SANE: Semantic-Aware Namespace in Ultra-Large-Scale File Systems

Yu Hua, *Senior Member, IEEE*, Hong Jiang, *Senior Member, IEEE*, Yifeng Zhu, *Member, IEEE*, Dan Feng, *Member, IEEE*, and Lei Xu

Abstract—The explosive growth in data volume and complexity imposes great challenges for file systems. To address these challenges, an innovative namespace management scheme is in desperate need to provide both the ease and efficiency of data access. In almost all today's file systems, the namespace management is based on hierarchical directory trees. This tree-based namespace scheme is prone to severe performance bottlenecks and often fails to provide real-time response to complex data lookups. This paper proposes a Semantic-Aware Namespace scheme, called SANE, which provides dynamic and adaptive namespace management for ultra-large storage systems with billions of files. SANE introduces a new naming methodology based on the notion of semantic-aware per-file namespace, which exploits semantic correlations among files, to dynamically aggregate correlated files into small, flat but readily manageable groups to achieve fast and accurate lookups. SANE is implemented as a middleware in conventional file systems and works orthogonally with hierarchical directory trees. The semantic correlations and file groups identified in SANE can also be used to facilitate file prefetching and data de-duplication, among other system-level optimizations. Extensive trace-driven experiments on our prototype implementation validate the efficacy and efficiency of SANE.

Index Terms—File systems, storage systems, semantic awareness, namespace management

1 Introduction

Petabyte-, or Exabyte-scale data sets and Gigabit data streams are the frontiers of today's file systems [1]. Storage systems are facing great challenges in handling the deluge of data stemming from many data-intensive applications such as business transactions, scientific computing, social network webs, mobile applications, information visualization, and cloud computing. Approximately 800 Exabytes of data were created in 2009 alone [2]. According to a recent survey of 1,780 data center managers in 26 countries [3], over 36 percent of respondents faced two critical challenges: efficiently supporting a flood of emerging applications and handling the sharply increased data management complexity. This reflects a reality in which we are generating and storing much more data than ever and this trend continues at an accelerated pace. This data volume explosion has imposed great challenges to storage systems, particularly to the metadata management of file systems. For example, many systems are required to perform hundreds of thousands of metadata operations per second and the performance is severely restricted by the hierarchical directory-tree based metadata management scheme used in almost all file systems today [4].

The most important functions of namespace management are file identification and lookup. File system namespace as an information-organizing infrastructure is fundamental to system's quality of service such as performance, scalability, and ease of use. Almost all current file systems, unfortunately, are based on hierarchical directory trees. This namespace design has not been changed since it was invented more than 40 years ago [5]. As the data volume and complexity keep increasing rapidly, conventional namespace schemes based on hierarchical directory trees have exposed the following weaknesses.

Weakness 1: Limited system scalability. The directory-based management is effective only when similar documents or files have been stored in the same directory [4]. Although the directory size distribution has not significantly changed [6], the file system capacity has increased dramatically. This not only causes great inconvenience for file systems users, but also slows down data-intensive applications by generating more random accesses to underlying disks. In addition, since lookups are performed recursively starting from root directories, disks or servers serving requests to higher levels of the trees have a highly unbalanced share of the workloads, leading to a higher probability of becoming performance bottlenecks.

Weakness 2: Reliance on end-users to organize and lookup data. Locating a target file by manually navigating the directories through directory trees in a large system amounts to searching a needle in a haystack. As the directory tree becomes increasingly "fatter", it is equally difficult for users to instruct the file systems where a file should be stored and to find them quickly. When one does not know the full pathname of a file, slow exhaustive search over all directories is

- *Y. Hua and D. Feng are with the Wuhan National Lab for Optoelectronics, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, Hubei, China. E-mail: {csyhua, dfeng}@hust.edu.cn.*
- *H. Jiang and L. Xu are with the Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588-0150. E-mail: {jiang, lxu}@cse.unl.edu.*
- *Y. Zhu is with the Department of Electrical and Computer Engineering, University of Maine, Orono, ME 04469-5708. E-mail: zhu@eece.maine.edu.*

Manuscript received 27 Oct. 2012; revised 10 Mar. 2013; accepted 9 May 2013; date of publication 22 May 2013; date of current version 21 Mar. 2014. Recommended for acceptance by X.-H. Sun.
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/TPDS.2013.140

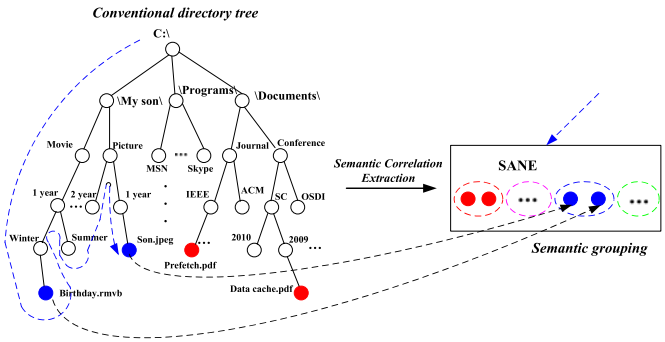


Fig. 1. SANE and hierarchical directory tree.

often resorted to. Such exhaustive search on a large system with billions of files takes a prohibitive amount of time. It is even more difficult to locate correlated files since users often cannot explicitly define mandatory search criteria in most file systems.

