2017 IEEE 41st Annual Computer Software and Applications Conference

HybridFS - A High Performance and Balanced File System Framework with Multiple Distributed File Systems

Lidong Zhang*, Yongwei Wu†, Ruini Xue‡, Tse-Chuan Hsu§, Hongji Yang§, Yeh-Ching Chung¶
*Graduate School at Shenzhen, Tsinghua University, Shenzhen 518057, China
Email: zld15@mails.tsinghua.edu.cn
†Department of Computer Science and Technology,
Tsinghua National Laboratory for Information Science and Technology (TNLIST),Tsinghua University, Beijing 100084, China
Research Institute of Tsinghua University in Shenzhen,Shenzhen 518057, China
Email: wuyw@tsinghua.edu.cn
‡University of Electronic Science and Technology, Chengdu China
Email: xueruini@uestc.edu.cn
§Centre for Creative Computing, BathSpa University, England
Email: davidhsu@hcu.edu.tw, h.yang@bathspa.ac.uk
¶Research Institute of Tsinghua University in Shenzhen, Shenzhen 518057, China
Email: Yehching.chung@gmail.com

***Abstract*—In the big data era, the distributed file system is getting more and more significant due to the characteristics of its scale-out capability, high availability, and high performance. Different distributed file systems may have different design goals. For example, some of them are designed to have good performance for small file operations, such as GlusterFS, while some of them are designed for large file operations, such as Hadoop distributed file system. With the divergence of big data applications, a distributed file system may provide good performance for some applications but fails for some other applications, that is, there has no universal distributed file system that can produce good performance for all applications. In this paper, we propose a hybrid file system framework, HybridFS, which can deliver satisfactory performance for all applications. HybridFS is composed of multiple distributed file systems with the integration of advantages of these distributed file systems. In HybridFS, on top of multiple distributed file systems, we have designed a metadata management server to perform three functions: file placement, partial metadata store, and dynamic file migration. The file placement is performed based on a decision tree. The partial metadata store is performed for files whose size is less than a few hundred Bytes to increase throughput. The dynamic file migration is performed to balance the storage usage of distributed file systems without throttling performance. We have implemented HybridFS in java on eight nodes and choose Ceph, HDFS, and GlusterFS as designated distributed file systems. The experimental results show that, in the best case, HybridFS can have up to 30% performance improvement of read/write operations over a single distributed file system. In addition, if the difference of storage usage among multiple distributed file systems is less than 40%, the performance of HybridFS is guaranteed, that is, no performance degradation.**

I. Introduction

In recent years, the distributed file system (DFS), such as Ceph [24], Hadoop distributed file system (HDFS) [3], GlusterFS [10] , Google file system (GFS) [9], etc., is getting more popular due to the characteristics of its scale-out capability, high availability, and high performance for big data applications. For different usage purposes, the design of these distributed file systems is different. Some distributed file systems are designed to handle small files, such as GlusterFS. Some distributed file systems are designed for large files, such as HDFS. While some file systems are designed with multiple file storages for different use cases, such as Ceph. Since the design goals of different distributed file systems are different, their performance may vary for different file sizes. For example, when the file sizes are less than 8MB, the performance of read/write operations of GlusterFS is better than that of HDFS. On the other hand, if the file sizes are greater than or equal to 8MB, the performance of read/write operations of HDFS is better than that of Ceph which is better than that of GlusterFS. This indicates that there has no universal distributed file system that can deliver good performance for all kinds of file sizes.

To overcome the above drawback, some research has focused on promoting the performance, availability and scalability of Multi-Cloud Storage (MCS) [18] [23] by using the erasure code. The MCS with erasure code scheme divides a data object into k equal-sized fragments that are the original data shares. Each data share has m-k redundant, where m is the number of storages to store a data object and $m \geq k$. From the m storages where a data object is stored, we only need shares from any k storages to reconstruct the original data object. This scheme suffers from low throughput and high latency for not considering the characteristics of different storage systems and the geographical locations of storages. Some research has

0730-3157/17 $31.00 © 2017 IEEE
DOI 10.1109/COMPSAC.2017.140

IEEE computer society

Authorized licensed use limited to: The Chinese University of Hong Kong CUHK(Shenzhen). Downloaded on November 11,2024 at 05:40:22 UTC from IEEE Xplore. Restrictions apply.

focused on using multiple DFSs with a file placement strategy to decide what DFS to store a file based on the usage of DFSs and the current throughput of DFSs [21]. However, the scheme proposed in [21] did not consider some important file characteristics, such as access permission, etc., which have a great impact on DFS performance, resulting in low write/read throughputs.

In this paper, we propose a hybrid file system framework, HybridFS, which is composed of multiple DFSs with the integration of advantages of these DFSs. In contrast to the scheme proposed in [21], the design of HybridFS takes the characteristics of files and DSFs into considerations. Therefore, HybridFS can deliver satisfactory performance for all big data applications. To design a file system with multiple DFSs, there are two challenges:

(1) How to intelligently decide what DFS to store a file such that the system has highest overall performance;

(2) How to balance the usage of DFSs.

To cope with these challenges, in HybridFS, an intelligent metadata management server (MMS) is designed on top of multiple DFSs with three functions, file placement, partial metadata store, and dynamic file migration. The file placement function is used to intelligently decide what DFS to store a file based on a decision tree that is one of the classic machine learning algorithm. Machine learning has a natural advantage for big data applications. It can automatically identify and model attributes required with high efficiency. The decision tree is constructed based on two file attributes size and permission. The partial metadata store function is used to store partial file metadata (file name, file path, primary file location, file size, read frequency, write frequency, operation frequency, and file data) of files whose size is less than a few hundred Bytes to increase throughput. For example, if a read operation is performed for a small size file, we can retrieve its data from the partial metadata stored in MMS without the involvement of DFSs. We use the small file optimization method proposed in [15] to manage the metadata of small size of files and dynamically adjust the store threshold based on the file size and MMS space usage, etc. When using multiple DFSs, it is possible that the storage usage of one DFS is full while that of another DFS is almost empty if the files processed by the file placement policy are favoring one DFS and there has no file migration mechanism. In HybridFS, the dynamic file migration function is used to migrate files from one DFS to another in order to balance the storage usage of DFSs without throttling performance. In the dynamic file migration function, we propose a dynamic storage balance model based on the file size, the file read/write throughput that is predicted by the regression analysis [7], the difference of storage usage of DFSs, etc. With this model, we can decide what file should be migrated to what DFS and the performance is almost no loss.

