Performing Initiative Data Prefetching in Distributed File Systems for Cloud Computing

Jianwei Liao, Francois Trahay, Guoqiang Xiao, Li Li, and Yutaka Ishikawa, *Member, IEEE*

Abstract—This paper presents an initiative data prefetching scheme on the storage servers in distributed file systems for cloud computing. In this prefetching technique, the client machines are not substantially involved in the process of data prefetching, but the storage servers can directly prefetch the data after analyzing the history of disk I/O access events, and then send the prefetched data to the relevant client machines proactively. To put this technique to work, the information about client nodes is piggybacked onto the real client I/O requests, and then forwarded to the relevant storage server. Next, two prediction algorithms have been proposed to forecast future block access operations for directing what data should be fetched on storage servers in advance. Finally, the prefetched data can be pushed to the relevant client machine from the storage server. Through a series of evaluation experiments with a collection of application benchmarks, we have demonstrated that our presented initiative prefetching technique can benefit distributed file systems for cloud environments to achieve better I/O performance. In particular, configuration-limited client machines in the cloud are not responsible for predicting I/O access operations, which can definitely contribute to preferable system performance on them.

Index Terms—Mobile cloud computing, distributed file systems, time series, server-side prediction, initiative data prefetching

1 Introduction

The assimilation of distributed computing for search engines, multimedia websites, and data-intensive applications has brought about the generation of data at unprecedented speed. For instance, the amount of data created, replicated, and consumed in United States may double every three years through the end of this decade, according to the EMC-IDC Digital Universe 2020 study [1]. In general, the file system deployed in a distributed computing environment is called a distributed file system, which is always used to be a backend storage system to provide I/O services for various sorts of data-intensive applications in cloud computing environments. In fact, the distributed file system employs multiple distributed I/O devices by striping file data across the I/O nodes, and uses high aggregate bandwidth to meet the growing I/O requirements of distributed and parallel scientific applications [2], [10], [21], [24].

However, because distributed file systems scale both numerically and geographically, the network delay is becoming the dominant factor in remote file system access [26], [34]. With regard to this issue, numerous data prefetching mechanisms have been proposed to hide the latency in distributed file systems caused by network communication and disk operations. In these conventional prefetching mechanisms, the client file system (which is a part of the file system and runs on the client machine) is supposed to predict future access by analyzing the history of occurred I/O access without any application intervention. After that, the client file system may send relevant I/O requests to storage servers for reading the relevant data in advance [22], [24], [34]. Consequently, the applications that have intensive read workloads can automatically yield not only better use of available bandwidth, but also less file operations via batched I/O requests through prefetching [29], [30].

On the other hand, mobile devices generally have limited processing power, battery life and storage, but cloud computing offers an illusion of infinite computing resources. For combining the mobile devices and cloud computing to create a new infrastructure, the mobile cloud computing research field emerged [45]. Namely, mobile cloud computing provides mobile applications with data storage and processing services in clouds, obviating the requirement to equip a powerful hardware configuration, because all resource-intensive computing can be completed in the cloud [46]. Thus, conventional prefetching schemes are not the best-suited optimization strategies for distributed file systems to boost I/O performance in mobile clouds, since these schemes require the client file systems running on client machines to proactively issue prefetching requests after analyzing the occurred access events recorded by themselves, which must place negative effects to the client nodes.

Furthermore, considering only disk I/O events can reveal the disk tracks that can offer critical information to perform I/O optimization tactics [41], certain prefetching techniques have been proposed in succession to read the data on the disk in advance after analyzing disk I/O traces [25], [28]. But, this kind of prefetching only works for local file systems, and

- *J. Liao is with the College of Computer and Information Science, Southwest University of China, Chongqing, China 400715. E-mail:liaojianwei@il.is.s.u-tokyo.ac.jp.*
- *F. Trahay is with Telecom SudParis, France.*
- *G. Xiao is with the Southwest University of China.*
- *L. Li is with the Southwest University of China.*
- *Y. Ishikawa is with the University of Tokyo, Japan.*

Manuscript received 19 Nov. 2014; revised 16 Mar. 2015; accepted 22 Mar. 2015. Date of publication 27 Mar. 2015; date of current version 6 Sept. 2017. Recommended for acceptance by Y. Wu.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCC.2015.2417560

the prefetched data is cached on the local machine to fulfill the application's I/O requests passively. In brief, although block access history reveals the behavior of disk tracks, there are no prefetching schemes on storage servers in a distributed file system for yielding better system performance. And the reason for this situation is because of the difficulties in modeling the block access history to generate block access patterns and deciding the destination client machine for driving the prefetched data from storage servers.

To yield attractive I/O performance in the distributed file system deployed in a mobile cloud environment or a cloud environment that has many resource-limited client machines, this paper presents an initiative data prefetching mechanism. The proposed mechanism first analyzes disk I/O tracks to predict the future disk I/O access so that the storage servers can fetch data in advance, and then forward the prefetched data to relevant client file systems for future potential usages. In short, this paper makes the following two contributions:

1) *Chaotic time series prediction and linear regression prediction to forecast disk I/O access.* We have modeled the disk I/O access operations, and classified them into two kinds of access patterns, i.e., the random access pattern and the sequential access pattern. Therefore, in order to predict the future I/O access that belongs to the different access patterns as accurately as possible (note that the future I/O access indicates what data will be requested in the near future), two prediction algorithms including the chaotic time series prediction algorithm and the linear regression prediction algorithm have been proposed respectively.
2) *Initiative data prefetching on storage servers*. Without any intervention from client file systems except for piggybacking their information onto relevant I/O requests to the storage servers. The storage servers are supposed to log disk I/O access and classify access patterns after modeling disk I/O events. Next, by properly using two proposed prediction algorithms, the storage servers can predict the future disk I/O access to guide prefetching data. Finally, the storage servers proactively forward the prefetched data to the relevant client file systems for satisfying future application's requests.

The remaining of the paper is organized as follows: The background knowledge and related work regarding I/O access prediction and data prefetching will be described in Section 2. The design and implementation details of this newly proposed mechanism are illustrated in Section 3. Section 4 introduces the evaluation methodology and discusses experimental results. At last, we make concluding remarks in Section 5.

2 Related Work

For the purpose of improving I/O system performance for distributed/parallel file systems, many sophisticated techniques of data prefetching have been proposed and developed consecutively. These techniques indeed contribute to reduce the overhead brought about by network communication or disk operations when accessing data, and they are generally implemented in either the I/O library layer on client file systems or the file server layer.

As a matter of fact, the data prefetching approach has been proven to be an effective approach to hide latency resulted by network communication or disk operations. Padmanabhan and Mogul [19] proposed a technique to reduce the latency perceived by users through predicting and prefetching files that are likely to be requested shortly in the World Wide Web (WWW) servers. There are also several data prefetching tactics for distributed/parallel file systems. For instance, *informed prefetching* that leverages hints from the application to determine what data to be fetched beforehand, since it assumes that better file system performance can be yielded with information from the running application [20], [27], [30], [31], [33]. However, this kind of prefetching mechanisms cannot make accurate decisions when there are no appropriate hints (I/O access patterns) from the running applications, but inaccurate predictions may result in negative effects on system performance [22], [23]. Moreover, the client file systems are obliged to trace logical I/O event and conduct I/O access prediction, which must place overhead on client nodes.

Prefetching in shipped distributed file systems. Regarding the prefetching schemes used in real world distributed file systems, the Ceph file system [32] is able to prefetch and cache files' metadata to reduce the communication between clients and the metadata server for better system performance. Although the Google file system [3] does not support predictive prefetching, it handles I/O reads with quite large block size to prefetch some data that might be used by the following read requests, and then achieves better I/O data throughput. The technique of read cache prefetching has been employed by the Lustre file system using *Dell PowerVault MD Storage* for processing sequential I/O patterns [37].

Prefetching on file servers. Li and Zhou first investigated the block correlation in the storage servers by employing data mining techniques, to benefit I/O optimization on file servers [35], [39]. Narayan and Chandy [41] researched disk I/O traffics under different workloads, as well as different file systems, and they declared the modeling information about physical I/O operations can contribute to I/O optimization tactics for better system performance [42]. In [43], an automatic locality-improving storage has been presented, which automatically reorganizes selected disk blocks based on the dynamic reference stream to effectively boost storage performance. After that, *DiskSeen* has been presented to support prefetching block data directly at the level of disk layout [25], [28]. Song et al. [44] have proposed a server-side I/O collection mechanism to coordinate file servers for serving one application at a time to decrease the completion time. He et al. [29] have explored and classified patterns of I/O within applications, thereby allowing powerful I/O optimization strategies including pattern-aware prefetching. Besides, Bhadkamkar et al. [38] have proposed BORG, which is a block reorganisation technique, and intends to reorganize hot data blocks sequentially for smaller disk I/O time, which can also contribute to prefetching on the file server. However, the prefetching schemes used by local file systems aim to reduce disk latency, they fail to hide the latency caused by network communication, as the prefetched data is still buffered on the local storage server side.

