INTERNATIONAL JOURNAL OF
NUMERICAL ANALYSIS AND MODELING
Volume 9, Number 2, Pages 289–303

ADAPTIVE TRADEOFF IN METADATA-BASED SMALL FILE OPTIMIZATIONS FOR A CLUSTER FILE SYSTEM

XIUQIAO LI, BIN DONG, LIMIN XIAO, AND LI RUAN

Abstract. Metadata-based optimizations are the common methods to improve small files performance in local file systems. However, serval problems will be introduced when applying the similar optimizations for small files in cluster file systems. In this paper, we study the tradeoffs between the performance of metadata and small files in metadata-based optimizations for a cluster file system. Our method aims to guarantee the metadata performance by adaptively migrating small files among file system nodes. We establish a theory model to analyze the small files load need to be migrated. To compute the migrated load in advance, a novel forecasting method is devised to accurately predict the one-step-ahead load of metadata and small files on a MDS. Then we propose a adaptive small file threshold model to decide the small files to be migrated. In the model, we consider the long-term and short-term tradeoffs respectively. To reduce the migration overhead, we discuss the migration tradeoffs for small files and present methods and schemes to eliminate unnecessary overheads. Finally, experiments are performed on a cluster file system and the results show the efficiency of our method in terms of promoting the load forecasting accuracy, trading off the performance of metadata and small files, and reducing migration overhead.

Key words. metadata-based small files optimization, adaptive tradeoff, load forecasting, cluster file systems

1. Introduction

Recently, the small files problem in cluster file systems has aroused wide concern [1, 2, 3] in the high performance computing area. Modern cluster file systems such as PVFS2 [4] and Lustre [5] exploit a similar client/server architecture, which divides file system nodes into three roles: client, metadata server (MDS) and I/O server (IOS). Current design of cluster file systems mainly focus on optimizing large file I/O, which improve performance by distributing files among multiple IOSs to increase parallelism. However, network overheads are introduced as clients require to connect a MDS to retrieve the file layout information before transferring file data. Compared with large file accesses, small files accesses cannot benefit from the parallel I/O due to the small amount of data. According to one study [6] on access patterns in scientific computing, small file requests account for more than 90% of total requests while only contributing to less than 10% of total I/O data. Therefore, the performance of small files becomes one of the bottlenecks for cluster file systems.

In local file systems, metadata-based optimizations [2, 7] are common techniques to reduce disk accesses and improve small file I/O performance. This type of optimizations store a small file with its file metadata. Thus the file data can be fetched in a single disk access. Cluster file systems can also apply the similar

Received by the editors December 6, 2009 and, in revised form, August 7, 2010.
2000 *Mathematics Subject Classification.* 62M10.
This research was supported by the National Natural Science Foundation of China under Grant No. 60973007, the Fundamental Research Funds for the Central Universities under Grant No. YWF-10-02-05, the Doctoral Fund of Ministry of Education of China under Grant No. 20101102110018, and the fund of the State Key Laboratory of Software Development Environment under Grant No. SKLSDE-2009ZX-01.

ideas to eliminate the bottleneck of small files. Compared with local file systems, however, serval problems will be introduced when small files are stored with their file metadata on MDSs in cluster file systems.

1) MDS overload

When large amount of small files are placed on MDSs, the small files will definitely increase server load and degrade the performance of metadata. According to the studies on file system traces, metadata requests account for up to 83 percent of the total number of I/O requests in many large scale file systems [6]. Therefore, the performance of metadata cannot be guaranteed when the small files accesses overload the MDS in file system.

2) Migration overhead

Another problem introduced by metadata-based optimizations is that small files need to be migrated to IOSs as the file size is increasing. Clients need to wait for the completion of migration before performing subsequent I/O requests. Moreover, small files can be concurrently accessed by multiple clients in many workloads, such as Web applications and scientific applications. In this case, the application performance can be significantly affected by the migration overhead. To the best of our knowledge, no substantial research is conducted on this problem at this time.

In this paper, we study the adaptive tradeoff between the performance of metadata and small files in metadata-based optimizations for a cluster file system. Our method can guarantee the metadata performance when the load of small files on MDSs are heavy. The small files are dynamically migrated among MDSs and IOSs.

First, we model the load of MDSs when enabling metadata-based optimizations for small files and analyze the theoretical tradeoffs for the cases of multiple MDSs and a single MDS in a cluster file system. Second, we present a novel forecasting method to predict the one-step-ahead load of metadata and small files on a MDS. Then we propose a adaptive small file threshold model to decide the files stored on a MDS dynamically. The model considers serval factors, such as the spare storage capacity, and the load of a MDS. Third, we present the methods of selecting small files to migrate in order to reduce the small file load to guarantee the metadata performance on the MDS. Moreover, we also propose serval methods and schemes to reduce the migration overhead.

The main feature of our method is that file migration can be performed adaptively and dynamically. Therefore, the shortcomings of metadata-based optimizations can be overcome with our method. The performance of small files can be traded off without degrading the metadata performance. We evaluate our method in a well-known cluster file system PVFS2 [4] to show the merits.

The rest of this paper is presented as follows. Section 2 describes the overview and design objectives of our method. Section 3 gives the theory model analysis of tradeoffs in metadata-based optimization for small files. Section 4 presents the details of our method. Section 5 reports the results of our method with serval experiments. Related work and Concluding remarks are provided in Sections 6 and 7, respectively.