We have implemented HybridFS in java on eight nodes and choose Ceph, HDFS, and GlusterFS as designated distributed file systems. The experimental results show that, in the best case, HybridFS can have up to 30% performance improvement of read/write operations over a single distributed file system. In addition, if the difference of storage usage among multiple distributed file systems is less than 40%, the performance of HybridFS is guaranteed, that is, no performance degradation. The contributions of this paper are as follows:

- We have proposed a hybrid file system framework, HybridFS, which can deliver satisfactory performance for all big data applications. The framework is generic and can be used for different DFSs combinations as long as the characteristics of files and DFSs can be derived.
- We use the decision tree to classify different characteristics of files and design a novel place strategy to put a files in a proper DFS.
- We have designed a dynamic file migration function to balance the the storage usage of DFSs in HybridFS in the premise that system throughput is almost unchanged.

The remainder of this paper is organized as follows. In section II, the related work is given. Section III describes the design details of HybridFS. The evaluation and analysis of proposed framework is given in Section IV. Sections V concludes the paper and points out some possible future directions.

II. Related Work

A. Distributed File System

Many efficient and practical DFSs have been proposed in the literature, such as Ceph, HDFS, GlusterFS. Ceph is one of the most efficient Distributed file system that maximizes the separation between data and metadata management. It uses pseudo-random data distribution function (CRUSH) to store data and a dynamic subtree partitioning mechanism to place metadata. Since the subtree partitioning metadata operation is efficient, Ceph can deliver good performance for small file operations.

HDFS is an Apache open source software. Its metadata mechanism is different from that of Ceph. In HDFS, a namenode is the master server that manages the name space of file system and handles file access requests by clients. The datanodes are common storage nodes. The design goal of HDFS is to provide a reliable store for large files across machines in a large cluster. To achieve this goal, in HDFS, each file is stored as a sequence of blocks on datanodes. Each block has a number of replicas specified by the replica placement policy managed by the active namenode. With streaming data access patterns [25], HDFS are suitable for large file operations.

In GlusterFS, the storage nodes are divided into two categories, client and server. The server nodes are typically deployed as storage bricks. Each server node has a GlusterFS daemon to export the local file system as a sub-volume. The storage file-system is assigned to a volume that is composed of all sub-volumes of server nodes. The system does not have a metadata server to handle files. Instead it uses the elastic hash algorithm for file placement. By eliminating the metadata server, many file operations can be performed in parallel.

Authorized licensed use limited to: The Chinese University of Hong Kong CUHK(Shenzhen). Downloaded on November 11,2024 at 05:40:22 UTC from IEEE Xplore. Restrictions apply.

Therefore, GlusterFS can handle small files more efficient than Ceph and HDFS.

In [5] [8], the authors have evaluated the performance of Ceph, GusterFS, and HDFS. The evaluation indicates that Ceph and GlusterFS can produce better performance than HDFS for small files operations. On the opposite, HDFS has better performance than Ceph and GlusterFS for large files. In [12] [16] [22] , some optimizations have been proposed for HDFS to handle small file operations. However, these optimizations still cannot meet the demand required by applications. In [4], an optimization method has been proposed for a generic DFS based on metadata storage, usage of caching, and design of replication algorithms. Since this method does not consider the characteristics of a specific DFS, its performance is not ideal.

B. Multiple Cloud Storage System

The MCS proposed in [18] [23] integrates multiple storages that located in different geographical locations to form a DFS with the erasure code scheme. The MCS provides better features such as availability and scalability, but have low performance and high latency due to not considering the characteristics of different storage systems and the geographical locations of storages.

Software define storage (SDS) [21] is a virtualization technology for cloud storage. It uses policy-based provisioning mechanism for data storage management to improve accessibility and usability of the hardware resources. In [21], a SDS has been proposed by using Ceph, HDFS, and Swift as the underlined storages. The proposed SDS can assign a file to an appropriate storage based on the usage of DFS. However, its performance is not good due to ignore some important file attributes, such as permission, etc.

C. Machine Learning and Load Balancing

Using machine learning to optimize the file system is another hot research. The machine learning method is very suitable for predicating and classifying files and jobs in a storage system. In [26], the authors proposed an efficient job classification approach, Bejo, to classify different jobs based on resource consumption. Bejo [26] treats the job as a document and assigns each collected resource consumption snapshot to a certain 'resource word'. Based on the distribution of words, it can classify a job using support vector machine (SVM). In [1], an unsupervised learning method was proposed to cluster Hadoop jobs that have similar characteristics. A decision tree defined in [19] is a decision supporting tool that uses a tree-like graph or a model of decisions to determine the possible consequences, such as chance event outcomes, resource costs, utility, etc. In [17], a decision tree is used to automatically classify the attributes of existing files (e.g., read-only access pattern, short-lived, small in size). Based on the classification, the attributes of new files can be predicated and the locations of new files can be determined.

Storage usage unbalancing is another challenge. Many methods have been proposed in the literature to solve this problem. In [11], a fully distributed load rebalancing algorithm was proposed to cope with the load imbalance problem based on the analysis of storage usage of a DFS. In [6], a dynamic and adaptive load balancing algorithm, SALB, based on a distributed architecture was proposed. SALB is composed of tow models. One is an online load prediction model that can reduce the decision delay caused by the network transmission. Another is a file migration model to select the possible migration candidates so as to minimal the overhead of file migration. Mantle [20] is a system built on Ceph. It exposes the trade-offs of resource migration and the processing capacity of the MDS nodes by separating migration policies from the migration mechanisms. It can select different techniques to distribute metadata and to balance diverse metadata workloads. In [2], the authors proposed a solution that enables administrators to make decisions in the presence of multiple workloads dynamically to strike an optimal balance point in between utilization and performance.

III. The Design of HybridFS

In this section we will discuss the design details of HybridFS. The design goal of HybridFS is to provide a satisfactory performance for all big data applications based on multiple DFSs. To achieve this goal, the design of metadata mechanism is critical. Two issues need to be considered. The first one is how to decide a proper DFS to store a given file such that a satisfactory performance can be guaranteed. The second one is how to balance the storage usage of DFSs without throttling the performance. To solve the first issue, we use a machine learning approach. For the second issue, we proposed a dynamic file migration mechanism that can balance the storage usage of DFSs.