Distributed file systems for mobile clouds. Moreover, many studies about the storage systems for cloud environments that enable mobile client devices have been published. A new mobile distributed file system called *mobiDFS* has been proposed and implemented in [6], which aims to reduce computing in mobile devices by transferring computing requirements to servers. *Hyrax*, which is a infrastructure derived from Hadoop to support cloud computing on mobile devices [7]. But Hadoop is designed for general distributed computing, and the client machines are assumed to be traditional computers.

In short, neither of related work targets at the clouds that have certain resource-limited client machines, for yielding attractive performance enhancements. Namely, there are no prefetching schemes for distributed file systems deployed in the clouds, which offer computing and storage services for mobile client machines.

3 Architecture and Implementation

This paper intends to propose a novel prefetching scheme for distributed file systems in cloud computing environments to yield better I/O performance. In this section, we first introduce the assumed application contexts to use the proposed prefetching mechanism; then the architecture and related prediction algorithms of the prefetching mechanism are discussed specifically; finally, we briefly present the implementation details of the file system used in evaluation experiments, which enables the proposed prefetching scheme.

3.1 Assumptions in Application Contexts

This newly presented prefetching mechanism cannot work well for all workloads in the real world, and its target application contexts must meet two assumptions:

- *Assumption 1 (resource-limited client machines).* This newly proposed prefetching mechanism can be used primarily for the clouds that have many resource-limited client machines, not for generic cloud environments. This is a reasonable assumption given that mobile cloud computing, which employs powerful cloud infrastructures to offer computing and storage services on demand, for alleviating resource utilization in mobile devices [46].
- *Assumption 2 (On-Line Transaction Processing (OLTP) applications).* It is true that all prefetching schemes in distributed file systems make sense for a limited number of read-intensive applications such as database-related OLTP and server-like applications. That is because these long-time running applications may have a limited number of access patterns, and the patterns may occur repetitively during the lifetime of execution, which can definitely contribute to boosting the effectiveness of prefetching.

3.2 Piggybacking Client Information

Most of the I/O tracing approaches proposed by other researchers focus on the logical I/O access events occurred on the client file systems, which might be useful for affirming application's I/O access patterns [16], [30]. Nevertheless, without relevant information about physical I/O access, it is difficult to build the connection between the applications and the distributed file system for improving the I/O performance to a great extend. In this newly presented initiative prefetching approach, the data is prefetched by storage servers after analyzing disk I/O traces, and the data is then proactively pushed to the relevant client file system for satisfying potential application's requests. Thus, for the storage servers, it is necessary to understand the information about client file systems and applications. To this end, we leverage a piggybacking mechanism, which is illustrated in Fig. 1, to transfer related information from the client node to storage servers for contributing to modeling disk I/O access patterns and forwarding the prefetched data.

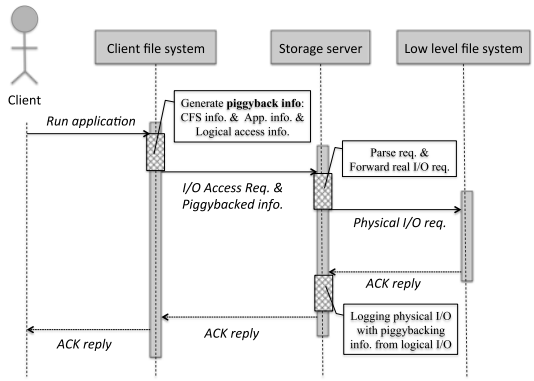

Fig. 1. Work flow among client who runs the application, client file system, storage server and low level file system: the client file system is responsible for generating extra information about the application, client file system (CFS) and the logical access attributes; after that, it piggybacks the extra information onto relevant I/O request, and sends them to the corresponding storage server. On the other hand, the storage server is supposed to parse the request to separate piggybacked information and the real I/O request. Apart from forwarding the I/O request to the low level file system, the storage server records the disk I/O access with the information about the corresponding logical I/O access.

As clearly described in Fig. 1, when sending a logical I/O request to the storage server, the client file system piggybacks information about the client file systems and the application. In this way, the storage servers are able to record disk I/O events with associated client information, which plays a critical role for classifying access patterns and determining the destination client file system for the prefetched data. On the other side, the client information is piggybacked to the storage servers, so that the storage servers are possible to record the disk I/O operations accompanying with the information about relevant logical I/O events. Fig. 2 demonstrates the structure of each piece of logged information stored on the relevant storage server. The information about logical access includes *inode* information, *file descriptor*, *offset* and *requested size*. And the information about the relevant physical access contains *storage server ID*, *stripe ID*, *block ID* and *requested size*.

3.3 I/O Access Prediction

Many heuristic algorithms have been proposed to shepherd distributing file data on disk storage, as a result, data stripes that are expected to be used together will be located close to one another [17], [18]. Moreover, Oly and Reed discovered that the spatial patterns of I/O requests in scientific codes could be represented with Markov models, so that future access can be also predicted by Markov models with proper

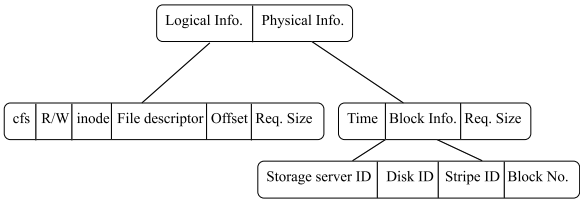

Fig. 2. Logged information about both logical access and its corresponding physical access.

state definitions [36]. Tran and Reed have presented an automatic time series modeling and prediction framework to direct prefetching adaptively, this work employs ARIMA (Autoregressive Integrated Moving Average model) time series models to forecast the future I/O access by resorting to temporal patterns of I/O requests [27]. However, none of the mentioned techniques aims to guide predicting physical access via analyzing disk I/O traces, since two neighbor I/O operations in the raw disk trace might not have any dependency. This section will illustrate the specifications about how to predict disk I/O operations, including modeling disk I/Os and two prediction algorithms adopted by our newly presented initiative prefetching mechanism.

3.3.1 Modeling Disk I/O Access Patterns

Modeling and classifying logical I/O access patterns are beneficial to perform I/O optimization strategies including data prefetching on the client nodes [29], [36]. But, it is different from logical I/O access patterns, disk I/O access patterns may change without any regularities, because the disk I/O access does not have information about applications. As mentioned before, Li et al. have confirmed that block correlations are common semantic patterns in storage systems, and the information about these correlations can be employed for enhancing the effectiveness of storage caching, prefetching and disk scheduling [39]. This finding inspired us to explore the regularities in the disk I/O access history for hinting data prefetching; currently, we can simply model disk I/O access patterns as two types, i.e., the sequential access pattern and the random access pattern.

For modeling disk I/O access, i.e., determining an I/O operation belong to a sequential stream or a random stream, we propose a working set-like algorithm to rudimentarily classify I/O access patterns into either a sequence tendency or a random tendency. Actually, Ruemmler and Wilkes [40] verified that the working set size on the selected systems is small compared to the total storage size, though the size exhibits considerable variability. This working set algorithm keeps a working set *(W(t, T))*, which is defined for time *t* as the set of access addresses referenced within the process time interval *(t-T, t)*. The working set size *w(t, T)* is the offset difference in *W(t, T)*, but the noise operations will be ignored while computing the working set size. After that, the algorithm will compare the working set size with a defined threshold to indicate whether an access pattern change occurred or not. As a matter of fact, the employed working set-like algorithm has very similar effects as the techniques that utilize Hidden Markov Models (HMM) to classify access patterns after analyzing client I/O logs [36], but it does not introduce any overhead resulted by maintaining extra data structures and performing complex algorithms.

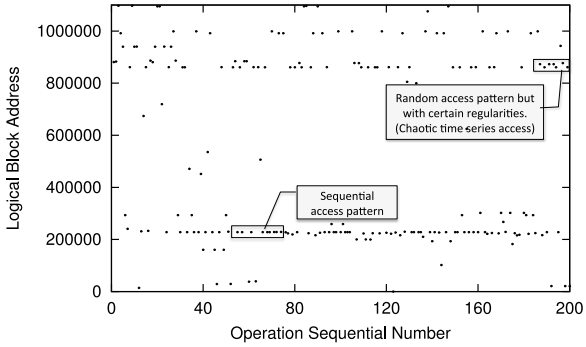

Fig. 3. Specified Disk I/O Traces with (randomly) selected 200 I/O Operations while running a database-related OLTP benchmark, i.e., *Sysbench* [13]: all of them are read operations, most of the operations indicate certain access regularities. To be specific, there are two kinds of trends in the figure, including a sequential tendency and a random tendency, but parts of the random series may be chaotic.