2. Method overview and design objectives

The design of PVFS2 emphasizes on improving large file I/O while little consideration is made for the performance of small files. The file metadata in PVFS2 contains two types of attributes: common attributes, file objects related attributes and extended attributes. Common attributes contains Unix-like file attributes, such as create time, file type, and credentials. The second one includes special attributes

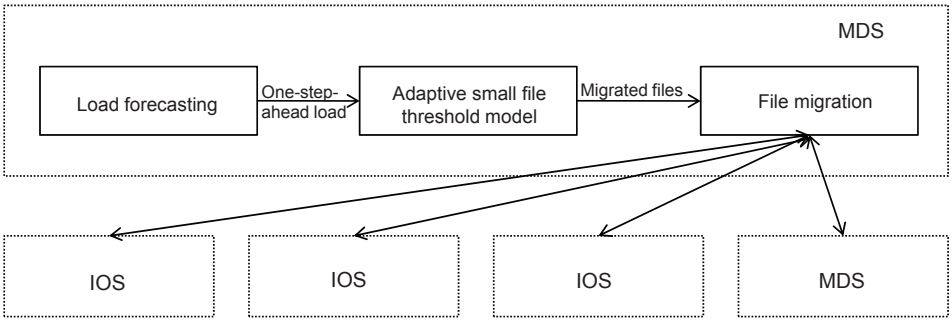


FIGURE 1. Overview of proposed method in PVFS2

related to file objects, such as data distribution and datafile handles. We store the file data as a separate datafile on the MDS located by the file metadata. Figure 1 gives the overview of our method implemented on PVFS2. First, a load forecasting module is implemented on each MDS to predict one-step-ahead load of metadata and small files, respectively. Second, a adaptive migration threshold module is responsible for deciding the files need to be migrated dynamically. Third, the MDS relies on a migration module to proceed the migration actions, such as the selection of the migration target, the selection of migration methods and schemes and response to concurrent clients.

Our method aims to guarantee the metadata performance while improving small files performance with the metadata-based optimization. The main design objectives are as follows.

1) Guarantee metadata performance

Metadata-based optimizations store the file data of a small file with its file metadata on the same MDS. Hence, the I/O performance of small files can be improved without interacting datafiles on mutiple IOSs. However, the small files accesses increase the load of MDS and degrade the metadata performance when the load of small files is heavy. Our method promotes the priority of metadata operations on MDSs and aims to guarantee the metadata performance by reducing the small files load adaptively.

2) Trade off small files performance

In order to guarantee metadata performance, our method migrates small files from a MDS to other MDSs or IOSs. The methods to trade off the small files performance are studied in cluster file systems with different amount of MDSs.

3) Reduce migration overhead

Migration of small files will introduce overhead and affect the performance of concurrent accesses. Our method intends to reduce the migration cost by eliminating unnecessary overhead.

3. Model analysis

In this section, we model the performance tradeoffs of migrating small files among file system nodes in theory. Many cluster file systems can be configured with more than one MDS to balance the metadata load. We consider the cases of both a single MDS and mutiple MDSs in a cluster file system respectively.

3.1. Case 1: Single MDS. The I/O load on a MDS are measured at a predefined interval. We define the load of metadata and small files at time T_i as $L_{meta,i}$ and

$L_{io,i}$, respectively. The maximum bandwidth of MDS_j is given as B_{logic}, which can be measured by experiments. Then, the load efficiency of MDS_j can be defined as:

$$E(MDS_j, T_i) = \frac{L_{meta,i} + L_{io,i}}{B_{logic}}. \tag{1}$$

Apparently, the total load of metadata and small files cannot exceed the maximum bandwidth:

$$B_{logic} \geq L_{meta,i} + L_{io,i}. \tag{2}$$

However, the disk on the MDS can only served for one request at the same time. When the metadata accesses or the small file accesses are heavy, the two workloads need to compete to be served by the disk. We define the load of metadata and small files on the non-compete MDS_j as $L_{io}^{'}$ and $L_{meta,i}^{'}$, respectively. In order to guarantee the metadata performance, $L_{meta,i}$ should be equal to $L_{meta,i}^{'}$. Hence, in this case, the load of small files on MDS_j need to be limited as:

$$L_{io,i} = \rho L_{io,i}^{'}. \tag{3}$$

According to Equation 2, we have the following:

$$\rho = \frac{E(MDS_j, T_i)B_{logic} - L_{meta}}{L_{io,i}^{'}}. \tag{4}$$

Let $E(MDS_j, T_i)$ is 1 and we can get the theoretical solution ρ to guarantee the metedata performance at time T_i. That is, the small files load needs to be reduced to $(1-\rho)L_{io,i}^{'}$ at least.

3.2. Case 2: Multiple MDSs. In case of multiple MDSs, a small file on an overloaded MDS can be migrated to other low load MDSs rather than IOSs to guarantee its I/O performance. However, it need to decide how much load can be migrated to other MDSs or IOSs. A greedy solution is that all MDSs are sorted by their load and the overloaded MDS migrates its load to the left MDSs with the lowest load one by one. However, as multiple MDSs may request to migrate their own load at the same time, this solution will make MDSs with low load overloaded quickly. Therefore, in this paper, only one MDS can be selected for an overloaded MDS to migrate its load. The left load that cannot be served by the selected MDS is migrated to IOSs.