The system architecture of HybridFS is shown in Figure 1. In Figure 1, we can see that HybridFS contains three components, clients, metadata manage server (MMS), and multiple DFSs. The clients are the file operations initiators. The MMS is responsible for the metadata management of HybridFS. The multiple DFSs are used to perform file operations. Among them, the MMS is the core of HybridFS. The MMS has three functions: file placement, partial metadata store, and dynamic file migration. The file placement function is used to determine a proper DFS to store a given file such that a satisfactory performance can be guaranteed for a given big data application. The partial metadata store function is used for files whose size is less than a few hundred Bytes to increase read or write throughput. The dynamic file migration function is used to balance the storage usage of DFSs without throttling performance. In the following, we describe the design of MMS in details.

A. The Design of File Placement Function

The file placement function in HybridFS uses a decision tree to determine what DFS to store a given file such that the best read/write performance can be achieved. In general, a decision tree algorithm recursively splits the samples into clusters, where each leaf node is a cluster. The goal is to

Authorized licensed use limited to: The Chinese University of Hong Kong CUHK(Shenzhen). Downloaded on November 11,2024 at 05:40:22 UTC from IEEE Xplore. Restrictions apply.

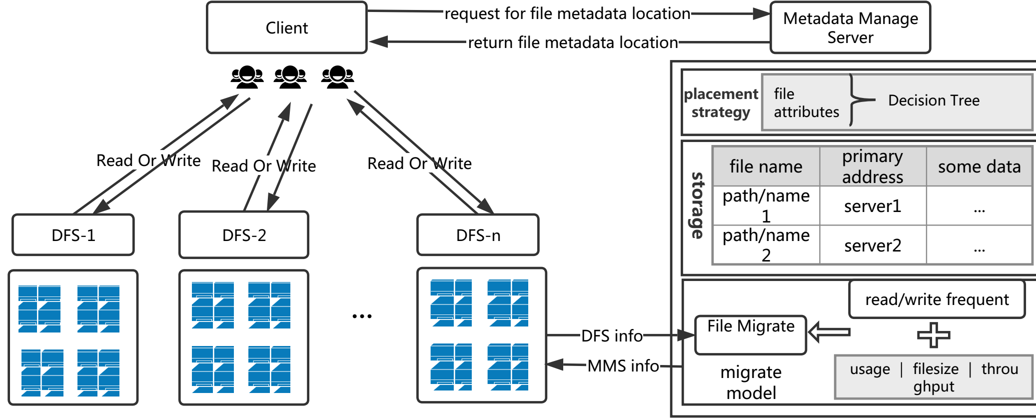

Fig. 1. The system architecture of HybridFS

create clusters whose files have similar attributes and the same classification. When a file is created, some attributes such as name, path, owner, group, size, time, permission, symbolic link, etc., are associated with the file. Among those attributes, only size and permission attributes are used in the decision tree for file classification.

The construction of decision tree can be divided into two phases. In the first phase, a 3-tuple (s, p, dfs) dataset is generated, where s is the file size attribute, p is the file permission attribute, and dfs is the designated DFS to store a file with s and p attributes. To generate the 3-tuple dataset, we need to determine the values of s, p, and dfs in the 3-tuple. The values of the file size attribute s are continuous. The information entropy theory [14] was used to discretize the continuous values. For the file permission attribute, its values are discrete. The values of file permission attribute p are read only, write only, and read/write. Given m DFSs and the values of s and p of a file f, the value of dfs is determined by the following equation

$$dfs = max(F_{irt} + F_{iwt}) \ for \ i = 1, 2, ..., m \quad (1)$$

where F_{irt} is the file read throughput in DFS_i and F_{iwt} is the file write throughput in DFS_i.

We now give an example to explain how to generate a 3-tuple dataset. Assume that the values of s are 1MB, 5MB, and 9MB. The values of p are read only, write only, and read/write. The values of The designated multiple DFSs are DFS_1, DFS_2, and DFS_3. The read throughput of a file with s = (1MB, 5MB, 9MB) in DFS_1, DFS_2, and DFS_3 are (10.5MB/s, 10MB/s, 9.5MB/s), (10.5MB/s, 10MB/s, 9.5MB/s), and (10.5MB/s, 11MB/s, 11.5MB/s), respectively. The write throughput of a file with s = (1MB, 5MB, 9MB) in DFS_1, DFS_2, and DFS_3 are (11MB/s, 10MB/s, 8MB/s), (10.5MB/s, 10MB/s, 9MB/s), and (8MB/s, 10MB/s, 11MB/s), respectively. For (s, p, dfs) = (1MB, read only, dfs), the value of dfs is DFS_1 since DFS_1 has the best read throughput (10.5MB/s) for file size = 1MB among DFSs used. For (s, p, dfs) = (5MB, write only, dfs), the value of dfs is DFS_1 since DFS_1 has the best write throughput (10.5MB/s) for file size = 5MB among DFSs used. For (s, p, dfs) = (9MB, read/write, dfs), the value of dfs is DFS_3 since DFS_3 has the best read + write throughput (22.5MB/s) for file size = 9MB among DFSs used. By enumerating cases of s and p, we can generate the corresponding 3-tuple dataset that like Figure 2(a).

In the second phase, a decision tree is generated based on the ID3 algorithm proposed by J. Ross Quinlan [19] and the 3-tuple dataset. The tree is built in a top-down manner. Initially, the tree is empty and the information entropy theory is used to determine the attribute that has the most impact on the classification result. In our design, the file size attribute s has the most impact on the classification result. It is selected as root. The dataset associated with the root is the original 3-tuple dataset. Next, a two-level decision tree can be constructed by insertion of n children to the root, where n is number of values of attribute s. Each newly inserted node is associated with a sub-dataset in which each instance has the same value in attribute s. For each newly inserted node, we apply the same process as performed for root to construct a 3-level decision tree and so on until all attributes are processed. When a decision tree is constructed, if all leaf nodes branched from a non-leaf node has the same classification result, the decision tree can be further optimized by merging these nodes as a leaf node.