Fig. 3 illustrates that the disk access patterns of an OLTP application benchmark, i.e., *Sysbench* can be classified into either a sequential access tendency or a random access tendency by using our presented working set-like algorithm. After modeling I/O access patterns, we can understand that block I/Os may have certain regularities, i.e., a linear tendency and a chaotic tendency. Therefore, we have proposed two prediction algorithms to forecast the future I/Os when the I/O history follows either of two tendencies after pattern classification by using the working set-like approach.

3.3.2 Linear Regression Prediction

This linear regression prediction can be employed to forecast the future access events only if the sequence of current access events follows a linear access pattern. That is to say, confirming the current access series is sequential or not is critical to prediction accuracy. To this end, we can assume that a new access request comes at time T_{cur}, starting at address $Addr_{cur}$ and with a request size $Size_{cur}$, which can be labeled as $Access(Addr_{cur}, Size_{cur}, T_{cur})$. And we expect to forecast the information about the following read request $Access(Addr_{new}, Size_{new}, T_{new})$. From the available block access history, the proposed prediction algorithm attempts to find all block access events that occurred recently after a given time point. Namely, all $Access(Addr_{prev}, Size_{prev}, T_{prev})$, while T_{prev} is greater than (T_{cur}-T_{def}) are supposed to be collected and then put into a set, i.e., $Access\_set$. Note that the mentioned variable, i.e., T_{def}, which is a pre-defined threshold on the basis of the density of I/O operations, it can limit the time range for the previous access operations that will be used for judging whether the current access sequence is a sequential one or not.

Fig. 4 demonstrates the work flow of the proposed linear prediction algorithm. We first count the total number of occurred access events in $Access\_set$ when their target block addresses are within the range of [$Addr_{cur}$-$Addr_{def}$, $Addr_{cur}$]. It is similar to T_{def}, $Addr_{def}$ is another pre-defined threshold to confine the offset range of occurred access events. After that, if the total number of found access events is greater than our pre-defined threshold value, we can ascertain that this access request is a part of a linear access

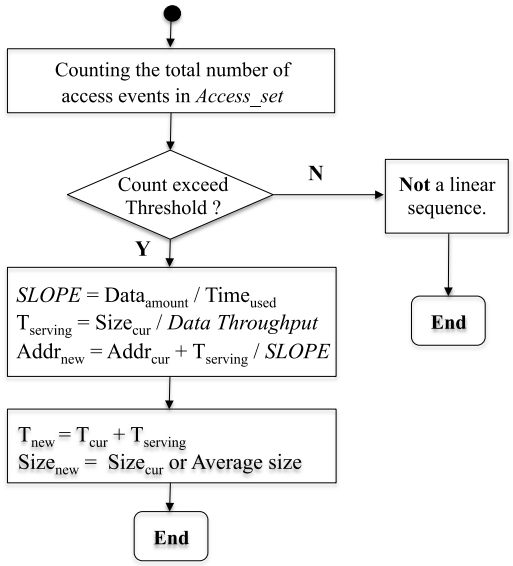

Fig. 4. The flow chart of linear regression prediction algorithm: which intends to forecast future access event when the current access events follow a linear access pattern.

sequence. As a result, the slope rate of this sequential access pattern, i.e.,*SLOPE* can be calculated to the rate of $(Data_{amount}/\ Time_{used})$, where $Data_{amount}$ is the total size of processed data during the period of this linear access sequence, and $Time_{used}$ is the time consumed for completing all relevant access requests during this period. Furthermore, $T_{serving}$ is the time required for processing the currently received read request, which indicates that the next read request can be handled only if this received one has been fulfilled. In this case, we can assert that the following future access request is probably a part of this linear access sequence, as long as the current access series follows a sequential tendency. Therefore, the future request can be predicted by using linear regression prediction, and the details about how to forecast the address, request size and time of the future access request have been shown in the last two steps of the flow chart.

3.3.3 Chaotic Time Series Prediction

Although certain disk I/Os might be a part of a sequential sequence and the algorithm of linear regression prediction can contribute to forecasting future I/O access when the occurred I/Os follow a sequential access pattern, random I/Os make up a sizable portion of the track of all I/Os. For the sake of predicting future I/O access while it follows a random trend, many prediction algorithms have been proposed, such as the Markov model prediction algorithm [36].

It is obvious that all I/O operations in a random access tendency are in chronological order; besides, we found that the neighbor I/O operations might not have any dependency, as the relationship between I/O operations is influenced by several factors including network traffic and application behaviors. Therefore, we employ an exponent called Lyapunov characteristic exponent, which characterizes the rate of separation of infinitesimally close trajectories (offsets of block I/O operations) to affirm whether the random access tendency is a chaotic time series or not. In fact, two offsets belonging to neighbor I/O operations in a phase space with an initial separation $\delta\mathbf{Z}_0$ diverge, and then after t iterations, the $\delta\mathbf{Z}(t)$ can be defined by Equation (1).

$$|\delta\mathbf{Z}(t)| \approx e^{\lambda t}|\delta\mathbf{Z}_0|, \tag{1}$$

where λ is the Lyapunov exponent.

The largest Lyapunov exponent is defined as the Maximal Lyapunov exponent (MLE), which can be employed to determine a notion of predictability for a dynamical system or a sequence [14]. A positive value of MLE is usually leveraged as an indication that the system or the sequence is chaotic, and then we can utilize chaotic time series prediction algorithms to forecast future disk I/O access. Otherwise, I/O access prediction and corresponding data prefetching should not be done while the access pattern is random but not chaotic. In this paper, we propose a chaotic time series prediction algorithm to forecast future I/O access. There are two steps in this algorithm: the first step is to calculate the Lyapunov exponent, which is a norm to indicate a chaotic series; and the second step is to predict the future I/O access by employing the obtained Lyapunov exponent.

Step 1: Lyapunov exponent estimation. Because the offsets of disk I/O operations are critical to guide data prefetching, we only take the offsets of I/O operations in the access history into account while designing the prediction algorithm. To be specific, in our prediction algorithm, each I/O operation can be treated as a point in the time series, and its value is the operation's offset.

(1) We employ offsets of disk I/O operations to build an m-dimensional phase space R^m:

$$X_i = [x_i, x_{i+k}, x_{i+2k}, \ldots, x_{i+(m-1)k}]^T \tag{2}$$

where $i = 1, 2, \ldots, M$, and $X_i \in R^m$. And k is a delay exponential, ($\tau = k\tau_s$, τ represents the delay time, and τ_s is the time interval for sampling points in the offset series); N is the total number of all points in the series, and $M = N-(m-1)k$ is the total number of phase points in R^m.

(2) To find all points x_j that are nearby point x_i while the neighbourhood is ϵ, but excluding time correlation points in the sequence,

$$\begin{cases} Dist(X_j, X_i; 0) = \left\|X_j - X_i\right\| \ \leq \epsilon \\ |j - i| \geq p, \end{cases} \tag{3}$$

where $Dist(X_j, X_i; 0)$ is the distance between X_j and X_i at starting time, and $\|*\|\$$ is the absolute Euclidean distance. p is the average time for each cycle in the time series. All points satisfying Equation (3) make up the assembly of u_j.

(3) After the time interval of $t + \tau_s$, X_i comes to X_{i+1}. As a result, the nearby point of X_i changes to X_{j+1} from X_j, and the distance can be calculated with the following equation:

$$Dist(X_j, X_i; t) = \left\|X_{j+1} - X_{i+1}\right\| \ (t = 1, 2, \ldots,). \tag{4}$$

According to the definition of Maximal Lyapunov exponent, we can achieve the following equation:

$$Dist(X_j, X_i; t) = Dist(X_j, X_i; 0)e^{\lambda t\tau} \tag{5}$$

Then, after taking a logarithm operation on both sides of Equation (5), we can receive:

$$\frac{1}{\tau_s}\ln Dist(X_j, X_i; t) = \frac{1}{\tau_s}\ln Dist(X_j, X_i; 0) + \lambda t. \quad (6)$$

λ is the slope of $\frac{1}{\tau_s}\ln Dist(X_j, X_i; t) \sim t$, we have done the relevant optimization in Equation (7) to reduce the error range of the slope,

$$S(t) = \frac{1}{M}\sum_{i=1}^{M}\ln\left(\frac{1}{|u_j|}\sum_{X_j \in u_j} Dist(X_j, X_i; t)\right), \quad (7)$$

where M is a sum number of phase points, $|u_j|$ is a total number of elements in Assembly u_j.

(4) Draw a curve of $\frac{1}{\tau_s}S(t) \sim t$, and we know the slope of this curve is λ. Actually, the value of λ can indicate whether the access series presents a chaotic trend or not. While the series is not a chaotic series, the predication will not be performed, and the relevant data prefetching is not triggered, as well.