As we discussed in Section 3.1, the total migrated load of MDS_j can be given as $ML(MDS_j) = (1-\rho)L_{io,i}^{'}$. Let $MDS_j^{'}$ be the selected migration target. The maximum available load can be served by $MDS_j^{'}$ can be computed as:

$$ML(MDS_j, MDS_j^{'}) = (1 - E(MDS_j, T_i))B_{logic}. \tag{5}$$

Then the left load that needs to be migrated to IOSs is given by:

$$ML(MDS_j, IOS) = ML(MDS_j) - ML(MDS_j, MDS_j^{'}); \tag{6}$$

4. Our approach

In this section, we presents the details of our approach to adaptively and dynamically migrate the load of small files.

4.1. Load forecasting. The hypothesis of the theoretical model presented in Section 3 is the load of metadata and small files is known in advance. Hence, a forecasting method needs to be adopted to predict the approximate values for the load. In time series analysis, exponential smoothing methods (ESM) are the common short-term forecasting techniques applied in many fields [9]. The methods are suitable for on-line prediction, because they are easy to be implemented and both the data storage and computing requirements are minimal. In this paper, we devise a new forecasting method based on ESM by the collaborate effort of the I/O scheduler on the MDS. We first briefly introduce one of the simplest ESM method, and then presents the details of our method.

4.1.1. Exponential smoothing method (ESM). ESM is based on the argument that the new observed value contains more information about the trend and should have larger weight than old observed values. One of the simplest ESM is the single exponential smoothing method (SESM). In SESM, the forecast value F_i at time T_i can be calculated as the following:

$$F_i = \alpha O_{i-1} + (1-\alpha)F_{i-1}, \tag{7}$$

where O_{i-1} is the observed value at time T_{i-1} and α is the smoothing parameter between 0 and 1.

If we rewritten the equation 7 by replacing the F_{i-1} recursively, we can easily find that the value of F_i is the accumulation of previous observed values with different weights. The forecast accuracy of SESM depends on the value of the smoothing parameter. Therefore, a lot of previous studies [10, 11] focused on how to adaptively specific the proper parameter during forecasting.

4.1.2. Burst-aware exponential smoothing method (BA-ESM). In the context of cluster file systems, the I/O load often exhibits the burst characteristic as the scale of concurrent clients is hundreds and even thousands. One of the example is the checkpoint applications in supercomputing systems, where every processor core create or write its own file in the same directory at the same time [12]. In order to guarantee metadata performance, the forecast method must predict the burst load as early as possible. However, the design hypothesis of ESM is that the forecast values should fluctuate with a constant range or change slowly over time. Hence, the original ESM cannot capture such wide fluctuations of burst load required by our problem.

We present a new forecasting method BA-ESM based on SESM to predict the burst load more accurately. The basic idea of our method is that the amount of the I/O requests blocked in I/O scheduler imply the information of possible burst load in future. We can use the information to adjust certain forecasting values to keep pace with the variation of burst load.

The I/O scheduler on a MDS is used to decide the service order of I/O requests from clients with specific algorithm and block requests in a queue. Hence, we can infer the burstiness of the load from the length of the queue. A request monitor functionality is added to the I/O scheduler on MDSs to record and analyze the possibility of load burstiness. The monitor collects and counts the amount of requests and their access size for both metadata and small files requests, respectively. First, we compute the total amount of requests being served for metadata and small files during interval between T_{i-2} and T_{i-1} as:

$$TD_{meta,i-1} = \sum_{k=0}^{C_{meta,i-1}} (S_{meta,i-1,k}), \tag{8}$$

$$TD_{io,i-1} = \sum_{k'=0}^{C_{io,i-1}} (S_{io,i-1,k'}), \tag{9}$$

where TD is the total amount of requested data, C is the count of blocked requests in the queue and S is the access size of a request.

Similarly, we can compute the data amount of the metadata and small files requests blocked in the queue. We refer the two values to $\widehat{TD}_{meta,i-1}$ and $\widehat{TD}_{io,i-1}$, respectively. To be noticed, the values cannot exceed the logical I/O capability provided by the MDS and should be reduced to a predefined value if it is too large.

Based on the above information, we now can infer the possibility of burstiness at time T_i by computing the following:

$$Ratio_{meta,i-1} = \frac{\widehat{TD}_{meta,i-1}}{TD_{meta,i-1}}, \tag{10}$$

$$Ratio_{io,i-1} = \frac{\widehat{TD}_{io,i-1}}{TD_{io,i-1}}, \tag{11}$$

where $Ratio_{meta,i-1}$ and $Ratio_{io,i-1}$ are the data amount ratios between future requests and served requests at time T_{i-1}. Clearly, the larger the ratios are, the larger possibilities of I/O burstiness occurs, and vice versa. Therefore, we can use the ratios to guide the correction of the forecast load. The forecast value at time T_i in our method is given by the following formula:

$$F_i = \begin{cases} (Ratio_{i-1} - 1 + \alpha)O_{i-1} + (1-\alpha)F_{i-1} & \text{if } Ratio_{i-1} < \beta \text{ or } Ratio_{i-1} > \gamma \\ \alpha O_{i-1} + (1-\alpha)F_{i-1} & \text{if } \beta < Ratio_{i-1} < \gamma \end{cases}, \tag{12}$$

where β and γ are the low and high thresholds of the ratio for recognizing the begin and end time of the burst period.

4.2. Adaptive small file threshold model. The small file threshold decides whether a file belongs to a "small" one. To trade off the performance for both metadata and small files workloads, we propose a dynamic migration threshold model to adaptively migrate certain files among MDSs and IOSs. On one hand, the model generates small files threshold dynamically for long-term tradeoff by considering the conditions of the MDS. On the other hand, for short-term tradeoff, the model adjust the threshold for certain files to migrate their load in time to guarantee the metadata performance.