Figure 2 illustrates how to construct a decision tree with a 3-tuple dataset. In Figure 2(a), there are 9 instances in the 3-tuples dataset. Based on the information entropy theory, attribute s is selected as root. Since the values of attribute s are 1M, 5M, and 9M, we insert 3 children nodes to the root and get a two-level decision tree as shown in Figure 2(c). The sub-datasets associated with each children node are shown in Figure 2(b). Next, for each children node, we apply the information entropy theory to attributes without s. Since only attribute p is available, attributed p is selected as the roots of children nodes of attribute s. The value of attribute p is read

Authorized licensed use limited to: The Chinese University of Hong Kong CUHK(Shenzhen). Downloaded on November 11,2024 at 05:40:22 UTC from IEEE Xplore. Restrictions apply.

only, write only, and read/write, three children nodes are added to each children node of attribute s. We obtain a three-level decision tree as shown in Figure 2(d). Since all attributes are processed, the construction of the decision tree is completed. In Figure 2(d), by merging some leaf nodes, we can obtain an optimized decision as shown in Figure 2(e).

To obtain the most realistic estimate of the true error, we calculate the training error using Ten-fold cross-validation [13]. Ten-fold cross-validation divide dataset into Ten-Equal size, take nine of them as training data and one as test data in turn. We can then use this estimate to automatically determine whether or not the model is suitable enough

B. The Partial Metadata Store Function

For Internet of Thing (IoT) applications, in general, they will produce many files with file size ranging from a few tens to a few hundreds Bytes. Since the design goals of most of DFSs are not for files with such small size, in HybridFS, we have proposed a partial metadata store mechanism to handle such files. In the partial metadata store mechanism, such small files are stored as metadata in MMS, instead of storing them on DFSs. A threshold is used to classify if a file is a small file or a normal file. If a file is classified as a small file and its file size is less than the available MMS storage capacity assigned for such small files, it is stored in MMS as metadata. Otherwise, it is stored in a DFS. In HybridFS, we use the small file optimization method proposed in [12] to dynamically adjust the threshold based on the file size and MMS space usage, etc.

C. Dynamic migrate model

In HybridFS, it is possible that the storage usage of one DFS is almost full while that of another DFS is almost empty if the files processed by the file placement function are favoring one DFS and there has no file migration mechanism. The purpose of the dynamic file migration function is to migrate exiting files from one DFS to another in order to balance the storage usage of DFSs without throttling performance. To achieve this goal, we propose a dynamic storage balance method based on the file size, the file read/write throughputs that are predicted by the regression analysis, and the difference of storage usage of DFSs. With this model, we can decide what file should be migrated to what DFS and the performance is almost no loss.

The read/write throughput of a file f is defined as the average flow rate of read/write operations on file f. To predict the read and write throughputs of a file, we first need to get the read/write throughputs of different size files on each DFS. Figure 3 shows an example of the read/write throughputs of different size files on GlusterFS, Ceph, and HDFS. Based on the curves of read/write throughputs shown in Figure 3, we can predict the read and write throughputs of a file based on the regression model proposed in [7]. In this paper, we only used the first-order, second-order, third-order, and fourth-order functions for read/write throughputs prediction. These functions are shown in Table I. To determine what function has the best prediction result for a curve c_i shown in Figure 3, the least-square method is first applied to calculate the unknowns of each function by using the throughputs presented in curve c_i. Then, a root mean square error (RMSE) equation is applied to calculate the error of throughputs predicted by each function and curve c_i. The function that has the smallest RMSE is selected as the function to predict the throughput associated with curve c_i.

In the dynamic file migration function, the difference of storage usage of DFSs is used as a threshold to determine whether files should be migrated from one DFS to another without throttling the performance. The threshold depends on the characteristics of DFSs used. In general, we need to perform a simulation before setting the threshold. We have a dynamic file migration daemon to monitor whether the threshold is satisfied. When a threshold is satisfied, the file migration process is launched to migrate files from the DFS with a higher storage usage, DFS_i, to the DFS with a lower storage usage, DFS_j. The file migration process is performed as follows:

Step 1. Calculate the read and write throughputs of file x on DFS_i and DFS_j using the regression functions associated with the corresponding curves, for x = 1, ..., z, where z the total number of files in DFS_i. Let $F_{xrt}(\text{DFS}_i)$ and $F_{xwt}(\text{DFS}_i)$ be the read and write throughputs of file x in DFS_i, respectively.

Step 2. Obtain the read and write frequency of all files in DFS_i from MMS. Let F_{xrf} and F_{xwf} denote the read and write frequency of file x, respectively.

Step 3. For all files in DFS_i, calculate the performance difference if they are migrated to DFS_j by using the following equation:

$$diff_x(\text{DFS}_i, \text{DFS}_j) = (s_x \div F_{xrt}(\text{DFS}_i) - s_x \div F_{xrt}(\text{DFS}_j)) \times F_{xrf} + (s_x \div F_{xwt}(\text{DFS}_i) - s_x \div F_{xwt}(\text{DFS}_j)) \times F_{xwf} \quad (2)$$

where s_x is the size of file x.

Step 4. Sort the values of $diff_x(\text{DFS}_i, \text{DFS}_j)$ in descending order for all files in DFS_i.

Step 5. Migrate the first y files with largest values of $diff_x(\text{DFS}_i, \text{DFS}_j)$ from DFS_i to DFS_j such that the difference of storage usage of DFS_i and DFS_j is less than the threshold. The dynamic file migration function is given in Algorithm 1.

In Algorithm 1, lines 2-10 are used to determine if the difference of storage usage of any two DFSs is over a threshold p_0 that is set by the system administrator. Line 14 uses Equation (2) to calculate the performance difference of every file in original DFS and in destination DFS. Line 15 is used to sort the performance difference in descending order. Lines 17-19 perform real file migration based on migrateList[] until the storage usage of two DFSs is less than threshold p_0.

IV. Experimental Evalutation

In the experimental evaluation, we use an eight-node clusters to verify the proposed methods. The configuration and

Authorized licensed use limited to: The Chinese University of Hong Kong CUHK(Shenzhen). Downloaded on November 11,2024 at 05:40:22 UTC from IEEE Xplore. Restrictions apply.