Setp 2: Lyapunov exponent-based prediction. Once the maximum Lyapunov exponent, i.e., λ is obtained, we can estimate the series is chaotic or not. If the series is chaotic, it is possible to predict the offsets of future I/O operations. We set X_M is the center for each prediction process, X_k is the nearest point to X_M, and the distance between two points is $d_M(0)$:

$$d_M(0) = \min_{\hat{j}} \|X_M - X_{\hat{j}}\| = \|X_M - X_k\| \quad (8)$$

$$\|X_{M+1} - X_{k+1}\| = \|X_M - X_k\|\ e^{\lambda} \quad (9)$$

$$\sqrt[2]{\sum_{j=1}^{m}|x_{M+1}(j) - x_{k+1}(j)|^2} = \sqrt[2]{\sum_{j=1}^{m}|x_M(j) - x_k(j)|^2 e^{\lambda}}. \quad (10)$$

As for the phase point X_{M+1}, only its last component is unknown, but this component is predictable and can be predicted with a given λ by using Equation (10).

3.4 Initiative Data Prefetching

The scheme of initiative data prefetching is a novel idea presented in this paper, and the architecture of this scheme is demonstrated in Fig. 5 while it handles read requests (the assumed synopsis of a read operation is *read(int fildes, size_t size, off_t off)*). In the figure, the storage server can predict the future read operation by analyzing the history of disk I/Os, so that it can directly issue a physical read request to fetch data in advance.

The most attractive idea in the figure is that the prefetched data will be forwarded to the relevant client file system proactively, but the client file system is not involved in both prediction and prefetching procedures. Finally, the client file system can respond to the hit read request sent by the application with the buffered data. As a result, the read latency on the application side can be reduced significantly. Furthermore, the client machine, which might have limited computing power and energy supply, can focus on its own work rather than predicting-related tasks.

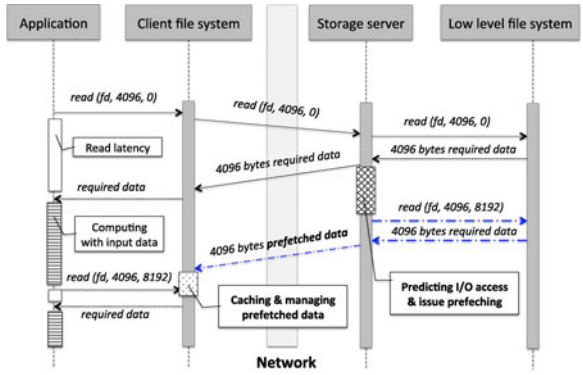

Fig. 5. The algorithm of initiative data prefetching: the first read request is fulfilled as a normal case in the distributed file system, so that the application on the client machine can perform computing tasks after receiving the required data. At the same time, the storage server is able to predict the future I/O access by analyzing the history of disk I/O access, therefore, it issues a physical read request to fetch the data (that will be accessed by the predicted read I/O access) in advance. At last, the prefetched data will be pushed to relevant client file systems proactively. While the prediction of I/O access made a hit, the buffered data on the client file systems can be responded to the application directly, that is why the read latency can be reduced.

3.5 Implementation

We have applied the newly proposed initiative prefetching scheme in the PARTE file system [15], [47], which is a C-implemented prototype of a distributed file system. Fig. 6 shows the architecture of the PARTE file system used in our evaluation experiments, it has three components and all components work at user level:

- PARTE client file system (*partecfs*), which is designed based on Filesystem in Userspace (FUSE), and aims to provide POSIX interfaces for the applications to use the file system. Normally, *partecfs* sends metadata requests or file I/O requests to the metadata server or storage servers separately to obtain the desired metadata or file data. Moreover, *partecfs* is responsible for collecting the information about the client file system and applications, as well as piggybacking it onto real I/O requests.
- PARTE metadata server (*partemds*), which manages metadata of all objects including files and directories in the file system. Besides, it monitors all storage servers and issues commands to create or remove a stripe on them. Note that the initiative prefetching is transparent to *partemds*, as it works like the metadata server in a traditional distributed file system.
- PARTE storage server (*parteost*), which handles file management (but the basic unit is a stripe rather than a file) and provides the actual file I/O service for client file systems. Furthermore, *parteost* performs initiative prefetching, that is to say, it records the block access events with the piggybacked information sent by the client file systems, then conducts pattern classification and access prediction to guide prefetching block data.

4 Evaluation and Experiments

We first introduce the experimental platform, the used comparison counterparts and the selected benchmarks in

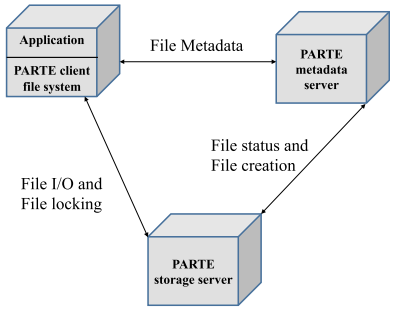

Fig. 6. Architecture of the PARTE file system:in which the initiative prefetching mechanism has been implemented.

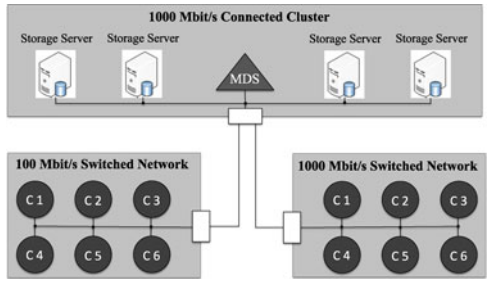

Fig. 7. The topology of our experimental platform.

evaluation experiments. Next, evaluation methodologies and relevant experimental results are discussed in details. Then, the analysis on prediction accuracy of two proposed prediction algorithms is presented. At last, we conduct a case study with real block traces.

4.1 Experimental Setup

4.1.1 Experimental Platform

One cluster and two LANs are used for conducting the experiments, the active metadata server and four storage servers are deployed on the five nodes of the cluster, and all client file systems are located on the 12 nodes of the LANs. Table 1 shows the specification of nodes on them.

Moreover, for emulating a distributed computing environment, six client file systems are installed on the nodes of the LAN that is connected with the cluster by a 1 GigE Ethernet; another six client file systems are installed on the nodes of the LAN, which is connected with the cluster by a 100 M Ethernet, and both LANs are equipped with MPICH2-1.4.1. Fig. 7 demonstrates the topology of our experimental platform clearly.

4.1.2 Evaluation Counterparts

To illustrate the effectiveness of the initiative prefetching scheme, we have also employed a non-prefetching scheme and another two prefetching schemes as comparison counterparts in our experiments:

- *Non-prefetching scheme (Non-prefetch)*, which means that no prefetching scheme is enabled in the distributed file system. That is to say, although there are no benefits that can be yielded from prefetching, there is also no overhead caused by trace logging and access prediction required for prefetching data.
- *Readahead prefetching scheme (Readahead)*, which may prefetch the data in next neighbor blocks on the storage servers, and it can adjust the amount of the prefetched data based on the amount of data already requested by the application. This mechanism is simple but has been proven to be an effective scheme for better I/O performance [24].
- *Signature-based prefetching scheme (IOSig+)*, which is a typical data prefetching scheme on the client file systems by analyzing applications' access patterns, and the source code is also available [4]. *IOSig+* allows users to characterize the I/O access patterns of an application in two steps: 1) trace collecting tool can get the trace of all the I/O operations of the application; 2) through the offline analysis on the trace, the analyzing tool gets the I/O signature [30]. Thus, the file system can enable data prefetching by employing the fixed I/O signatures.
- *Initiative prefetching scheme (Initiative)*, which is our proposed prefetching technique. In addition to predicting future I/O access events, the storage servers are responsible to prefetch the block data and forward it to the corresponding client file systems.

TABLE 1
Specification of Nodes on the Cluster and the LANs

	Cluster	LANs
CPU	4xIntel(R) E5410 2.33G	Intel(R) E5800 3.20G
Memory	1x4 GB 1,066 MHz/DDR3	4 GB DDR3-SDRAM
Disk	6x114 GB 7,200 rpm SATA	500 GB 7,200 rpm SATA
Network	Intel 82598EB, 10GbE	1,000 Mb or 100 Mb
OS	Ubuntu 13.10	Debian 6.0.4

4.1.3 Benchmarks

It is well-known that the prefetching scheme is able to improve I/O performance only if the running applications are database-related OLTP and server-like programs. Therefore, we selected three related benchmarks including an OLTP application benchmark, a server-like application benchmark and the benchmark that has certain regular access patterns to evaluate the initiative prefetching scheme. In the evaluation experiments, all benchmarks are running on 12 client nodes of two LANs by default if there are no particular specifications.

