect happens when the number of I/O servers is 8, making the system throughput 15% less than the maximum.

However, the Incast effect does not necessarily happen for all workloads of any types of access patterns. For example, when average request size of a workload is less than striping unit size, the Incast effect will be less likely to occur since the chance of switch buffer saturation caused by the requested data from multiple servers in response to a single request is very low. However, we cannot simply always set the striping unit size larger than the average request size, because this would essentially eliminate data access parallelism. A better way to alleviating the Incast effect and improving the networking efficiency is to carefully select the number of storage servers for file striping. However, it would take too much time for administrators to find the number simply by human

experience. SSG uses a proposed I/O speedup model to find the optimal number of storage servers before a file is striped across storage servers to avoid the Incast effect.

2.2 Interference among Requests from Different Clients

Usually when a storage server is dedicated to serve requests from one client, the efficiency of its disks can be well maintained. Compared with increasingly large disk capacity used for storing data for many clients, the amount of data accessed by one client is limited, or the disk region covered by requests from one client is relatively small. This allows the disk heads to move in relatively short distances in continuously serving requests from the same client and to keep relatively high disk throughput. However, serving requests from multiple clients concurrently requires disk heads to move in longer distances among disk regions storing data belonging to different clients. The expensive mechanical operation would dramatically reduce both the I/O throughput received by each individual client and the throughput of the entire disk-based server cluster. This inter-client interference becomes more intensive when data are striping over more storage servers and thus each server serves requests from more clients. The worst scenario is to have a long disk-head movement across client-data regions in serving every request.

To observe impact of the interference on the I/O throughput, we run the Iozone benchmark on different numbers of clients, ranging from 1 to 16, while keeping the number of servers constant, so that each I/O server will serve increasingly large number of concurrent requests. We test two scenarios. In the scenario for light interference, the Iozone thread on each client accesses one set of files, which are distributed on close disk regions. In the scenario for heavy interference, the thread on each client accesses a different set of data files on the disks. In both scenarios, sequential access pattern is used. Figure 3 shows the system-wide I/O throughputs in these two scenarios. The results clearly show that the interference has a major impact on the I/O efficiency. For example, with 16 clients, the throughput with the heavy-interference scenario is only 74% of that with a light interference.

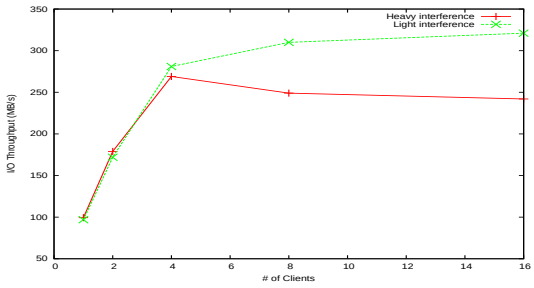

Fig. 3. System-wide I/O throughput observed at the application level when running Iozone benchmark on multiple clients while keeping the number of I/O servers constant. Data access to different set of disk regions (heavy interference) results in much more overhead when the number of clients is larger than 6.

The interference can be alleviated by serving many requests from the same client in a batch before moving disk heads to serve requests from other clients. In this way, the long-

distance movements of disk heads can be reduced and disk throughput can be improved. This technique is especially effective with asynchronous I/O, in which a number of requests from the same process are pending and ready to be scheduled together. However, for the synchronous I/O, which is commonly used in most applications, a technique named as anticipatory scheduling [10] is used to serve multiple requests from the same process together. In the scheduling policy, the disk will wait for the next request from the same process to serve after it served its current request, even though there are pending requests from other processes. For the policy to be effective, the time gap between two consecutive requests from the same process, or thinktime of the process, must be small so that the gain from the reduced disk head movement is larger than the loss from the disk idling. However, the thinktime can be significantly increased if a file is striped over many I/O servers because a request is considered to be completed only when its requested data on all the servers are accessed. As the disk operations on different servers are not synchronized, the thinktime can be significantly increased with the increase of the parallelism, or the number of servers involved, in the serving of a request. While the parallelism can be beneficial to I/O throughput when the interference is not serious, it can hurt the throughput when the interference is intensive.