3-tiple dataset

size(s)	permission(p)	DFSs
1M	read only	DFS1(10.5MB/s)
1M	write only	DFS1(11MB/s)
1M	read,write	DFS1(21.5MB/s)
5M	read only	DFS1(10.5MB/s)
5M	write only	DFS1(11MB/s)
5M	read,write	DFS1(21.5MB/s)
9M	read only	DFS3(11.5MB/s)
9M	write only	DFS3(11MB/s)
9M	read,write	DFS3(22.5MB/s)

(a)

sub-dataset1

size(s)	permission(p)	DFSs
1M	read only	DFS1(10.5MB/s)
1M	write only	DFS1(11MB/s)
1M	read,write	DFS1(21.5MB/s)

sub-dataset2

size(s)	permission(p)	DFSs
5M	read only	DFS1(10.5MB/s)
5M	write only	DFS1(11MB/s)
5M	read,write	DFS1(21.5MB/s)

sub-dataset3

size(s)	permission(p)	DFSs
9M	read only	DFS3(11.5MB/s)
9M	write only	DFS3(11MB/s)
9M	read,write	DFS3(22.5MB/s)

(b)

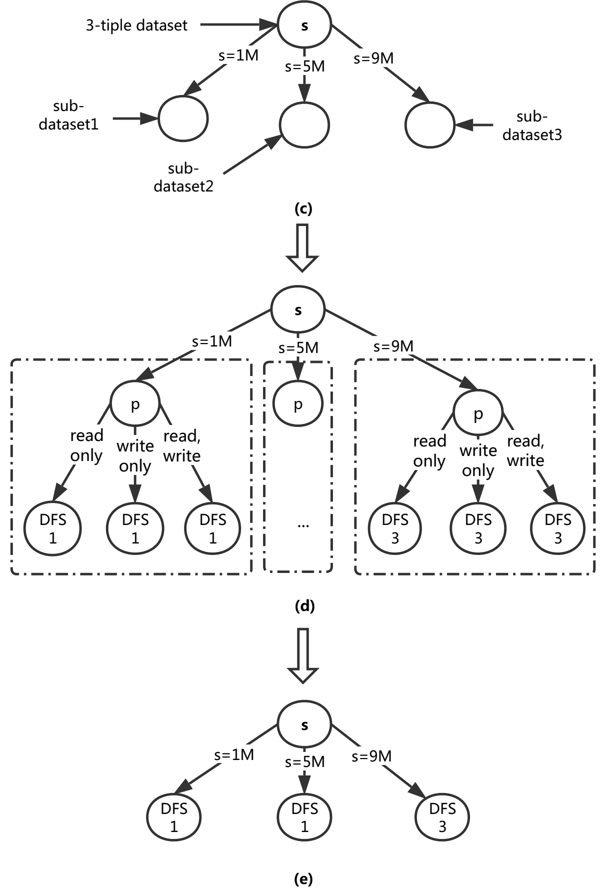

Fig. 2. An example of decision tree construction

TABLE I
DIFFERENT-ORDER REGRESSION MODELS

Model	Regression function
First-order model	$y(k) = a_0 + a_1 e^{-pk}$
Second-over model	$y(k) = a_0 + a_1 e^{-p_1 k} + a_2 e^{-p_2 k}$
Third-order system model	$y(k) = a_0 + a_1 e^{-pk} + be^{-\delta wk} \cos(w\sqrt{1-\delta^2}k) + ce^{-\delta wk} \sin(w\sqrt{1-\delta^2}k)$
Fourth-order system model	$y(k) = a_0 + b_1 e^{-\delta_1 w_1 k} \cos(w_1\sqrt{1-\delta_1^2}k) + c_1 e^{-\delta_1 w_1 k} \sin(w_1\sqrt{1-\delta_1^2}k)$ $+ b_2 e^{-\delta_2 w_2 k} \cos(w_2\sqrt{1-\delta_2^2}k) + c_2 e^{-\delta_2 w_2 k} \sin(w_2\sqrt{1-\delta_2^2}k)$

usage of the experimental environment is shown in Table II. In this cluster, all nodes are equipped with 4GB RAM, 128GB SSD, and 1TB HDD. The OS used is ubuntu14.04. The network between them is 100Mbps. The designated DFSs used in the experimental evaluation are HDFS, GlusterFS, and Ceph each with two nodes. By performing some file operations on different files size on these three DFSs, we observe that, for small files, the performance of these three DFSs has the order GlusterFS $>$ Ceph $>$ HDFS. On the opposite, for large files, the performance of these three DFSs has the order HDFS $>$ Ceph $>$ GlusterFS. This indicates that the file size is the most important attributed for throughput of DFSs. Therefore, our experimental evaluation is mainly on analyzing the file size impact to throughput. The read/write throughput of a file f on a DFS is defined as the file size of f divided by the time of read/write operation on file f. In the following, we evaluate and discuss how the file placement function, the partial metadata storage function, and the dynamic file migration function affect the throughputs of HybridFS.

Authorized licensed use limited to: The Chinese University of Hong Kong CUHK(Shenzhen). Downloaded on November 11,2024 at 05:40:22 UTC from IEEE Xplore. Restrictions apply.

TABLE II
NODES USAGE

Node number	DFS	Hostname	Usage	Notes
Node1	MMS	Master	Metadata manage server	1TB capacity
Node2	HDFS	HDFS1	Name node	1TB capacity
Node3	HDFS	HDFS2	Data node	1TB capacity
Node4	Ceph	Ceph1	mds,mon,osd	1TB capacity
Node5	Ceph	Ceph2	osd	1TB capacity
Node6	GlusterFS	GlusterFS1	GlusterFS server1	1TB capacity
Node7	GlusterFS	GlusterFS2	GlusterFS server2	1TB capacity
Node8	Client	Client	Client	1TB capacity

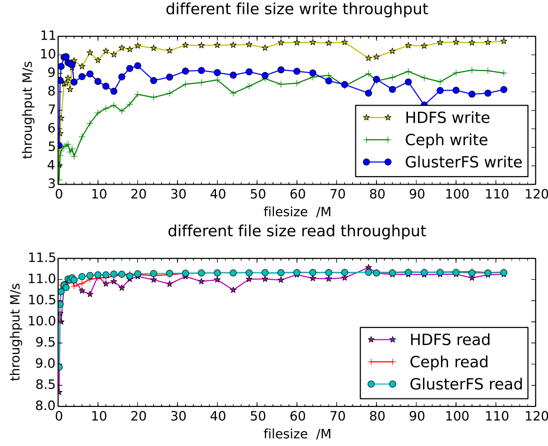

Fig. 3. The write/read throughputs of files with various sizes in designated DFS

A. The Impact of File Placement Function on Overall Throughput

In order to understand the impact of file size on throughput of DFSs, files with size ranging from 512 Bytes to 512MB are used to test the throughputs of HybridFS, GlusterFS, Ceph, and HDFS. Figure 4 demonstrates the test results. In Figure 4, $x-$axis is file size and $y-$axis is the normalized throughputs of designated DFSs by aligning the throughput of HybridFS to 1, that is, the throughput of each DFS is divided by that of HybridFS. From Figure 4, we have the following observations:

Observation 1: When the file size is less than 8MB, the throughputs of the four DFSs have the order GlusterFS $\approx$ HybridFS $>$ Ceph and HDFS.