4.2.1. Long-term tradeoff. Four factors are considered to decide the threshold for long-term tradeoff: (1)the spare capacity on the MDS; (2)the load of the MDS; (3)the frequency of file migration; (4)the maximum threshold. Besides, the threshold in our model is generated on a file basis. Hence, the threshold of specific file depends on two parameters, namely global threshold and fine tuning parameter. The first two factors contribute to the global threshold and the third one is considered to tune threshold for a specific file with the purpose to avoid frequent migrations. The fourth factor limits upper threshold of small files that can be stored on a MDS.

We briefly introduce the process of deciding the threshold for small files with our model. First, the MDS collects the information of spare capacity and the load dynamically at a given interval. The spare capacity P_{sg} is calculated by the proportion of available space in the total disk capacity on a MDS. Similarly, the total load P_{ld} can be computed by the proportion of current bandwidth in the

theoretical maximum bandwidth of the MDS. We use the observed load rather than forecast one for long-term tradeoff. Then the global threshold can be computed by the following equation:

$$Th_{global} = \begin{cases} (1-p)Th_{old} & \text{if } P_{sg} \geq Th_{sg} \\ max((1+q)Th_{old}, Th_{max}) & \text{if } P_{ld} < Th_{ld}^{'}, P_{sg} < Th_{sg} \\ (1-q)Th_{old} & \text{if } P_{ld} > Th_{ld}, P_{sg} < Th_{sg} \end{cases}, \tag{13}$$

where Th_{global} is the global migration threshold, p and q are adjustment parameters, Th_{sg} is the threshold of spare capacity, $Th_{ld}, Th_{ld}^{'}$ are the low and high thresholds of the load, Th_{old} is the final threshold of last round and Th_{max} is the given maximum threshold.

In our model, the factor of spare capacity is given the highest priority, because the MDS should always provide available space for metadata storage. Once the small files storage exceeds the threshold Th_{sg}, we choose to migrate the least accessed small files and decrease the small file threshold. For the factor of MDS load, the model relies on the values of Th_{ld} and $Th_{ld}^{'}$ to decide whether the small file threshold should be increased or decreased. The purpose of this adjustment is to reduce the chance of overloading a MDS.

In order to avoid unnecessary migration, we maintain a fine-tuned field in the file metadata to record the migration frequency for each small file. Thus, the final threshold of a file is chosen from the larger one between Th_{max} and the fine-tuned threshold by the following equation:

$$Th_{final} = max((1 + MF \cdot \delta) \cdot Th_{global}, Th_{max}), \tag{14}$$

where MF is the migration frequency of the file, and δ is the penalizing weight for migration frequency.

Finally, when a client requests a small file stored on a MDS, the MDS first fetches the global threshold and the migration frequency of the file and then computed the final threshold using equation 14. If the file size of the file exceeds the threshold, migration operations will be invoked to migrate the file to other IOSs. Reversely, when the MDS detecting a file stored on IOSs is truncated to a size below the threshold, the migration will be invoked to migrate the file back to the MDS. Therefore, the performance of small files can be traded off according to the system condition of the MDS.

4.2.2. Short-term tradeoff. Although the model in Section 4.2.1 can adaptively trade off small file performance by altering the threshold, the metadata performance cannot be guaranteed in case of burst I/O occurs. Both the metadata and small files workloads can introduce burst load on a MDS and the threshold model in Section 4.2.1 cannot react to the burstiness instantly.

We address the problem by migrating the files that are heavily accessed in advance according to the forecasting load. We adopt the proposed BA-ESM to forecast the value of $L_{meta,i}$ and $L_{io,i}$. Then, according to the model analysis in Section 3, the load needs to be migrated on MDS_j at time T_i can be coined as $ML(MDS_j)$. Therefore, the key to the problem is how to choose files to be migrated.

In theory, we should choose from the files that are accessed at time T_i. However, it is complex and costly to forecast load for each small file, especially in large file systems. Many studies show that the accessed files in large file systems have a skew to a small portion of files. Meanwhile, the accesses on a file often follow a determinate access pattern. In this paper, we choose to select files according to the load of files at time T_{i-1} instead. Hence, it is possible that the load delivers

by the selected files cannot meet the load requirement. In order to guarantee the metadata performance, the short-term tradeoffs can be two folds. On one hand, on each MDS, the load of accessed small files is recorded in a list and updated periodically. The MDS will choose serval files to migrate load from the list when $ML(MDS_j)$ is greater than zero. On the other hand, we monitor the I/O scheduler on the MDS and block the requests on small files when the actual load of small files exceeds $ML(MDS_j)$.

We now describe the selection methods for migrated files with different configurations of MDSs.

1)Single MDS

In this case, the selected small files are all migrated to IOSs. In order to reduce the migration overhead, the candidate files should have a small size and a heavy load. Hence, the files in the list is sorted by the migration profits(MP), which can be defined as the ratio between the file load and the file size. Then, the MDS selects migrated files from the list with the following steps.

(1) Sort the accessed files by their MP on MDS_j. The total load of selected files is recorded as TL.
(2) Query the file f_i with the largest MP and update TL with the load of f_i. If TL exceeds $ML(MDS_j)$ or the amount of selected file exceeds N, the selection is complete. Otherwise, this step is repeated.

2)Multiple MDSs

In this case, the selection of migrated files is the same with the case of a single MDS. However, we need to tradeoff the small file performance by choosing the migration targets. As we discussed in Section 3, only one MDS can be selected as the migration target. The selection rule is based on the load efficiency of a MDS. We briefly introduce the load migration as the following steps.