- *Sysbench*, which is a modular, cross-platform and multi-threaded benchmark tool for evaluating OS parameters that are important for a system running a database under intensive loads [13]. It contains several programs and each program exploits the performance of specific aspects under Online Transaction Processing workloads. As mentioned before, the prefetching mechanism makes sense for database-related OLTP workloads, thus, we utilized *Sysbench*to measure the time required for processing online transactions to show the merits brought about by our proposed prefetching mechanism.
- *Filebench*, which allows generating a large variety of workloads to assess the performance of storage systems. Besides, *Filebench* is quite flexible and enables to minutely specify a collection of applications, such

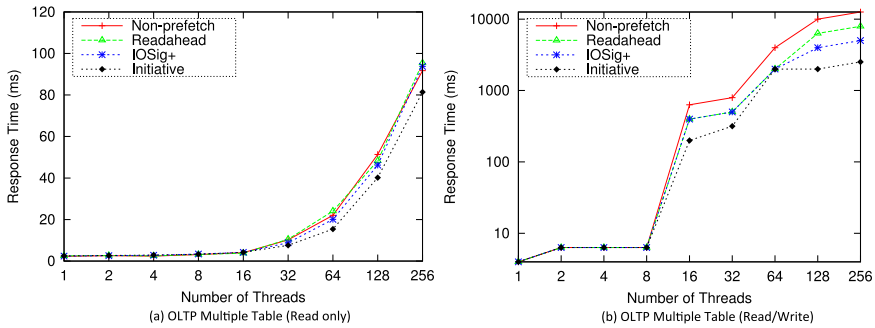

Fig. 8. *Sysbench* experimental results on I/O responsiveness: *X*-axis represents the number of threads on each client node, and *Y*-axis indicates the time required for handling each OLTP transaction, in which the lower value is preferred.

as mail, web, file, and database servers [12]. We chose *Filebench* as one of benchmarks, as it has been widely used to evaluate file systems by emulating a variety of several server-like applications [9].

- *IOzone*, which is a micro-benchmark that evaluates the performance of a file system by employing a collection of access workloads with regular patterns, such as sequential, random, reverse order, and strided [11]. That is why we utilized it to measure read data throughput of the file systems with various prefetching schemes, when the workloads have different access patterns.

4.2 I/O Processing Acceleration

In fact, I/O responsiveness of file systems is a crucial indicator to express their performance, and our primary motivation to introduce the initiative prefetching scheme is to provide I/O service for OLTP applications. Thus, we employed the online transaction processing benchmark of *SysBench* suite with a *MySQL*-based database setup to examine the validity of the *Initiative* prefetching scheme. We selected *SysBench* as the benchmark to create OLTP workloads, since it is able to create similar OLTP workloads that exist in real systems. All the configured client file systems executed the same script, and each of them run several threads that issue OLTP requests. Because *Sysbench* requires *MySQL* installed as a backend for OLTP workloads, we configured *mysqld* process to 16 cores of storage servers. As a consequence, it is possible to measure the response time to the client request while handling the generated workloads.

In the experiments, we created multiple tables with total 16 G of data, and then set the benchmark has a fixed number of threads on each client node to perform 100,000 database transactions. Moreover, *Sysbench* has two operating modes: default mode (**r/w**) that reads and writes to the database, and a readonly mode (**ro**). The r/w mode executes the following database operations: *five SELECT operations, two UPDATE operations, one DELETE operation and one INSERT operation*. To put it in proportion specifications about both operations, the observed read/write ratio is about 75 percent reads and 25 percent writes. Figs. 8a and 8b illustrate the response times while working in **ro** and **r/w** operating modes respectively. It is clear that our *Initiative* scheme brought about the least response times in both operating modes. That is to say, as storage servers could make accurate predictions about future I/O events for *Sysbench*, the *Initiative* prefetching technique outperformed other prefetching schemes, even though when the servers were heavily loaded.

4.3 Data Bandwidth Improvement

In this section, we first measure read data throughput by using *IOzone* benchmark, and then we evaluate overall data throughput and processing throughput by executing *filebench* benchmark. The experimental results and related discussions will be shown in the following two sections respectively.

4.3.1 Read Data Throughput

We have verified read data throughput of the file system while employing the mentioned prefetching schemes, through performing multiple kinds of read operations that generated by the *IOzone* benchmark. Fig. 9 reports the experimental results, and each sub-figure shows the data throughput when read operations follow a specific access pattern. In all sub-figures, the horizontal axis shows record sizes for each read operation, the vertical axis shows data rate, and the higher data rate indicates the better performance of the prefetching scheme.

From the data presented in the figure, we can safely conclude that the *Initiative* prefetching scheme performed the best in most of cases when running the *IOzone* benchmark. Especially, while conducting both backward reads and stride reads, whose results are demonstrated in sub-figures (b) and (c) in Fig. 9, our proposed scheme increased data throughput by 30 to 160 percent, in contrast to the *Readahead* prefetching scheme and the *Non-prefetching* scheme. Besides, it is obvious that *IOSig+*, which is a typical client-side prefetching scheme, does not work better than the *Non-prefetching* scheme in a major part of cases. That might because *IOSig+* does not have any effective algorithms to classify access patterns or any practical implementation of this notion [29].

4.3.2 Overall Data Throughput

As we demonstrated before, *IOzone* could measure the read data throughput with various types of read access patterns, but it fails to unveil overall performance of the file system, while applications have mixed access patterns. Thus, we also leveraged *FileBench* to measure overall data and processing throughput, since *FileBench* is able to create filesets that have varying sizes files prior to actually running a

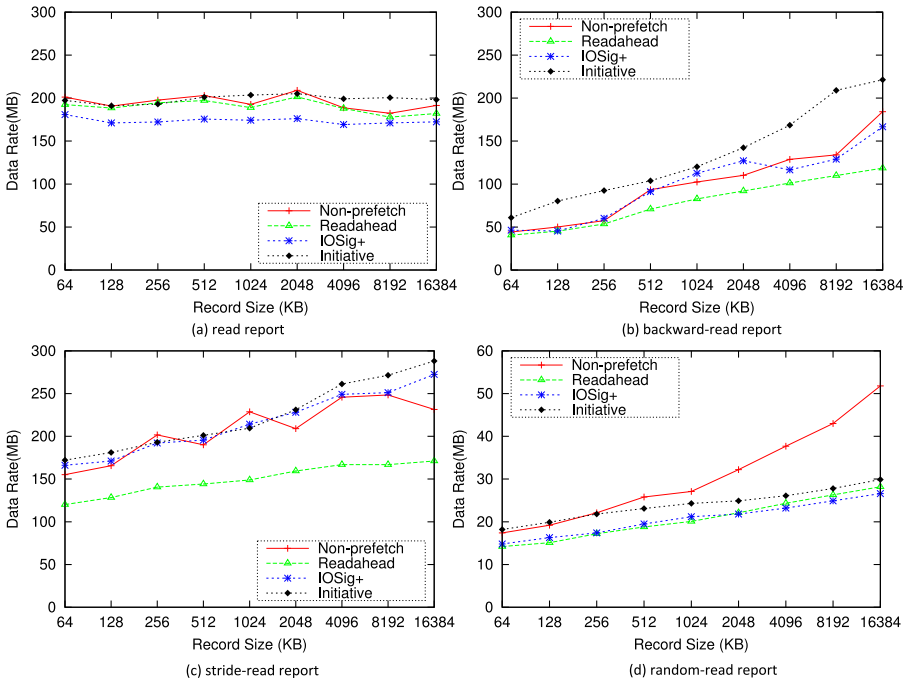

Fig. 9. Read data throughput while running *IOzone* benchmark: (a) in the read report, all the prefetching schemes have the similar results, it seems that prefetching schemes do not have positive effects on I/O performance while read operations do not have certain regularities. (b) in the backward read report, both *IOSig+* and *Initiative* prefetching scheme can achieve preferable data throughput. (c) in the stride read report, the proposed *Initiative* prefetching scheme outperforms others. (d) in the random read report, the *Non-prefetching* scheme performs better than other three prefetching schemes. That is because the prefetching scheme consumes computing power and memory capacity to predict future I/O operations and then to guide prefetching data, but the prefetched data may not be used in random read patterns, i.e., prefetching leads to negative effects in this case.

workload. The workloads of web, database, email and file server have been chosen in our tests, because they can represent most common server workloads, and they differ from each other distinctly [8], [9]. All tests were piloted by using the workflow definitions provided by *Filebench* with *run 60*, and we only recorded the results of server applications, i.e., mail server (varmail), file server, web server and database server (oltp) because the prefetching scheme can benefit these server-like applications.