Observation 2: When the file size is equal to 8MB, the throughputs of the four DFSs have the order GlusterFS $\approx$ HybridFS $\approx$ HDFS $>$ Ceph.

Observation 3: When the throughputs of the four DFSs is greater than 8MB, have the order HDFS $\approx$ HybridFS $>$ Ceph and GlusterFS.

From above observations, we conclude that the GlusterFS is suitable for storing small size files (size $<$ 8M). The HDFS is suitable for storing large size files (size $>$ 8M). The proposed HybridFS is suitable for storing both small and large size files. This indicates that the file placement function proposed

Algorithm 1 The Dynamic File Migration Function

Input: $p_0, DFSs$
Output: $null$

```
1:  for i = 0 to DFSs.size() do
2:      for j = i to DFSs.size() do
3:          if (DFSs[i].usage − DFSs[j].usage) > p_0 then
4:              originalLoc = i
5:              destinationLoc = j
6:              stop
7:          end if
8:      end for
9:  end for
10: if i=j then
11:     return null
12: end if
13: files[] = DFSs[originalLoc].files
14: Throuhput [] = ClaculateThroughputDegrade (files[],
    DFSs[originalLoc],DFSs[destinationLoc])
15: migrateList[] = sort(Throuhput [])
16: for i = 0 to migrateList.size() do
17:     data = readFile(migrateList[i], DFSs[originalLoc])
18:     writeFile(data,DFSs[destinatioLoc])
19:     deleteFile(migrateList[i], DFSs[originalLoc])
20:     if (DFSs[orig].usage − DFSs[des].usage )< p_0 then
21:         return null
22:     end if
23: end for
```

in HybridFS can take the advantages of both GlusterFS and HDFS. When small size files come, the file placement function store them in GlusterFS. While large size files come, the files are stored in HDFS.

Next, we evaluate the impact of the large size file portion in a DFS to the throughputs of that DFS. We have generated 12 datasets in which the large size file portions are set as 0%, 5%, 10%, 20%, ..., to 100 %. Each dataset contains 1000 different size of files. Figure 5 shows the average read/write throughputs for each dataset tested on the designated DFSs. The average read/write throughput of a file on a DFS is defined as the average of its read throughput and write throughput on a DFS. From Figure 5, we have the following observation:

Observation 4: The average read/write throughput of HybridFS is the best when the large size file portion is less than

Authorized licensed use limited to: The Chinese University of Hong Kong CUHK(Shenzhen). Downloaded on November 11,2024 at 05:40:22 UTC from IEEE Xplore. Restrictions apply.

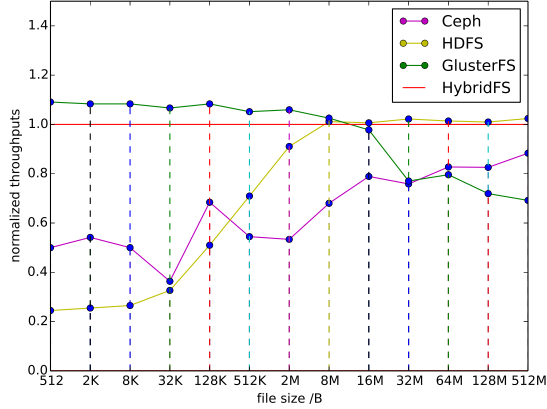

Fig. 4. The throughputs of designated DFSs with different file sizes

or equal to 54%. When the large size file portion is greater than 54%, the average read/write throughput of HybridFS is slightly less than that of HDFS. In general, the large size file portion in an application is in between 5% to 20%. Observation 4 indicates that HybridFS can deliver the best performance for most of applications compared to HDFS, Ceph, and GlusterFS.

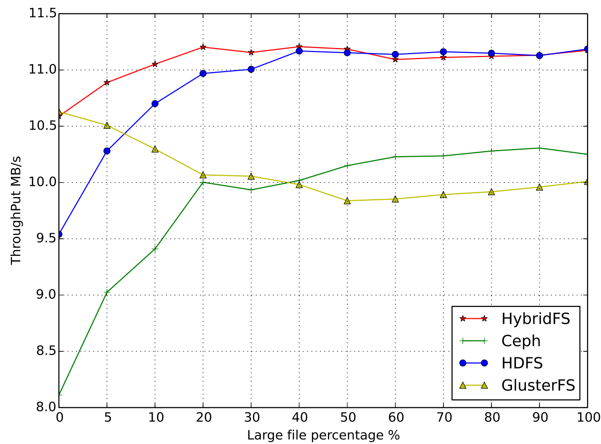

Fig. 5. The average read/write throughputs of 12 datasets with various portions of large size files on Ceph, HDFS, GlusterFS, and HybridFS.

B. The Impact of Partial Metadata Store Function on Small File Performance

To evaluate the performance of partial metadata store function for file sizes between a few bytes to a few hundred bytes, we have generated 100 files with file size = 1, 2, 3, ..., 500 Bytes, that is, the total number of files generated is 50000. Figure 6 shows the average time to perform read/write operations on these files. From Figure 6, we have the following observations:

Observation 5: With the partial metadata store function, HybridFS can deliver the best performance for file with size in between a few Bytes to a few hundred Bytes compared to HDFS, Ceph, and GlusterFS. Without the partial metadata store function, the GlusterFS will deliver the best performance among the designated DFSs.