(1) The migrated files on MDS_j is selected by the same steps. The total load of selected files is recorded as TL.
(2) MDS_j contacts other MDSs to request their available load. Then, we greedily select the MDS that can satisfy the migration load as the migration target by $\min_{j'}(TL - ML(MDS_j, MDS_j^{'}))$. After that, the files that migrated to $MDS_j^{'}$ are chosen as the files with heavy load from the migrated files as many as possible. If no MDS can be selected or the target MDS cannot served all required load, go to Step 3.

    In order to avoid overwhelming a low load MDS, we set a threshold for the load efficiency of a MDS and the MDS can exclude from the candidates of migration target. Furthermore, a MDS can only be selected as migration target by one MDS during a time interval.
(3) The left migrated files are migrated to IOSs.

4.3. Migration issues. In this section, we presents the methods and schemes to reduce migration overhead.

4.3.1. Migration methods. Migration of small files will introduce network overhead inevitably. We presents two kinds of migration methods to reduce the overhead of transmitting data. As we discussed in Section 4.2.1, the migration process will be invoked when the file size exceeds the threshold. The total amount of data need to transfer over network can be given as:

$$TD = 2 \cdot RD + FD, \tag{15}$$

where RD is the amount of data written by the request, and FD is the file size before serving the request. It is clear that the data written by the request will be transferred two times over network. For most of small files, this extra overhead can be ignored since the access size of the requests is rather small. However, for large file accesses, the overhead will increase significantly when transferring the large amount of data duplicately.

The basic idea of our methods is that migration can be invoked by a client or a MDS. The first method, namely, active migration, is that a MDS migrates a file to IOSs when the file size exceeds current threshold. This method is suitable for the case of the migration invoked by small accesses. The second method, namely, passive migration, is that clients can take the initiative to trigger the migration on a MDS when the amount of written data exceeds current threshold. This method could avoid large amount of data being transferred more than onetime. The client will transfer the written data to IOSs in parallel after received the acknowledgement of migration completion from the MDS.

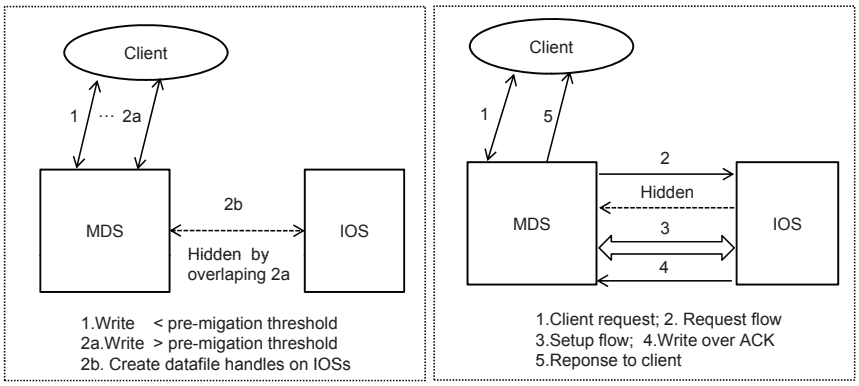
4.3.2. Latency hiding. There are two steps to migrate a file from a MDS to IOSs. First, the MDS need to create datafiles on each IOS to store file data. Second, the file data are transferred to IOSs and stored in the datafiles. However, the creation of datafiles will spend rather expensive time and the data flow of transferring data introduce large network overhead [13, 14]. In order to reduce the side effects, two latency hiding schemes are proposed to reduce the overhead.

1)Latency hiding for datafile creation

We add a pre-migration threshold parameter on MDSs. When the small file size exceeds this threshold, the MDS create datafiles on IOSs to overlap the process of file writing. By setting this threshold carefully, the overhead of datafile creation can be hidden perfectly. Figure 2a show the process.

2)Latency hiding for file migration

Clients need to wait for the completion of migration on a MDS before completion. As Figure 2b shows, the migration in cluster file systems requires an extra network round to get the file size of datafiles. However, this step can be bypassed since all datafiles are empty.



(a) Datafile creation (b) File migration

FIGURE 2. Latency hiding schemes

4.3.3. Migration response. Cluster file systems support concurrent accesses from multiple clients. It is possible that there are other clients request the files being migrated by the MDS. To guarantee the consistency, the file need to be locked by setting a special flag in metadata attributes, which definitely affects the performance. We propose three response strategies depending on the request types.

(1) Revoke file migration when the client requests involve modification or deletion of small files data in order to avoid unnecessary files migration.
(2) Notify clients to retry at a reasonable interval for read or write requests.
(3) Proceed other requests that do not request the file data.

5. Results

In this section, we report the results of our method with serval experiments. We implement the metadata-based small file optimization and our method in PVFS2 and the metadata-based optimization is used as a baseline for comparison. Our experiment platform consists of 7 Lenovo X86 servers. Each node has four Intel Xeon5160 processors with 4Gb DDR2 memory and connects with other nodes with 1Gbps Ethernet. Table 1 shows the configuration of our method used in the experiments.

TABLE 1. Parameters of adaptive threshold model

Th_{sg}	Th_{ld}	$Th_{ld}^{'}$	p	q	δ	Th_{max}
0.8	0.8	0.2	0.05	0.05	0.05	2MB

First, we introduce the simulation methods to evaluate the benefits of our method. Then, we examines the forecasting accuracy of proposed BA-ESM by comparing with the case of SESM. Third, we conduct performance experiments to show the tradeoffs between metadata and small files workloads when applying metadata-based optimizations for small files. At last, we examine the benefits of reducing migration overhead with our method.