Fig. 10 shows the overall data throughput and processing throughput caused by various prefetching schemes. Without doubts, compared with other prefetching schemes, the newly proposed *Initiative* prefetching technique resulted in 16-28 percent performance improvements on overall data throughput, as well as 11-31 percent performance enhancements on processing throughput. This is because the *Initiative* prefetching scheme can enhance the data throughput on read operations, and the overall data throughput can be improved consequently.

4.4 Benefits upon Client Nodes

We have moved the prefetching functionality from client nodes that might be resource-limited, to storage servers in a distributed file system for alleviating the consumption of computing and memory resources on the client nodes. In order to show the advantages to the client nodes while performing prefetching on the servers, we have programmed a matrix multiplication benchmark that runs on the client nodes. Fig. 11 demonstrates its workflow, which reads the matrix elements from files stored on the distributed file system. We intend to demonstrate that the task of matrix

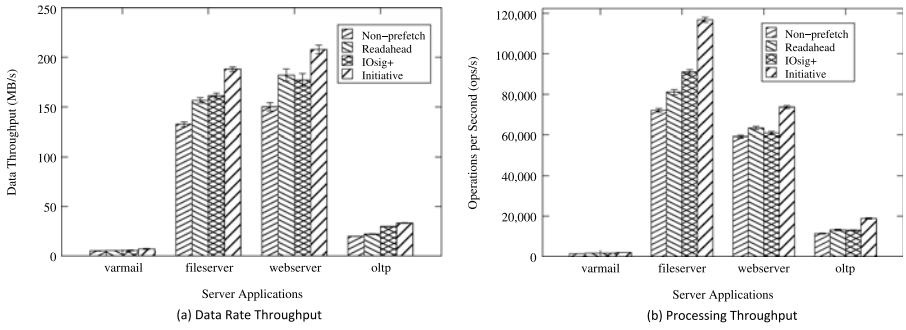

Fig. 10. *Filebench* experimental results on overall data throughput and processing throughput: (a) demonstrates overall data throughput, *X*-axis represent four selected server applications in *Filebench*, *Y*-axis shows overall data throughput and the higher one is preferred. (b) explores the results about processing throughput, and the vertical axis represents the number of handled read & write operations per second; moreover, the average latency for each operation including read and write in the selected server applications were 4.4, 1.5, 0 and 0.2 ms respectively.

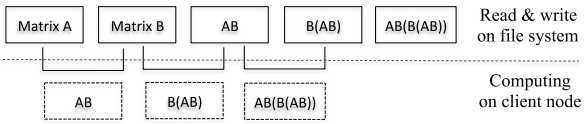

Fig. 11. The algorithm of matrix multiplication benchmark: there are three times of multiplication, and the intermediate results are flushed back to the file system. In other words, all input data should be read from the file system.

TABLE 2
Initative Prefetching Overhead

Benchmark	Time (%)	Space (MB)	Delay / Delay+3R (%)
IOzone	10.5	170.7	4.8/ 18.3
Sysbench	3.8	79.2	1.3 / 3.1
Filebench	7.2	124.1	5.1 /19.6
Matrix (1,000)	2.2	0.32	0.8 / 1.9

multiplication can be completed on the client nodes quickly when the *Initiative* prefetching scheme is enabled, in contrast to adopting *IOsig+* that is a typical client prefetching scheme.

After running the benchmark for multiplying the matrices with varying sizes, we measured the time needed for completing the multiplication tasks. Fig. 12a shows the time required for performing matrix multiplication on the file systems with different prefetching schemes. It is clear that the newly presented initiative prefetching scheme needed the least time to conducting multiplication tasks when the matrix size is bigger than 200. For instance, while the matrix size is 1,000, which means there are $1{,}000 \times 1{,}000$ matrix elements, the *Initiative* scheme could reduce the completion time by up to 21 percent compared to the other schemes. In short, the less completion time means less CPU and memory consumption on the client node.

Another critical information demonstrated in the experiments of executing matrix multiplication is about the overhead caused by conducting data prefetching on the client sides. The newly proposed *Initiative* scheme is a prefetching mechanism on the server sides, which aims to reduce the prefetching overhead on the resource-limited client machines. Therefore, we have also reported the time required for completing the prefetching-relevant tasks, such as conducting future access prediction through executing complicated algorithms on the client machines when running matrix multiplication. Fig. 12b demonstrates the overhead on the client nodes introduced by different prefetching mechanisms. It is obvious that both *Non-prefetching* and *Readahead* did not result in any prefetching overhead, because there are no trace logging and access predictions in both schemes. The *Initiative* prefetching scheme caused a little prefetching overhead on the client sides, because it requires the client file system to piggyback the client information onto the I/O requests. But *Initiative* outperformed *IOsig+* at the time needed to support the prefetching functionality to a great extend. For example, it saves more than 3,700 percent time on the client nodes in contrast to *IOsig+*, while the size of matrix is $1{,}000 \times 1{,}000$.

Note that although we have demonstrated that *IOSig+* introduces certain prefetching overhead on the client nodes, but it works better than *Non-Prefetch*, which has been verified in Fig. 12a before. This is because the application of matrix multiplication is a benchmark to read data regularly, the gains resulted by the client prefetching scheme could offset the overhead caused by conducting prediction on the client machines.

4.5 Overhead on Storage Servers

It has been confirmed that the proposed server-side prefetching scheme, i.e., *Initiative* indeed achieves better I/O performance, as well as less overhead on client nodes, but it is necessary to check the prefetching overhead on storage servers for ensuring whether this mechanism is practical or not. Table 2 illustrates the running time, space overhead and network delays for initiative prefetching on the storage servers. The results demonstrate that the *Initiative* prefetching technique can effectively and practically forecast future disk operations to guide data prefetching for different workloads with acceptable overhead. Since both *IOzone* and *Filebench* are typical I/O benchmarks to test I/O performance of storage systems, they caused more than 7.2 percent time overhead on prefetching and pushing prefetched data. But, for compute-intensive applications and OLTP workloads, our proposed mechanism utilized not much time to yield preferable system performance. For example, while the workload is matrix multiplication, the time required for predicting future disk operations to direct data prefetching and pushing the data to relevant client file systems was around 2.2 percent of total processing time. Therefore, we can see that this newly presented server-side prefetching is practical for storage systems in cloud computing environments.

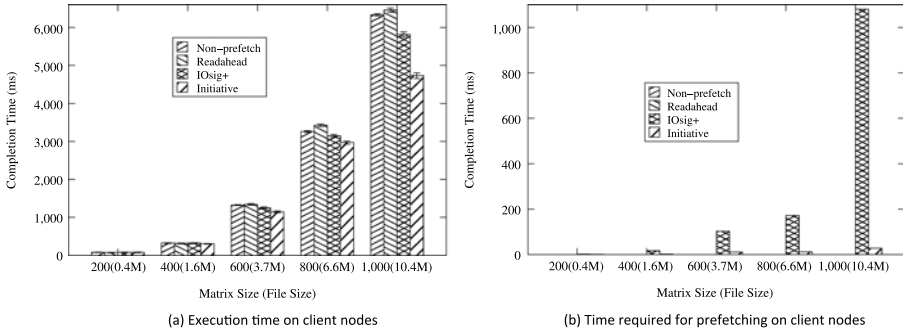

Fig. 12. The time needed for matrix multiplication on the client node: *X*-axis expresses the size of matrix and the size of the file that stores one of the target matrices. *Y*-axis in (a) presents the time needed for conducting matrix multiplication, and the lower the better, and *Y*-axis in (b) shows the time required for performing prefetching-related tasks when conducting matrix multiplication.

The consumed space for storing disk traces and the classified access patterns to conduct I/O prediction is also demonstrated in Table 2. It is clear that the presented initiative prefetching scheme is efficient in terms of space overhead for a major part of traces. The results show that running *IOzone* benchmark resulted in more than 100 MB but less than 180 MB tracing logs, as it is a typical I/O benchmark and focuses on I/O operations rather than computing tasks. Moreover, it is clear that less than 80 MB space was used for storing disk traces to forecast future disk I/Os while the benchmark is *Sysbench* (i.e., OLTP workloads). In a word, analyzing disk traces and prefetching block data can run on the same machine as the storage system without causing too much memory and disk overhead. For long-time running applications, the size of trace logs may become extraordinary large, in this case, the initiative prefetching may discard certain logs that occurred at the early stage, because disk block correlations are relatively stable during certain period, and the disk access logs at earlier stages are not instructive to forecast future disk access [35], [39].

Moreover, since transferring the prefetched data must bring about extra network traffics, which may affect the response times to I/O requests excluding the prediction-hit requests. To disclose the percentage of network delays to the requests, we first measured the response times to these requests caused by the system with prefetching or without prefetching; and then we computed the percentage of network delay according to Equation (11),

$$Delay = \frac{\sum_{i=1}^{n}[(Ti_{recv'} - Ti_{snd'}) - (Ti_{recv} - Ti_{snd})]}{\sum_{i=1}^{n}(Ti_{recv} - Ti_{snd})}, \qquad (11)$$

where $(Ti_{recv} - Ti_{snd)}$ indicates the response time to request i caused by the system with initiative prefetching, and $(Ti_{recv} - Ti_{snd})$ means the response time resulted by the system without prefetching.