Our SSG scheme addresses the issue by making a trade-off between potentially high throughout due to access parallelism and excessive interference due to high parallelism, looking for the optimal number of servers for the file striping.

2.3 Intensified Data Access Contention

Data access contention is intensified when there are multiple popular files in the same storage system and all the files are simultaneously accessed by different clients. The default consistently aggressive striping for data access parallelism over the same set of servers can lead to a high probability in which one server simultaneously services requests for actively used files from many clients. This increased intensity of competition for disk services causes high inter-client interference. System throughput degradation caused by data accesses to popular files can be alleviated through separating the popular files into different storage server groups using the SSG scheme proposed in this paper. One of the SSG components is responsible for tracking the popularity of each file and, accordingly, placing popular files into different groups by carefully selecting file striping parameters. Therefore, data access contention to popular files can be reduced in each group.

All these observations indicate that optimizing file striping parameters on line is critical to avoiding the Incast effect and reducing the interference overhead caused by aggressive striping and data access contention to popular files. SSG can automatically tune these parameters on line to achieve better I/O performance than the traditional default striping method.

3 Related Work

There are a large of body of work about modeling I/O storage systems, data-aware storage resource management, and solutions to the Incast effect.

3.1 Research on Storage System Simulation

Some existing disk simulators, like Disksim [15, 14], can help to accurately predict I/O workload performance from single disks or disk arrays by using software to simulate device behaviors. But developing such kind of simulators requires extensive expertise and knowledge about details of hard disks and disk arrays. Furthermore, it is even harder to have a simulator for the cluster-based storage system because many dynamics on the networks are involved. Analytic models which are presented in papers [16–18] describe device behavior with a set of formulae. They are computationally efficient but have the same drawbacks as the simulation methods.

Machine learning techniques can also be used to model a storage system. Wang et al. proposed a scheme to predict storage device performance by using CART model [12]. They treat an I/O device as a function whose input is workload characteristics, either at the request level or at the workload level, and whose output is the request response time. This CART model does not require a priori knowledge on storage devices and is able to predict I/O performance of single disks and disk arrays with less than 20% median relative error on average, as reported in the paper. Mesnier et al. proposed a relative fitness model [11], which captures the differences between a pair of devices instead of predicting the absolute performance of workloads. Their method can reduce prediction error by as much as a factor of two when compared to absolute models. Therefore, we choose to build our I/O speedup relative fitness model.

3.2 Research on Data-aware Storage Resource Management

Yu et al. proposed a hierarchical striping method [20] to reshape the data access pattern to storage servers, resulting in an improved aggregated bandwidth. But they did not analyze I/O speedup of a workload for file striping or consider the data access interferences to popular files. Kosar proposed a job scheduling approach [19] to explore data locality of distributed storage resources on-line by analyzing the processing of applications. However, the job-level approach does not have enough knowledge on dynamic changes of workload characteristics or current network statuses to reduce the Incast effect.

3.3 Research on Solutions to the Incast Effect

The Incast effect can be alleviated by increasing the switch buffer size or tuning TCP flow control parameters such as package loss timeouts [1]. However, solutions at the network level are less adaptive to the changes of workload behaviors and are not cost-effective as switches with large buffers can be expensive. The problem can also be addressed at the application level [1, 24]. Several possible application-level methods are proposed in [21], including increasing request size, global scheduling of data transfers, and limiting the number of synchronously communicating servers.

4 Storage Server Grouping Scheme

4.1 I/O Speedup

We introduce the concept of I/O speedup for each client's workload to characterize the reduced workload running time due to the use of increased number of storage servers.

The I/O speedup for k severs is defined as the ratio between the workload running time when its data is striped on the k storage servers and the running time when one dedicated server is used. We use a dedicated server as the baseline for the comparison to make sure that no dynamically varying interference is involved in the metric. The dedicated server is named as a reference server. If the I/O speedup for k servers is greater than 1, then a workload can benefit from striping its file across the k servers. Since the I/O speedup not only depends on the characteristics of a workload but also depends on the configuration of networks such as the size of the buffer in the network switch, it would be a time-consuming task for administrators to find the number of servers to obtain the highest speedup. The I/O speedup of a workload is automatically determined in our SSG scheme, using relative fitness modeling approach [11] rather than traditional direct modeling approach since the relative fitness modeling provides higher prediction accuracy [11].