5.1. Simulation methods. To evaluate that our method can guarantee metadata performance, a simulation architecture was designed to generate metadata workloads with various load. The basic idea is to directly post metadata requests to the I/O scheduler on a MDS rather issuing requests from clients. Therefore, we can simulate the burstiness and various metadata load in very large cluster file systems by customizing the arrival intervals among metadata requests.

Our simulation for metadata workloads consists of two phases: trace generation and replay. In the first phase, we generate the metadata trace to simulate the steady and burst metadata accesses [16]. The configuration of generating metadata workloads is controlled by the parameters represented as a tuple $< \phi, \varphi >$, where ϕ is the number of accesses, and φ is the arrival interval among the accesses. Besides, we can specify the repetitions to increase the simulation scale. In the second phase, we post the metadata requests of the generated trace to I/O scheduler with given arrival intervals. We implement a special routine to serve these requests and bypass the process of the response to clients. Each request write a 1KB metadata entry on a MDS. By relaying the traces, we can compare the results between our method and the original metadata-based optimization with the same conditions of experiments.

For small files workloads, we adopt the *IOR* benchmark [15] to access small files on a MDS. This is because pervious studies show that the benchmark can generate

burst I/O requests [6]. By configuring the amount of clients, we can simulate the small file workloads with various load.

5.2. Accuracy of load forecasting. In this section, we compare the forecasting accuracy of proposed BA-ESM with the SESM method. Table 2 shows the parameters of generating the tested metadata workload using our simulator.

TABLE 2. Simulation parameters for metadata workload

Steady accesses		Busty accesses		Repetitions
ϕ	φ	ϕ	φ	
5000	1ms	10000	0.1ms	1000

Figure 4 shows the forecasting results of the two methods with an example of the steady and burst metadata workload. The parameters of BA-ESM in equation 12 are chosen as $\beta = 0.9$ and $\gamma = 1.1$. For both two methods, we choose $\alpha = 0.1$ as the smoothing parameter.

From Figure 4, we can observe that the two methods forecast roughly the same load series. This is due to the steady workload generates load with little fluctuation. There is no need to correct the forecasting load since the value of ratio in equation 12 is always around one. However, from Figure 3, we can observe that BA-ESM can predict the trend of wide fluctuation more accurately than SESM. Furthermore, the forecasting load in SESM is apparently lagged in capturing the large load variation. The reason is that SESM can only predict the load with little fluctuation without a priori knowledge. For our method, however, we can infer the bustiness of load by monitoring the requests blocked in the queue of the I/O scheduler.

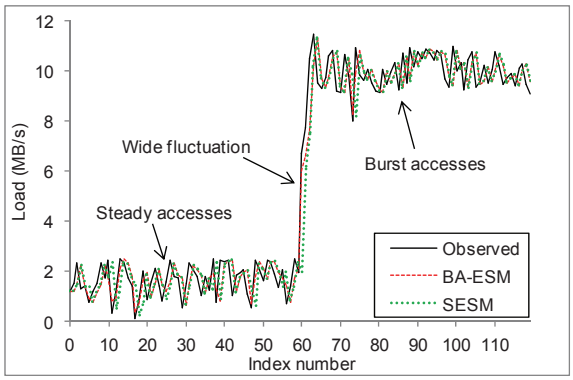


FIGURE 3. Comparison of forecasting results between BA-ESM and SESM

To quantize the comparison of forecasting accuracy, we adopts the mean absolute error(MAE) to measure the forecasting error of the two methods. MAE is computed as the following equation:

$$MAE = \frac{\sum_{i=1}^{n} |F_i - O_i|}{n}, \tag{16}$$

where n is the total amount of forecasting values. Table 3 gives the results of BA-ESM and SESM on forecasting load of tested metadata workload. We choose the smoothing parameter as 0.1 for BA-ESM. The forecasting accuracy of BA-ESM is

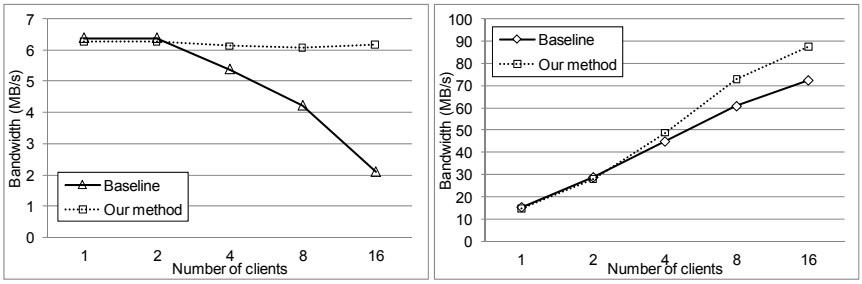
higher than all the results of SESM with different values of smoothing parameter. For metadata workload, the forecasting results is more accurate than the cases of small files workload. The reason is that the load fluctuations of *IOR* benchmark is more wide than metadata workload. As the requests of IOR are issued by remote clients, there is undetermined network latency among the requests from different clients. For metadata workload, the requests are simulated and issued by the MDS with pre-defined arrival intervals.

TABLE 3. Comparison of forecasting accuracy in terms of MAE

Method	BA-ESM	SESM		
α	0.1	0.1	0.6	0.9
Metadata workload	0.53	0.68	0.69	1.05
Small files workload	3.89	7.27	7.94	9.82

5.3. Performance results.

5.3.1. Metadata and small files performance. In this subsection, we shows the performance results of metadata and small files workloads with our method. To examine the effect of adaptive tradeoff, we configure the IOR benchmark to write small files on a MDS with different amount of clients.