Table 2 reports the network delay percentages when the storage servers have no copy and three replicas for each for stripe respectively. It is obvious that the traffics of prefetched data did place no more than 5.1 percent network delays to the response time to the requests excluding prediction-hit ones there was no copy for each data stripe (labeled as *Delay* in the table) . But the percentage of network delays increases significantly while there are three replicas for each stripe (labeled as *Delay + 3R*), that is because all storage servers having the same stripe replica are possible to push the prefetched data to the same client file system, the amount of data on the network may be increased manyfold.

4.6 Analysis of Prediction Algorithms

We not only exploited the advantages resulted by the initiative prefetching scheme, but also made an analysis of prediction error/deviation when applying our estimation models on the different workloads, to show the effectiveness of our prediction algorithms. In other words, we recorded the number of prediction hits, as well as the number of predictions and the total read operations occurred on the storage servers, while running the selected benchmarks in evaluation experiments.

TABLE 3
Prediction Hit Ratio

Benchmark	Reads	Predictions	Hits (Percentage)
IOzone:read/re-read	2,097,184	1,235,247	1,016,608 (82%)
IOzone:backward-read	1,048,576	924,739	780,479 (84.4%)
IOzone:stride-read	1,048,576	943,718	823,866 (87.3%)
IOzone:random-read	2,097,152	734,003	416,179 (56.7%)
Sysbench:read/write	3,326,112	2,135,436	1,665,640 (78.0%)
Sysbench:read-only	2,372,024	1,549,248	1,275,031 (82.3%)
Matrix (200 × 200)	91	88	87 (98.9 %)
Matrix (400 × 400)	355	352	347 (99.1 %)
Matrix (600 × 600)	795	790	782 (99.0 %)
Matrix (800 × 800)	1,410	1,401	1,389 (99.1 %)
Matrix (1,000 × 1,000)	2,201	2,188	2,170 (99.2 %)

Table 3 demonstrates the results of related statistics elaborately. From the reported data in the table, we conclude that our prediction models are not responsible for predicting all future read operations on the disk. Namely, only the disk I/Os follow either a linear trend or a chaotic trend, which can be distinguished by modeling disk I/O access patterns, relevant read predictions will be performed. Moreover, our presented two prediction algorithms resulted in attractive prediction accuracy, which indicates better hit percentage. As a consequence, I/O performance can be enhanced to a great extend by employing the initiative data prefetching mechanism.

4.7 Case Study with Real Block Traces

To check the feasibility of the proposed mechanism in practice, we have chosen two widely used state-of-the-art I/O traces, i.e., *Financial 1* and *Financial 2*, which describe block traces of OLTP applications extracted from two large financial institutions [5]. In the experiments, the client file system issues the I/O requests to the storage servers for writing/reading block data, according to the log events in the traces. Fig. 13 shows the experimental results about average response time to read and write requests in the selected two traces. In the figure, *X*-axis shows the trace name and the operation type, and *Y*-axis illustrates the average I/O response time (the lower one is better).

It is obvious that the *Non-prefetching* achieved the best I/O response time on write operations, because it is not affected by prefetching on either client machines or storage servers. Furthermore, compared with other prefetching schemes, the newly proposed *Initiative* prefetching could reduce average I/O response time on read operations by 26.4-10.6 percent in contrast to *Readahead*, and 16.7-19.4 percent compared to *IOSig+*. That is because the proposed mechanism can model block access events and classify access pattern among them for precisely guiding data prefetching on storage servers, which can benefit the read requests in the track of I/O requests.

5 Conclusions

We have proposed, implemented and evaluated an initiative data prefetching approach on the storage servers for distributed file systems, which can be employed as a back-end storage system in a cloud environment that may have

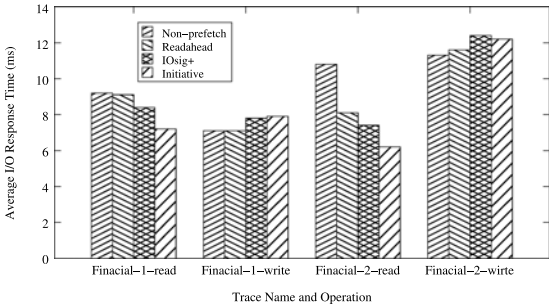

Fig. 13. The average I/O response time of the selected block traces.

certain resource-limited client machines. To be specific, the storage servers are capable of predicting future disk I/O access to guide fetching data in advance after analyzing the existing logs, and then they proactively push the prefetched data to relevant client file systems for satisfying future applications' requests. For the purpose of effectively modeling disk I/O access patterns and accurately forwarding the prefetched data, the information about client file systems is piggybacked onto relevant I/O requests, then transferred from client nodes to corresponding storage server nodes. Therefore, the client file systems running on the client nodes neither log I/O events nor conduct I/O access prediction; consequently, the thin client nodes can focus on performing necessary tasks with limited computing capacity and energy endurance. Besides, the prefetched data will be proactively forwarded to the relevant client file system, and the latter does not need to issue a prefetching request. So that both network traffics and network latency can be reduced to a certain extent, which have been demonstrated in our evaluation experiments.

To sum up, although the initiative data prefetching approach may place extra overhead on storage servers as they are supposed to predict the future I/Os by analyzing the history of disk I/O s, it is a nice option to build a storage system for improving I/O performance while the client nodes that have limited hardware and software configuration. For instance, this initiative prefetching scheme can be applied in the distributed file system for a mobile cloud computing environment, in which there are many tablet computers and smart terminals.

The current implementation of our proposed initiative prefetching scheme can classify only two access patterns and support two corresponding prediction algorithms for predicting future disk I/O access. After understanding of the fact about block access events generally follow the reference of locality, we are planning to work on classifying patterns for a wider range of application benchmarks in the future by utilizing the horizontal visibility graph technique. Because the proposed initiative prefetching scheme cannot work well when there is a mix of different workloads happening in the system, classifying block access patterns from the block I/O trace resulted by several workloads is one direction of our future work. Besides, applying network delay aware replica selection techniques for reducing network transfer time when prefetching data among several replicas is another task in our future work.

Acknowledgments

This work was partially supported by "National Natural Science Foundation of China (No. 61303038)" and "Natural Science Foundation Project of CQ CSTC (No. CSTC2013JCYJA40050)". The authors would like to thank Dr. Yanlong Yin for providing specifications about IOSig+. J. Liao is the corresponding author.

References

[1] J. Gantz and D. Reinsel. (2013). The digital universe in 2020: Big data, bigger digital shadows, biggest growth in the far east-united states. [Online]. Available: http://www.emc.com/collateral/analyst-reports/idc-digital-universe-united-states.pdf

[2] J. Kunkel and T. Ludwig, "Performance evaluation of the PVFS2 architecture," in *Proc. 15th EUROMICRO Int. Conf. Parallel, Distrib. Netw.-Based Process.*, 2007, pp. 509–516.

[3] S. Ghemawat, H. Gobioff, and S. Leung, "The Google file system," in *Proc. 19th ACM Symp. Oper. Syst. Principles*, 2003, pp. 29–43.

[4] IO Signature Plus (IOSIG+) Software Suite. (Oct. 2011). [Online]. Available: https://code.google.com/p/iosig/

[5] UMass Trace Repository: OLTP Application I/O. (May 2014). [Online]. Available: http://traces.cs.umass.edu/index.php/Storage/Storage

[6] MobiDFS: Mobile Distributed File System. (Nov. 2014). [Online]. Available: https://code.google.com/p/mobidfs/

[7] E. E. Marinelli, "Hyrax: Cloud computing on mobile devices using mapreduce," CMU, Pittsburgh, PA, USA, Tech. Rep. CMU-CS-09-164, 2009.

[8] P. Sehgal, V. Tarasov, and E. Zadok, "Evaluating performance and energy in file system server workloads," in *Proc. 8th USENIX Conf. File Storage Technol.*, 2010, pp. 253–266.

[9] V. Tarasov, S. Bhanage, E. Zadok, and M. Seltzer, "Benchmarking file system benchmarking: It* is* rocket science," in *Proc. 13th Workshop Hot Topics Operating Syst.*, 2011, pp. 1–5.

[10] N. Nieuwejaar and D. Kotz, "The galley parallel file system," *Parallel Comput.*, vol. 23, no. 4–5, pp. 447–476, 1997.