The I/O speedup relative fitness model is used to calculate the I/O speedup for k servers and expressed as the following function,

$$\frac{Time_{ref}}{Time_k} = RF_{ref \to k}(WC_{ref}, Perf_{ref}, Util_{ref}),$$

where WC_{ref} represents the workload characteristics, such as request size and request arrival time, which are profiled when the workloads run on the reference server; $Perf_{ref}$ is a performance metric such as latency; and $Util_{ref}$ is a vector of values, such as the devices' cache utilization and cache hit ratio. After we have built up the model, or specifically, have obtained function $RF_{ref \to k}$, we only need to measure the performance characteristics, such as WC_{ref}, $Perf_{ref}$, and $Util_{ref}$, on the reference server for a newly admitted workload and then use the models to derive the maximal I/O speedup of the workload. We instrument the MPI-IO library for parallel I/O benchmarks or source code of applications for filesystem benchmarks to collect the required measurements. Details on the model training are described on Section 3.1.1. If the total number of storage servers is N, SSG needs to generate N models, each on one possible striping factor k ($k = 1, 2, ..., N$).

Figure 4 shows two steps involved in the prediction of the I/O speedup. The first step is fitness off-line tests for building I/O speedup models. The models are trained before a cluster-based storage system starts to serve requests from clients using fitness tests and will be periodically updated using the actually measured I/O speedup values. The second step is I/O speedup on-line prediction during workload admission control. More specifically, SSG runs an application to obtain the characteristics, performance, and resource utilization of a workload on the reference server before actually striping the data set of the workload over the storage servers. Based on these profiling data and I/O speedup models, SSG predicts I/O speedup of the workload on different number of servers. After enumerating all I/O speedup relative fitness models, SSG can determine the optimal number of servers for file striping.

Training an I/O Speedup Model To train an I/O speedup model, SSG uses a synthetic workload generator to generate training requests, which should exhibit adequate coverage of storage system characteristics. The training data is obtained when profil-

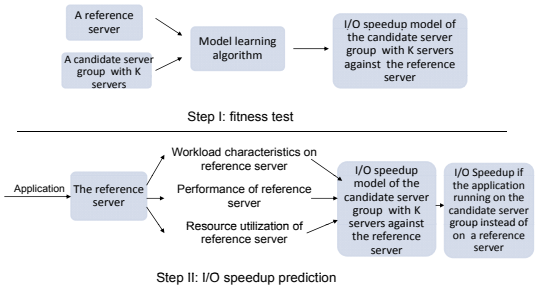

Fig. 4. There are two steps for the prediction of I/O speedup. In step I, SSG uses fitness tests to construct I/O speedup relative fitness model for a candidate server group with specific number of servers. In step II. SSG predicts the I/O speedup for running time of an application on the reference server over its running time on a candidate server group of k servers.

ing the training workloads running on the reference server and candidate server groups. Each training sample includes the following variables [12]:

$$\begin{aligned} Request_i = \{&TimeDiff_i(1), ..., TimeDiff_k(k), \\ &LBN_i, LBNDiff_i(1), ..., LBNDiff_i(l), \\ &Size_i, RW_i, Seq_i, Cache_i, Latency_i\} \end{aligned}$$

Where $TimeDiff_i(k) = ArrivalTime_i - ArrivalTime_{i-2^{k-1}}$ and $LBNDiff_i(l) = LBN_i - LBN_{i-l}$. The first two groups of parameters capture temporal and spatial locality of a workload. Seq_i indicates whether a request is a sequential access. $Cache_i$ indicates whether a request hits in memory buffer. $Size_i$ represents size of a request and RW_i represents read or write attribute of a request. $Latency_i$ is the service time of a request. The two parameters, k and l, determine how far we look back for request bursts and locality [12]. The value of k and l is pre-defined before fitness tests.