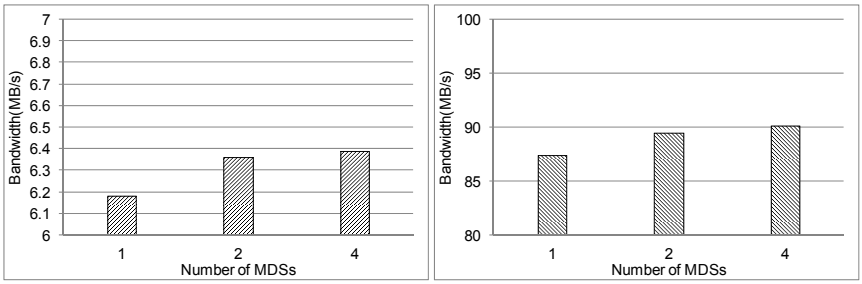


(a) Metadata performance (b) Small files perfomance

FIGURE 4. Comparison of the performance of metadata and small files workloads on a MDS

First, we examine the case of a single MDS in a file system. Figure 4 shows the comparison of metadata and small files performance with our method and the baseline optimization. The baseline metadata performance is comparable with the one of our method when only one or two clients accessing small files on the MDS. The MDS can afford the load of metadata and small files workloads and there is little interference between the two kinds of requests. However, with the scale of current clients increased, the small file accesses become intensive and affect the performance of metadata accesses. As shown in Figure 4a, the metadata performance decreases 66.7% in case of 16 processes compared with only one process accessing small files. The similar results can be seen from the performance of small files, which is shown in Figure 4b. As more requests concurrently issuing from clients, there is more chance of completing for the disk on the MDS for the metadata and small files accesses.

Compared with baseline performance, our method produce more steady metadata performance and scalable small files performance. This can be explained by two reasons. On one hand, our method can migrate the load of small files to IOSs according to the forecasting load of proposed BA-ESM method. The requests on the migrated small files can be served by IOSs rather than MDS. Hence, the interference of heavy metadata and small files accesses can be reduced. On the other hand, our method adopts a adaptive small files threshold model to adjust the amount of small files on the MDS according to the system conditions. By the adjustment, the file system can approaching better long-term tradeoffs for small files and metadata performance.



(a) Metadata performance (b) Small files perfomance

FIGURE 5. Effect of MDSs on the performance of metadata and small files workloads

Second, we examine the case of multiple MDSs in a file system. Figure 5 shows the performance of metadata and small files workloads under different amount of MDSs. Compared with the case of single MDS, the metadata performance increases about 3% with multiple MDSs. The migrated small files can be migrated to other MDSs rather than IOSs. As the metadata of these files are also migrated to other MDSs, the interference of the metadata accesses on these files can be avoided. For small files, the performance is also improved in case of multiple MDSs. The reason is that other MDSs have the higher priority to be the migration target than IOSs and the performance of small files on MDS is better than on IOSs.

5.3.2. Migration overhead. In this subsection, we reports the results of the experiments on migration overhead. We configure the *IOR* benchmark to create 1000 files on a MDS and then writing data to these files with various access size. The small file threshold is fixed to 1MB and the functionality of the adaptive model in Section 4.2.1 is turned off. The experiments are repeated when proposed migration methods and schemes are turned off to evaluate the improvements. The migration times are recorded and the average migration cost is reported in Figure 6.

For the migration invoked by small accesses, the active migration method is chosen to migrate the file data after proceeding the I/O access. The baseline migration also adopts the same method and produces identical overhead. However, for the migration invoked by large accesses, the passive migration method can bring substantial benefits by avoiding to transfer large amount of data duplicately. This can be proved by the results when the access size of the requests is 1MB or 4MB. Compared with the baseline method, the proposed latency hiding schemes can save the network overhead for the migrations invoked by both small and large accesses.

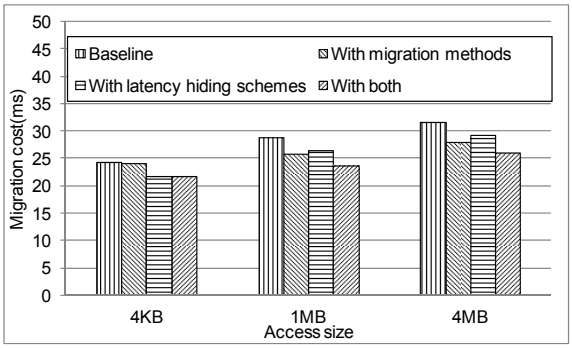


FIGURE 6. Comparison of migration cost

6. Related work

Small files optimization: Metadata-based optimizations are the natural methods to optimize small files performance. Carns et al. [2] implemented a metadata stuffing method to optimize small file performance in PVFS2. They placed the files with size smaller than the first stripe size on a MDS and do not span file data among multiple servers. Then, if a client attempts to access beyond the first stripe, the client sends a *unstuff* request to create datafiles on IOSs. However, this method can be enabled only when the MDS acts as a IOS. Hence, the metadata and data I/O are mixed on the same server and no mechanisms are adopted to guarantee the metadata performance, especially when experiencing burst metadata load. Directory hint method presented in [17] is designed for directory with large amounts of small files. The method provided a special directory hint *no_metadata* to store file metadata and data in a single datafile for files in the directory. Thus, clients can get the metadata and file size with one round message exchange. In this method, the IOS acts as a MDS only for small files at the same time. Hence, there is no impact on the metadata performance. However, the directory restriction limits its applying scope to a directory level. Thus only few applications can benefit from the method.