For HybridFS without the partial metadata store function, all small files will be stored at GlusterFS. To get the location of a small file, an additional query to MMS is required before query the metadata of GlusterFS. Therefore, the performance of HybridFS without the partial metadata store function is a little worse than that of GlusterFS.

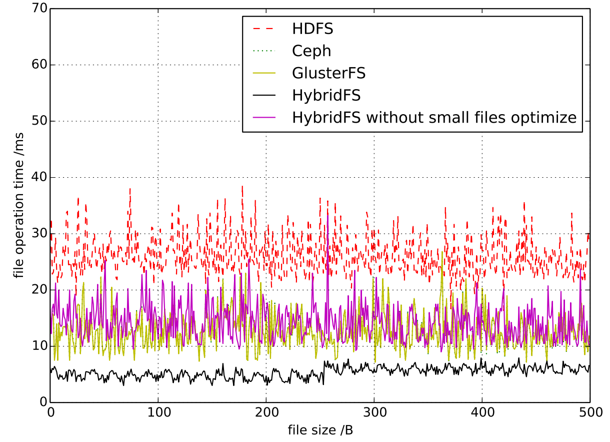

Fig. 6. The average time to perform read/write operations for files with file size ranging from 1 Byte to 500 Bytes on designated DFSs

C. The Impact of Dynamic File Migration Function on Balanced Storage Usage

To evaluate the impact of dynamic migration function on balanced storage usage, we need to decide the prediction functions for read/write throughputs on designated DFSs. We execute files with size = 0.5MB, 1MB, 5MB, 10MB,15MB, 20MB, 25MB, ..., 120MB on HDFS, Ceph and GlusterFS to get the curves of their read and write throughputs as shown in Figure 3. We apply the least-square method and the RMSE equation to determine the prediction functions. The result is shown in Table III.

To simulate the scenario of unbalanced storage usage of HybridFS, the storage capacities of HDFS, Ceph, and GlusterFS are all set to 100GB. File sizes are set in between 100 Bytes to 500MB. We randomly generate ten files every minute. For those generated files, the ratio of the number of files with file sizes less than 8M to the number of files with file size greater than or equal to 8MB is 4 : 1.

The next thing is to set the threshold of storage usage difference to determine when to launch the file migration process. Since the threshold depends on the characteristics of DFSs, we set the threshold as 10%, 20%,..., 90%. From the experimental tests, we found that when the threshold is less than or equal to 40%, the performance of HybridFS is guaranteed, that is, no performance degradation. However, the

Authorized licensed use limited to: The Chinese University of Hong Kong CUHK(Shenzhen). Downloaded on November 11,2024 at 05:40:22 UTC from IEEE Xplore. Restrictions apply.

TABLE III
DIFFERENT-ORDER REGRESSION MODELS

Curve	Fitting results of different curve
HDFS write curve	$y(k) = 10.39065 - 6.38257e^{-0.54163k}$
Ceph write curve	$y(k) = 8.79252 - 4.65085e^{-0.06894k}$
GlusterFS write curve	$y(k) = 8.43731 + 0.10894e^{-0.04518k}\cos(-38.07854k) - 1.89347e^{-0.04518k}\sin(-38.07854k)$ $+ 1.49443e^{-0.61613k}\cos(33.75146k) - 0.05625e^{-0.61613k}\sin(33.75146k)$
HDFS read curve	$y(k) = 11.0027 - 49.0537e^{-97.8321k} - 5.3826e^{-2.9596k}\cos(25.1327k) - 42.3298e^{-2.9596k}\sin(25.1327k)$
Ceph read curve	$y(k) = 11.128770 - 1.063236e^{-0.718258k}$
GlusterFS read curve	$y(k) = -0.0433 + 0.1108e^{0.00013k} - 6.2434e^{-4.3548k}\cos(0.000019k) + 17.2060e^{-4.3548k}\sin(0.000019k)$

overhead to migrate files is minimum when the threshold is set as 40% since the less the threshold, the more the migration times.

Figure 7 shows the storage usage and throughputs of HybridFS over a time period of 25 hours when the threshold is set to 40%. From Figure 7, at the beginning, files are stored either in HDFS or GlusterFS due to the characteristic of file placement function proposed in HybridFS. There has no files stored in Ceph. We also can see that the storage usage of HDFS grows faster than that of GlusterFS since large size of files are stored on HDFS. About 10 hours, the dynamic file migration daemon detects that the threshold is satisfied. The dynamic file migration function migrates some files from HDFS to Ceph. About 23 hours, the threshold is again satisfied. Some files are migrated from HDFS to Ceph and GlusterFS. The purple line shown at the top of Figure 7 is the throughput of HybridFS. From Figure 7, we can see that the throughput of HybridFS is stable.

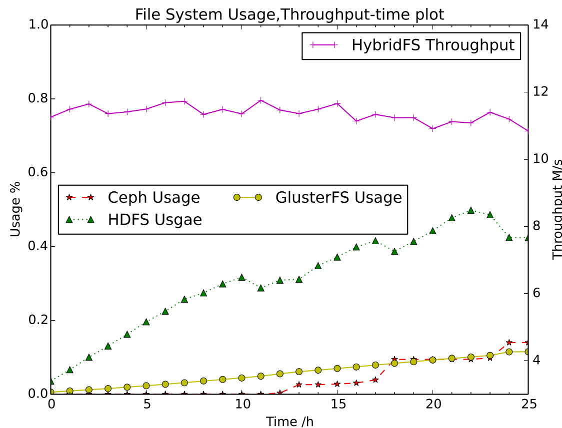

Fig. 7. The storage usage and throughputs of HybridFS over a time period of 25 hours when the threshold is set to 40%.

V. CONCLUSION

In this paper, we have proposed a hybrid distributed file system framework, HybridFS, which is composed of multiple DFSs. Three functions were designed to handle files. The file placement function is used to determine the proper DFS to store a given file. The partial metadata store function is used to store files with sizes ranging from a few bytes to a few hundred bytes. The dynamic file migration function is used to balance the storage usage of DFSs with throttling the system performance. The framework is generic and can be used for different DFSs combinations as long as the characteristics of files and DFSs can be derived. The experimental evaluation shows that HybridFS can deliver satisfactory performance for all big data applications with various file sizes.