SSG uses REPTree [13] to train the I/O speedup relative fitness model. For each training request in the workloads, the predicted variable is $\frac{Latency_{ref}}{Latency_{target}}$, where $Latency_{ref}$ is the request service time on a reference server and $Latency_{target}$ is the request service time on a candidate server group, and the predictor variables are a variable vector $\{TimeDiff_{ref}, ..., LBN_{ref}, ..., Size_{ref}, RW_{ref}, Seq_{ref}, Cache_{ref}\}$.

4.2 Harmoniousness Analysis based on File Popularity

Since there will be several server groups that have the same striping factor, SSG has to decide in which group a workload should join in by setting a striping index for a file. This process plays a critical role in reducing the contention due to serving many popular files concurrently on the same group of servers. Accordingly, a file accessed by a workload prefers to join in a group with fewer number of popular files.

We define file popularity as the access frequency of the file during each time slot T_{slot} and define a harmoniousness value H of a server group as the sum of popularity of 20% most popular files in the group. A fresh file is conservatively assigned a maximal access frequency Feq_{max} even though no requests have accessed it.

4.3 Putting it All Together

The whole procedure is an iterative process, including feedback control in looking for the best striping parameters for a file. Figure 5 shows the framework of the SSG scheme. During the workload admission control, SSG scheme first finds the number of I/O servers, where the workloads can achieve highest I/O speedup. Then SSG chooses the group that has the smallest harmoniousness value for workload files to stripe over by setting striping index as the index of the first server in the group. File popularity tracking component is responsible for periodically updating the file popularity by communicating with each tracking thread running on clients. The striping unit size is assigned as the average request size of a workload divided by the number of I/O servers in its server group.

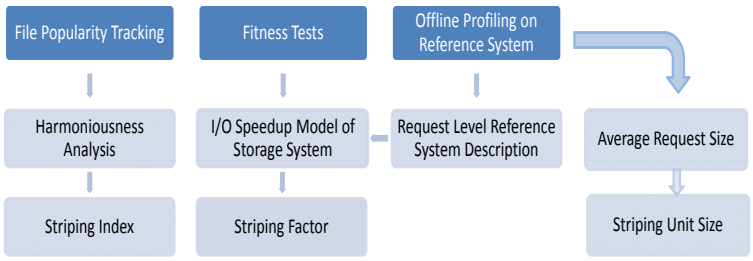

Fig. 5. The SSG scheme. In the workload admission control, SSG initializes striping parameters for files to be accessed. The file popularity tracking keeps updating popularity of each file periodically until the file is deleted.

The overhead of the SSG scheme includes the time for profiling applications on the reference server and the time for tracking file popularity. With limited loss of the prediction accuracy, SSG can approximate the characteristics of a workload by sampling I/O requests in a relatively coarse granularity during its running period. This profiling time can also be overlapped with the job scheduling time in high performance computing environment or request queueing time in other client-server environment. In addition, since SSG uses thread communication for file popularity tracking, the frequency for communication can be reduced for reduced overhead if data access has good locality.

5 Performance Evaluation and Analysis

We have implemented a prototype of the proposed SSG scheme over the Lustre parallel file system [3]. In this section, we will describe the experiment environment, evaluate prediction accuracy of I/O speedup model, and show the performance of the SSG scheme in different scenarios compared with static aggressive file striping policies to demonstrate that how SSG would respond to the Incast effect and variations of file popularity. We use (a, b, c) to represent the file striping triplets, where a is the striping unit size, b is the striping index, and c is the striping factor.

5.1 Experimental Setup

Our experiments are conducted on a dedicated cluster-based storage system with 16 nodes. All nodes are of identical configuration, each with a dual 1.6GHz Pentium processor, 1GB memory, and an 80GB SATA hard disk. The cluster uses the Lustre parallel file system (version 1.6.5.1), in which eight nodes were configured as storage servers and one of them also as a metadata server. The default striping triplet in our Lustre environment is (64KB, 0, 8). We also chose one server with the same configuration as the reference server. Each node runs Redhat Enterprise Linux (version 4.5) with kernel-2.6.9. All nodes are connected through a switched Gigabit Ethernet. The switch is D-Link DGS-1016D with 16 ports and 340Kbytes built-in buffer memory [22].