Compared with the above methods, our method can exploit the full performance potential of the cluster file system for both metadata and small files workloads. The small files can be adaptively migrated among file system servers according to the condition of MDSs and the burstiness of metadata load. Hence, both metadata and small files performance can be traded off.

Load forecasting: There are many short-term forecasting methods available in literature. Li et al. [11] summarized the advantages and disadvantages of several typical methods for traffic predictions. Their application requirements are similar with the load forecasting in this paper. That is, the forecasting methods should have little requirements of data storage and computing for runtime forecasting. Thus, the exponential smoothing methods are the proper methods to be used due to their simplicity of implementation. Lee et al. [10] proposed a OPHB (On-Line Parameter History Bank) mechanism to adaptively choose proper smoothing parameter for simple exponential smoothing method. The method is targeted for disk I/O load prediction and can capture the actual I/O behavior. However, their method is still based on history information and cannot predict the burst load, which is required by our problem. In contrast, our method can infer possible burst load from the I/O scheduler, which contains the information about future accesses.

7. Conclusion

This paper studied the tradeoffs in metadata-based optimization for small files in a cluster file system. We propose a novel forecasting method to predict one-step-ahead load accurately. Based on the load predictions, the performance of metadata and small files can be traded off by the proposed adaptive small files threshold model. Furthermore, the proposed migration methods and schemes can reduce the migration cost by eliminating unnecessary overhead. Experiments show our method can guarantee the metadata performance and reduce the migration overhead when applying metadata-based optimizations for small files in cluster file systems.

References

[1] Hildebrand, D., Ward, L., Honeyman, P., Large Files, Small Writes, and pNFS. Proceedings of the 20th Annual International Conference on Supercomputing. Queensland, Australia (2006) 116-124.
[2] Carns, P., Lang, S., Ross, R., Vilayannur, M., Kunkel, J., Ludwig, T., Small-file access in parallel file systems,IEEE IPDPS, Rome, Italy (2009) 1-11.
[3] Kuhn, M., Kunkel, J., Ludwig, T., Directory-based Metadata Optimizations for Small Files in PVFS2. Proceedings of the 14th international Euro-Par conference on Parallel Processing. Las Palmas de Gran Canaria, Spain (2008) 90-99.
[4] The Parallel Virtual File System,http://www.pvfs.org (2010).
[5] Lustre File System, http://www.lustre.org (2010).
[6] Wang, F., Xin, Q., Hong, B., Brandt S.A., Miller S.L., File System Workload Analysis For Large Scale Scientific Computing Applications. 12th NASA Goddard Conference on Mass Storage Systems and Technologies. USA (2004) 139-152.
[7] Gregory, R.G., Kaashoek, M.F., Embedded inodes and explicit grouping: exploiting disk bandwidth for small files, Proceedings of the Annual Technical Conference on Proceedings of the USENIX 1997 Annual Technical Conference Berkeley, CA, USA (1997) 1-11.
[8] Roselli, D., Lorch, Anderson, T.E., A Comparison of File System Workloads. Proceedings of the 2000 USENIX Annual Technical Conferencec, Berkeley, CA, USA (2000) 41-54.
[9] Box, G., Jenkins, G. M., Reinsel, G., Time Series Analysis: Forecasting and Control,Prentice Hall; 3rd edition (1994).
[10] Lee, D.W., Rudrapatna, S. R., Improving Disk I/O Load Prediction Using Statistical Parameter History in Online for Grid Computing, IEICE Transactions - IEICE (2006) 89(9):2484-2490.
[11] Li, Z.P., Yu, H., Liu, Y.C., Liu, F.Q., An Improved Adaptive Exponential Smoothing (IAES) model for Short-term Travel Time Forecasting of Urban Arterial Street, ACTA AUTOMATICA SINICA (2007).
[12] Patil, S.V., Gibson, G.A., Lang, S., Polte, M., GIGA+: scalable directories for shared filesystems, Preoceedings of PDSW 07', NewYork, NY, USA (2007) 26-29.
[13] Devulapalli, A., Wyckoff, P., File Creation Strategies in a Distributed Metadata file System. Parallel and Distributed Processing Symposium. USA (2007) 1-10.
[14] Sebepoul, Z., Magoutis, K., Marazakis, M., Bilas, A., A Comparative Experimental Study of Parallel File Systems for Large-Scale Data Processing. First USENIX Workshop on Large-Scale Computing, Boston, MA, USA (2008).
[15] IOR2 benchmark, http://ior-sio.sourceforge.net (2008).
[16] Wittawat, T., Swapnil, P., Garth, G., Data-intensive file systems for Internet services: A rose by any other name ..., Tech. Rep. CMU-PDL-08-114, Parallel Data Lab, Carnegie Mellon University (2008).
[17] Kuhn, M., Kunkel, J.M., Ludwig, T., Dynamic file system semantics to enable metadata optimizations in PVFS, Concurr. Comput. : Pract. Exper. (2009) 21(14):1775-1788.

State Key Laboratory of Software Development Environment, Beihang University, Beijing 100191, China

School of Computer Science and Engineering, Beihang University, Beijing 100191, China
E-mail: xiuqiaoli@cse.buaa.edu.cn and bdong@cse.buaa.edu.cn and xiaolm@buaa.edu.cn and ruanli@buaa.edu.cn