To further enhance the capabilities of HybridFS, we have the following possible directions:

1. For the DFSs used in HybridFS, they are statically configured in the current design. In the future, we will add mechanism to insert and delete DFS from HybridFS dynamically. With this feature, the performance of big data applications can be further improved.

2. For the current design of dynamic file migration function, the threshold to determine when to launch the file migration process is based on a simulation in advance. In the future, we will derive a model based on the characteristics of DFSs used to predict the threshold. With the model, the threshold can be determined dynamically based on the current status of HybridFS to further improve the overall system performance.

ACKNOWLEDGEMENTS

This Work is supported by National Key Research & Development Program of China (2016YFB1000504), Natural Science Foundation of China (61433008, 61373145, 61572280, U1435216).

REFERENCES

[1] Sonali Aggarwal, Shashank Phadke, and Milind Bhandarkar. Characterization of hadoop jobs using unsupervised learning. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 748–753. IEEE, 2010.

[2] Jayanta Basak and Madhumita Bharde. Dynamic provisioning of storage workloads. In *29th Large Installation System Administration Conference (LISA15)*, pages 13–24, 2015.

[3] Dhruba Borthakur. Hdfs architecture guide. *HADOOP APACHE PROJECT http://hadoop. apache. org/common/docs/current/hdfs design. pdf*, page 39, 2008.

[4] Pavel Bzoch and Jiri Safarik. State of the art in distributed file systems: Increasing performance. In *Engineering of Computer Based Systems (ECBS-EERC), 2011 2nd Eastern European Regional Conference on the*, pages 153–154. IEEE, 2011.

[5] Benjamin Depardon, Gaël Le Mahec, and Cyril Séguin. Analysis of six distributed file systems. 2013.

[6] Bin Dong, Xiuqiao Li, Qimeng Wu, Limin Xiao, and Li Ruan. A dynamic and adaptive load balancing strategy for parallel file system with large-scale i/o servers. *Journal of Parallel and Distributed Computing*, 72(10):1254–1268, 2012.

Authorized licensed use limited to: The Chinese University of Hong Kong CUHK(Shenzhen). Downloaded on November 11,2024 at 05:40:22 UTC from IEEE Xplore. Restrictions apply.

[7] Bo Dong, Qinghua Zheng, Feng Tian, Kuo-Ming Chao, Nick Godwin, Tian Ma, and Haipeng Xu. Performance models and dynamic characteristics analysis for hdfs write and read operations: A systematic view. *Journal of Systems and Software*, 93:132–151, 2014.

[8] Giacinto Donvito, Giovanni Marzulli, and Domenico Diacono. Testing of several distributed file-systems (hdfs, ceph and glusterfs) for supporting the hep experiments analysis. In *Journal of Physics: Conference Series*, volume 513, page 042014. IOP Publishing, 2014.

[9] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *ACM SIGOPS operating systems review*, volume 37, pages 29–43. ACM, 2003.

[10] GlusterFS. https://www.gluster.org/docs-redirect/. 2016.

[11] Hung-Chang Hsiao, Hsueh-Yi Chung, Haiying Shen, and Yu-Chang Chao. Load rebalancing for distributed file systems in clouds. *IEEE transactions on parallel and distributed systems*, 24(5):951–962, 2013.

[12] Liu Jiang, Bing Li, and Meina Song. The optimization of hdfs based on small files. In *Broadband Network and Multimedia Technology (IC-BNMT), 2010 3rd IEEE International Conference on*, pages 912–915. IEEE, 2010.

[13] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145, 1995.

[14] Shining Li, Zhenhai Zhang, and Jiaqi Duan. An ensemble multi-label feature selection algorithm based on information entropy. *Int. Arab J. Inf. Technol.*, 11(4):379–386, 2014.

[15] Xiuqiao Li, Bin Dong, Limin Xiao, and Li Ruan. Performance optimization of small file i/o with adaptive migration strategy in cluster file system. In *High Performance Computing and Applications*, pages 242–249. Springer, 2010.

[16] Xuhui Liu, Jizhong Han, Yunqin Zhong, Chengde Han, and Xubin He. Implementing webgis on hadoop: A case study of improving small file i/o performance on hdfs. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–8. IEEE, 2009.

[17] Michael Mesnier, Eno Thereska, Gregory R Ganger, Daniel Ellard, and Margo Seltzer. File classification in self-* storage systems. In *Autonomic Computing, 2004. Proceedings. International Conference on*, pages 44–51. IEEE, 2004.

[18] Shuai Mu, Kang Chen, Pin Gao, Feng Ye, Yongwei Wu, and Weimin Zheng. μlibcloud: Providing high available and uniform accessing to multiple cloud storages. In *2012 ACM/IEEE 13th International Conference on Grid Computing*, pages 201–208. IEEE, 2012.

[19] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[20] Michael A Sevilla, Noah Watkins, Carlos Maltzahn, Ike Nassi, Scott A Brandt, Sage A Weil, Greg Farnum, and Sam Fineberg. Mantle: a programmable metadata load balancer for the ceph file system. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 21. ACM, 2015.

[21] Yu-Chuan Shen, Chao-Tung Yang, Shuo-Tsung Chen, and Wei-Hsun Cheng. Implementation of software-defined storage service with heterogeneous object storage technologies. In *Proceedings of the ASE BigData & SocialInformatics 2015*, page 3. ACM, 2015.

[22] Srinivasa Kini Shrikrishna Utpat, K. A. Dehamane. An optimized storing and accessing mechanism for small files on hdfs. *International Journal of Advanced Research in Computer Science and Software Engineering*, 5(1):673–677, 2015.

[23] Maomeng Su, Lei Zhang, Yongwei Wu, Kang Chen, and Keqin Li. Systematic data placement optimization in multi-cloud storage for complex requirements. *IEEE Transactions on Computers*, 65(6):1964–1977, 2016.

[24] Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 307–320. USENIX Association, 2006.

[25] Tom White. *Hadoop: The Definitive Guide, 2nd ed.* 2009.

[26] Lin Xu, Jiannong Cao, Yan Wang, Lei Yang, and Jing Li. Bejo: Behavior based job classification for resource consumption prediction in the cloud. In *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, pages 10–17. IEEE, 2014.

Authorized licensed use limited to: The Chinese University of Hong Kong CUHK(Shenzhen). Downloaded on November 11,2024 at 05:40:22 UTC from IEEE Xplore. Restrictions apply.