5.2 I/O Workloads

We choose two kinds of I/O workloads, synthetic workloads and real workloads, in the experiments. Synthetic workloads are produced by a workload generator and used to build I/O speedup models. For this purpose we ensure that there is an adequate coverage of characteristics of workloads that are possibly presented to the storage system, including data access locality, the number of burst requests, and workload size. To evaluate the performance, we use I/O workloads from profiling of real benchmark running, such as *mpi-io-test* from PVFS2 software package, *noncontig* from the Parallel I/O Benchmarking Consortium at Argonne National Laboratory [23], and Iozone filesystem benchmark. The first two benchmarks are used to test I/O system performance through parallel I/O interfaces, while the last one generates POSIX-compatible Linux file operations to evaluate filesystem performance.

In the *mpi-io-test* benchmark , each process accesses (reads or writes) one segment of contiguous data at a time and all the parallel processes access several segments in a row. Benchmark *noncontig* uses complex MPI derived data structure *vector* to access a file which is considered to be a two-dimensional array by each process. There are several columns in the array and each process accesses a column of the array, starting at row 0 of its designated column. In one row of a column there are *elmtcount* elements of the MPI_INT type. So the width of a column is *elmtcount**sizeof(MPI_INT). In each call, the total amount of data accessed by the processes is fixed, determined by the buffer size, which is 16MB in our experiment. That is, the larger *elmtcount*, the more small pieces of data, which are non-contiguous to each other, that are accessed by each process. Each process only reads or writes one file in its running. For the Iozone benchmark [7], we calculate system-wide I/O throughputs of sequential read, random read, and backward read workloads.

5.3 Prediction Accuracy of I/O Speedup Models

In this section, we will evaluate the prediction accuracy of the I/O speedup models generated through the fitness tests[2] by using a metric, *relative error x%*, which is defined

[2] In fitness tests, we use 80% of synthetic workloads as training data set for discovering the I/O speedup relations, and the other 20% as test data set to evaluate prediction accuracy of this REPTree learning approach.

as $|T_m - T_p| * 100/T_m = x\%$, where T_m is the latency measurement (service time) of a request and T_p is predicted latency of the request. The smaller the relative error is, the better the predicted results are. We ran the *noncontig* benchmark on the reference server twice with one process and two processes respectively to get profiling data. Then we use the data and I/O speedup models to predict the I/O speedup of each request on N candidate server groups with different number of storage servers. Figure 6 shows the CDF curves of relative error of all the predicted latency of requests. Each curve has a long tail, indicating that while the majority of predictions are quite close to measured latency, there is a small percentage of relative errors that are greater than 50%.

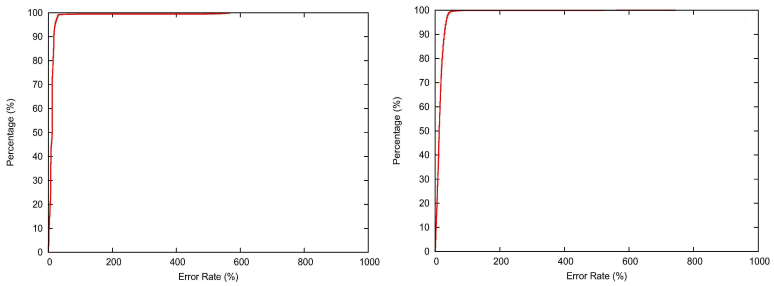

Fig. 6. CDF of the prediction error of predicted latencies when the noncontig benchmark runs with single process (left figure)or two concurrent processes (right figure). Over 90% of predictions are accurate within 16.7% error on average.