[11] IOzone Filesystem Benchmark. (Jul. 2010). [Online]. Available: http://www.iozone.org

[12] Filebench Ver. 1.4.9.1. (Nov. 2012). [Online]. Available: http://filebench.sourceforge.net

[13] SysBench benchmark. (Aug. 2013). [Online]. Available: http://sysbench.sourceforge.net

[14] M. Cencini, F. Cecconi, and N. Vulpiani, *Chaos from Simple Models to Complex Systems*. Singapore, World Scientific, 2010.

[15] J. Liao and Y. Ishikawa, "Partial replication of metadata to achieve high metadata availability in parallel file systems," in *Proc. 41st Int. Conf. Parallel Process.*, 2012, pp. 168–177.

[16] I. Y. Zhang, "Efficient file distribution in a flexible, wide-area file system," Master thesis, Dept. Elec. Eng. Comput. Sci., Massachusetts Inst. Technol., Cambridge, MA, USA, 2009.

[17] S. Akyurek and K. Salem, "Adaptive block rearrangement," *ACM Trans. Comput. Syst.*, vol. 13, no. 2, pp. 89–121, 1995.

[18] B. Bakke, F. Huss, D. Moertl, and B. Walk, "Method and apparatus for adaptive localization of frequently accessed, randomly accessed data," U.S. Patent 5765204, Jun., 1998.

[19] V. Padmanabhan and J. Mogul, "Using predictive prefetching to improve world wide web latency," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 26, no. 3, pp. 22–36, 1996.

[20] H. Lei and D. Duchamp, "An analytical approach to file prefetching," in *Proc. USENIX Annu. Tech. Conf.*, 1997, pp. 1–12.

[21] E. Shriver, C. Small, and K. A. Smith, "Why does file system prefetching work?," in *Proc. USENIX Annu. Tech. Conf.*, 1999, pp. 1–28.

[22] T. M. Kroeger and D. Long, "Predicting file system actions from prior events," in *Proc. USENIX Annu. Tech. Conf.*, 1996, p. 26.

[23] T. M. Madhyastha and D. A. Reed, "Exploiting global input/output access pattern classification," in *Proc. ACM/IEEE Conf. Supercomput.*, 1997, pp. 1–18.

[24] J. Stribling, Y. Sovran, I. Zhang, X. Pretzer, M. Kaashoek, and R. Morris, "Flexible, wide-area storage for distributed systems with WheelFS," in *Proc. 6th USENIX Symp. Netw. Syst. Des. Implementation*, 2009, pp. 43–58.

[25] X. Ding, S. Jiang, F. Chen, K. Davis, and X. Zhang, "DiskSeen: Exploiting disk layout and access history to enhance I/O prefetch," in *Proc. USENIX Annu. Tech. Conf.*, 2007, pp. 261–274.
[26] A. Reda, B. Noble, and Y. Haile, "Distributing private data in challenged network environments," in *Proc. 19th Int. Conf. world wide web*, 2010, pp. 801–810.
[27] N. Tran and D. A. Reed, "Automatic ARIMA time series modeling for adaptive i/o prefetching," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 4, pp. 362–377, Apr. 2004.
[28] S. Jiang, X. Ding, Y. Xu, and K. Davis, "A prefetching scheme exploiting both data layout and access history on disk," *ACM Trans. Storage*, vol. 9, no. 3, p. 23, 2013.
[29] J. He, J. Bent, A. Torres, G. Grider, G. Gibson, C. Maltzahn, and X. Sun, "I/O acceleration with pattern detection," in *Proc. 22nd Int. ACM Symp. High Perform. Parallel Distrib. Comput.*, 2013, pp. 26–35.
[30] S. Byna, Y. Chen, X. Sun, R. Thakur, and W. Gropp, "Parallel I/O prefetching using MPI file caching and I/O signatures," in *Proc. ACM/IEEE Conf. Supercomput.*, 2008, pp. 1–12.
[31] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka, "Informed prefetching and caching," in *Proc. 15th ACM Symp. Oper. Syst. Principles*, 1995, pp. 79–95.
[32] S. Weil, S. Brandt, E. Miller, D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. 7th USENIX Symp. Oper. Syst. Des. Implementation*, 2006, pp. 307–320.
[33] M. Al Assaf, X. Jiang, M. Abid and X. Qin, "Eco-Storage: A hybrid storage system with energy-efficient informed prefetching," *J. Signal Process. Syst.*, vol. 72, no. 3, pp. 165–180, 2013.
[34] J. Griffioen, and R. Appleton, "Reducing file system latency using a predictive approach," in *Proc. USENIX Annu. Tech. Conf.*, 1994, pp. 197–107.
[35] Z. Li, Z. Chen, and Y. Zhou, "Mining block correlations to improve storage performance," *ACM transactions on storage*, vol. 1, no. 2, pp. 213–245, May 2005.
[36] J. Oly and D. A. Reed, "Markov model prediction of I/O requests for scientific applications," in *Proc. 16th Int. Conf. Supercomput.*, 2002, pp. 147–105.
[37] W. Turek and P. Calleja, Technical Bulletin: High Performance Lustre Filesystems Using Dell PowerVault MD Storage. Cambridge, England, Univ. Cambridge, 2010.
[38] M. Bhadkamkar, J. Guerra, L. Useche, and V. Hristidis, "BORG: Block-reORGanization for self-optimizing storage systems," in *Proc. 7th Conf. File Storage Technol.*, 2009, pp. 183–196.
[39] Z. Li, Z. Chen, S. M. Srinivasan, and Y. Zhou, "C-Miner: Mining block correlations in storage systems," in *Proc. 3rd Conf. File Storage Technol.*, 2004, pp. 173–186.
[40] C. Ruemmler and J. Wilkes, "A trace-driven analysis of disk working set sizes," HP Labs, Palo Alto, CA, USA, Tech. Rep. HPL–OSR–93–23,1993.
[41] S. Narayan and J. A. Chandy, "Trace based analysis of file system effects on disk I/O," in *Proc. Int. Symp. Perform. Eval. Comput. Telecommun. Syst.*, 2004, pp. 1–8.
[42] S. Narayan, "File system optimization using block reorganization techniques," Master thesis, Dept. Elec. Comput. Eng., Univ. Connecticut, CT, USA, 2004.
[43] W. W. Hsu, A. Smith, and H. Young, "The automatic improvement of locality in storage systems," *ACM Trans. Comput. Syst.* vol. 23, no. 4, pp. 424–473, 2005.
[44] H. Song, Y. Yin, X. Sun, R. Thakur, and S. Lang, "Server-side I/O coordination for parallel file systems," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2011, pp. 1–11.
[45] M. S. Obaidat, "QoS-guaranteed bandwidth shifting and redistribution in mobile cloud environment," *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 181–193, Apr.-Jun. 2014.
[46] C. Chen, M. Won, R. Stoleru, and G. Xie, "Energy-efficient fault-tolerant data storage & processing in mobile cloud," *IEEE Trans. Cloud Comput.*, Jan.-Mar. 2015, vol. 3, no. 1, pp. 28–41.
[47] J. Liao, L. Li, H. Chen, and X. Liu, "Adaptive replica synchronization for distributed file systems," *IEEE Syst. J.*, 2014.

Jianwei Liao received the MS degree in computer science from the University of Electronic Science and Technology of China in 2006 and PhD degree in computer science from the University of Tokyo, Japan in 2012. Now, he works for the college of computer and information science, the Southwest University of China. His research interests are dependable operating systems and high-performance storage systems for distributed computing environments.

Francois Trahay received the MS and PhD degrees in computer science from the University of Bordeaux, in 2006 and 2009, respectively. He is now an associate professor at Telecom Sud-Paris. He was a postdoc researcher at the Riken institute in the Ishikawa Lab (University of Tokyo) in 2010. His research interests include runtime systems for high-performance computing (HPC) and performance analysis.

Guoqiang Xiao received the BE degree in electrical engineering from the Chongqing University of China in 1986 and the PhD degree in computer science from the University of Electronic Science and Technology of China in 1999. He is a full professor at the school of computer and information science, Southwest University of China. He was a postdoc researcher at the University of Hongkong from 2001 to 2003. His research interests include computer architecture and system software.

Li Li received the MS degree in computing science from the Southwest University of China in 1992 and the PhD degree in computing from the Swinburne University of Technology, Australia in 2006. She is a full professor at the Department of Computer Science, Southwest University, China. She was a postdoc researcher at Commonwealth Scientific and Industrial Research Organisation, Australia from 2007 to 2009. Her research interests include service computing and social network analysis.

Yutaka Ishikawa received the BE degree, MTech, and PhD degrees in electrical engineering from Keio University, Japan in 1982, 1984, and 1987, respectively. He is a professor at the Department of Computer Science, the University of Tokyo, Japan. He was a visiting scientist in the School of Computer Science at Carnegie Mellon University from 1988 to 1989. His current research interests include parallel/distributed systems and dependable system software. He is a member of the IEEE.