5.4 POSIX-I/O Experiments: Iozone Benchmark

We repeat the experiments described in Section 2.2 with the SSG scheme enabled. Iozone threads running with different numbers of clients sequentially read different files. Since all the files accessed by the clients have the same popularity, SSG isolates the accesses to each file, which is striped over a separate server group so that interference of data access from different threads is reduced as much as possible. Figure 7 shows the results of these tests. When the overhead due to disk-head movements becomes a dominating factor to system performance, SSG scheme helps to increase I/O throughput by up to 38.6% in the experiments, compared with the default aggressive setting. For the random and backward read, SSG achieves roughly the same improved performance. From this figure we also find that when the number of clients is fewer than 4, the I/O throughput with SSG scheme is slightly smaller than the default setting. This is because client I/O interference on disks can be alleviated through I/O scheduling like anticipatory scheduling when interference of data accesses is not intensive.

5.5 Scenario I: Parallel I/O Experiments

This experiment is to demonstrate the effectiveness of the SSG scheme to alleviate the Incast effect and to show how server grouping can lead to better I/O throughput. We execute four processes, each running the *mpi-io-test* benchmark with four different

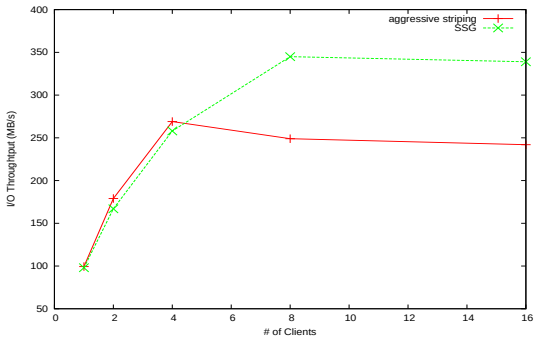

Fig. 7. The I/O throughput increases by up to 38.6% using the SSG scheme compared with default aggressive striping, while the number of clients increases.

segment size, 64KB, 128KB, 256KB, and 512KB, respectively, to access different files from $file_0$ to $file_3$ striped over storage servers. Processes start executing the application at the same time. In the first test, we bypass the SSG scheme and use default file striping (64KB, 0, 8) for the four files. In the second test, we launch the SSG threads and use file striping triplets suggested by the I/O speedup model. The striping unit size is rounded to the nearest multiple of 64KB following the Lustre system requirement. Figure 8 shows the predicted I/O speedup of candidate server groups with the specific number of I/O servers for the *mpi-io-test* benchmark. Based on the prediction results, SSG chooses 3 as file striping factor for $file_0$ and $file_3$, and 4 for the others.

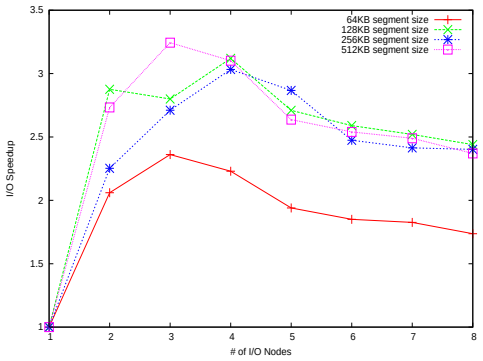

Fig. 8. The predicted I/O speedup of the candidate server groups for the mpi-io-test benchmark.

In next step, we assume that all the files have the same popularity in the beginning. SSG selects the file striping factor in the round-robin fashion. Finally, striping parameters for each file is set as follows: $file_0$(64KB, 0, 3), $file_1$(64KB, 3, 4), $file_2$(64KB, 7, 4), and $file_3$(128KB, 3, 3). Figure 9 shows the throughput observed by the benchmarks for each process with different segment sizes. The experiments show that the throughput is increased by 32.1% on average as SSG reduces the interferences on disks of each storage server makes the entire system relieved from the performance bottleneck due to the Incast effect.

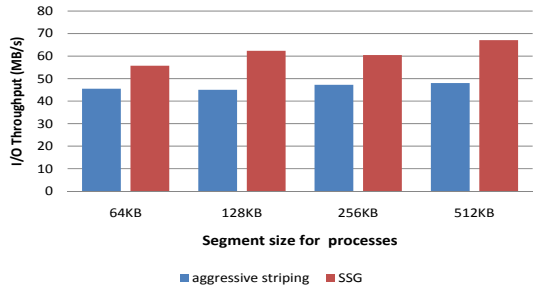

Fig. 9. Experiment results in scenario I. There are four processes involved in this test. Each process reads 1GB data with 64KB, 128KB, 256KB, and 512KB segment size, respectively.

5.6 Scenario II: Parallel I/O Experiments

This experiment shows how the SSG scheme reduces the overhead incurred by the contention on concurrent accesses of popular files. In this experiment, we also use the *mpi-io-test* benchmark, which is modified to support reading in different rates to simulate file access frequency, by controlling how many requests are produced in one second. There is a process running in background from the beginning, reading $file_0$ with 64KB segment size and default striping setting. At the beginning, two processes read $file_1$ and $file_2$ in the rates of $500req/s$ and $1req/s$ with segment size 128KB and 256KB, respectively. After 3 seconds, a newly launched process will write a new file $file_3$ with segment size 512KB. In this scenario, if without the SSG scheme, system will use default striping pattern (64KB, 0, 8) for all the files. In contrast, for SSG, the striping factors are determined by the trained I/O speedup models based on its prediction results shown in figure 8. SSG chooses (64KB, 0, 4) for $file_1$ and (64KB, 4, 4) for $file_2$. When $file_3$ is examined in admission control, the SSG scheme finds that $file_1$ is much more popular than $file_2$ based on the results from file popularity tracking. As a result, in order to reduce the interference between the new file and $file_1$, $file_3$ is striped across server4 to server6 with 128KB striping unit size.

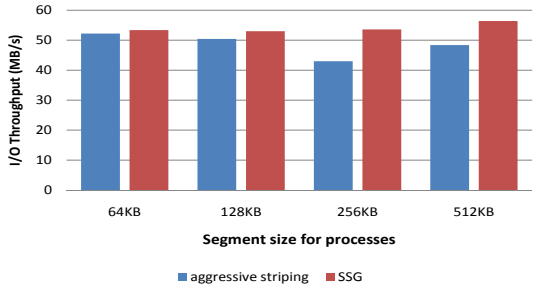

Fig. 10. Experiment results in scenario II. There are three processes involved in this test. Each process reads 1GB data with 128KB, 256KB, and 512KB segment sizes, respectively. Background process keeps reading data during the test with 64KB segment size.

Figure 10 shows the throughput of each process running the *mpi-io-test* benchmark in this experiment. The average throughput improvement is 12.2%. From this figure, we find that the process running in background benefits little from the server grouping scheme, because $file_0$ is striped over all the servers and without file redistribution SSG cannot help to reduce the interference of data accesses to this file.

6 Conclusions and Future Work

In this paper, we have shown that the conventional method for configuring the storage cluster can achieve the highest access parallelism but causes high interferences on disks of storage servers and more serious data access contention on popular files, and aggravated Incast effect. To make a trade-off between potentially high throughout due to access parallelism and excessive interference due to high parallelism, we proposed the storage server grouping (SSG) scheme over parallel file system. SSG uses admission control to carefully select the striping parameters for a file by predicting the I/O speedup of a workload and analyzing the harmoniousness of a file with the one coexisted in a group by tracking the file popularity. We have implemented a SSG prototype in the Lustre file system. Experimental results have shown that our technique is able to effectively optimize the striping of data in a cluster-based storage system. The performance evaluation on typical I/O benchmarks, such as *mpi-io-test* and Iozone, shows that SSG can improve the system throughput by up to 38.6% and 22.1% on average, demonstrating that SSG can effectively alleviate the Incast effect and reduce the interference of data accesses to popular files.

In the future, we would like to further improve adaptability and usability of the SSG scheme. For example, if SSG detects that the popularity of a file has changed substantially, it should be able to reconfigure the striping pattern for the file on line to reduce the potential interference.

7 Acknowledgments

We thank the anonymous reviewers for their constructive comments. This work was supported by US National Science Foundation under grant CAREER CCF 0845711. This work was also partially supported by the National Natural Science Foundation of China (No. 60775037 and No.60833004) and Chinese Fundamental Research Funds for the Central Universities.

References

1. A. Phanishayee, E. Krevat, and V. Vasudevan, D. Andersen, G. Ganger, G. Gibson, S. Seshan, "Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems", *Fast 2008*.
2. M. Abd-El-Malek, W. V.C.II, C.Cranor, G. Ganger, J. Hendricks, A.Klosterman, M. Mesnier, M. Prasad, B. Salmon, R. Sambasivan, S. Sinnamohideen, J. Strunk, E. Thereska, M. Wachs, J. Wylie, "Usra Minor: versatile cluster-based Storage", *Fast 2005*, San Francisco, CA, Dec. 2005.

3. Sun Microsystem, Inc. "Lustre: A Scalable, High Performance File System", http://www.lustre.org, 2009.
4. F. Schmuck and R. Haskin. "GPFS: A shared-disk file system for large computing clusters", *FAST 2002*, pages 231-244.
5. A.N. Laborarory, "Online-document", http://www.pvfs.org, 2008.
6. S. Baek and K. Park, "Prefetching with Adaptive Cache Culling for Striped Disk Arrays", *USENIX 2008*.
7. IOzone Filesystem Benchmark. http://www.iozone.org/, 2009.
8. X. Zhang and S. Jiang, "Making Resonance a Common Case: A High-performance Implementation of Collective-I/O on Paralle File System", *IPDPS 2009*.
9. D. Nagle, D. Serenyi, and A. Matthews, "The Panasas ActiveScale Storage Cluster-Delivering Scalable High Bandwidth Storage.", *SC 2004*.
10. S. Iyer and P. Druschel, "Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O.", *SOSP 2001*.
11. M. Mesnier, M. Waches, and G. Ganger, "Modeling the Relative Fitness of Storage.", *SIGMERICS 2007*.
12. M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. Ganger, "Storage device performance prediction with CART models.", *Technical Report CMU-PDL-04-103*, Carnegie Mellon University, 2004.
13. WEKA Online document, "http://weka.sourceforge.net/doc/weka/classifiers/trees/REPTree.html", 2008.
14. J. Bucy, and G.Ganger, " The Disksim simulation environment version 3.0 reference manual", *Technical Report CMU-CS-03-102*, Carnegie Mellon University, 2003.
15. C. Ruemmler and J. Wilkes,"An introduction to disk drive modeling", *IEEE Computer*, 27(3):17-28, 1994.
16. E. Shriver, A. Merchant, and J. Wilkes,"An analytical behavior model for disk drives with readahead caches and request reordering", *Preceedings of International Conference on Measurement and Modeling of Computer Systems*, pages 182-191, 1998.
17. E. Lee and R. Katz,"An analytic performance model of disk arrays", *SIGMETIRCS 2003*, pages 98-109, 2003.
18. M. Uysal, G. Alvarez, and A. Merchant,"A modular, analytical throughtput model for modern disk arrays", *Proceedings of 9th International Symmposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, Pages 183-192, 2001.
19. T. Kosar, "A new paradigm in data intensive computing: Stork and the data-aware schedulers", *Challenges of Large Applications in Distributed Environments*, 2006.
20. W. Yu, J. Vetter, R. Canon, and S. Jiang, "Exploiting Lustre File Joining for Effective Collective IO", *CCGrid 2007*.
21. E. Krevat, V. Vasudevan, A. Phanishayee, D. Andersen, G. Ganger, G. Gibson, S. Seshan, "On Aplication-level Approaches to Avoiding TCP Throughput Collapse in Cluster-based Storage Systems", *Proceedings of Petascale Data Storage Workshop, Supercomputing, 2007*.
22. D-Link Systems, Inc. "Product/Performance Specifications", http://www.dlink.com/products/resource.asp, 2008.
23. Parallel I/O Benchmarking Consortium. http://www-unix.mcs.anl.gov/pio-benchmark/, 2008.
24. J. Butler. Panasas Inc., "Personal Communication", *August 2007*.