Weakness 3: Lack of metadata-semantics exploration. While it is difficult to manage massive data through a centralized hierarchical structure, research in both industry and academia has shown that, in most file systems, a small subset of file system's data serves a majority of data access requests [6], [7], [8], [9], [10]. Being able to identify this subset of frequently accessed data by semantic exploration is hence beneficial to system designers in their pursuit of system optimizations such as file prefetching and data deduplication. Conventional file systems have by and large ignored the semantic context in which a file is created and accessed during its lifetime.

While a rich body of research in the recent literature has attempted to overcome these weaknesses, such as Spyglass [7], Ceph [11], Glance [12], quFiles [13], DiFFS [14], SmartStore [15], Haystack [16] and Ursa Minor [17], these solutions are not comprehensive and still limited by the inherent weaknesses of the directory-tree naming scheme. Our design shares with them the similar goals of improving file organization and simplifying data management.

We propose a new namespace management scheme, called SANE, which provides a flat but small, manageable and efficient namespace for each file. In SANE, the notion of semantic-aware per-file namespace is proposed in which a file is represented by its semantic correlations to other files, instead of conventional static file names. Our goal is not to replace conventional directory-tree management that already has a large user base. Instead, we aim to provide another metadata overlay that is orthogonal to directory trees. SANE runs concurrently with the conventional file system that integrates it and takes over the responsibilities of file search and semantic file grouping from the file system when necessary. Moreover, SANE, while providing the same functionalities, makes use of a new naming scheme that only requires constant-scale complexity to identify and aggregate semantically correlated files. SANE extracts the semantic correlation information from a hierarchical tree. Fig. 1 illustrates the relationship and difference between SANE and the existing hierarchical directory tree. For instance, in order to serve a complex query, SANE only needs to check the small and flat namespace one time, thus avoiding a time-consuming search of brute-forced traversal over the entire hierarchical tree.

SANE is intended for an integration into modern file systems such as pNFS [18], PVFS [19], GFS [20], and HDFS [21]. Our goal in this research is to complement existing file systems and improve system performance. Our major contributions are summarized below.

First, addressing Weaknesses 1 and 2, SANE is designed to leverage semantic correlations residing in multi-dimensional attributes, rather than one-dimensional attributes such as pathnames, to represent a file. The metadata of files that are strongly correlated are automatically aggregated and then stored together in SANE. When a user performs a file lookup, SANE will also present the user files that are strongly correlated to this searched file, which constitute the semantic-aware per-file namespace of this file. This allows the user to access the correlated files easily without having to perform additional searches or directory tree navigations. In a distributed environment, this also improves the system performance since it improves the affinity: files that tend to be accessed together are placed on the adjacent servers. As a result, the operations on similar (i.e., semantically correlated) data take place in limited subsets of data without incurring extra overheads on the whole system, thus significantly improving the system scalability.

Second, addressing Weakness 3, SANE leverages locality sensitive hashing (LSH) [22] to automatically organize semantically correlated files without the involvement of end-users or applications. Our algorithm has very little performance overhead since LSH has a low complexity of probing constant-scale buckets. SANE represents each file based on its semantic correlations to other files. As the file system evolves, SANE can efficiently identify their changes to update the namespace by exploiting the file semantics. The semantics residing in files' correlation are obtained from multiple dimensions, rather than a single one, thus also allowing us to optimize the overall system design.

Third, SANE is implemented as a transparent middleware that can be deployed/embedded in most existing file systems without modifying the kernels or applications. SANE provides users with two auxiliary namespace views, i.e., default (conventional hierarchy) and customized (semantic-aware per-file representation). Both views hide the complex details of the physical representation of individual files, and export only a context-specific logical outlook of the data. Experimental results demonstrate that SANE efficiently supports query services for users, while facilitating system performance improvements, such as file prefetching and data de-duplication.

The rest of this paper is organized as follows. Section 2 presents the backgrounds and problem statement. Section 3 discusses the design and implementation details. Section 4 evaluates the performance. Section 5 presents the related work. We conclude our paper in Section 6.

2 Backgrounds and Problem Statement

2.1 Multi-Dimensional Attributes

Real-world applications demonstrate the wide existence of access locality that is helpful to identify semantic correlation. For instance, Filecules [8] examines a large set of real traces

and concludes that files can be classified into correlated groups since 6.5 percent of files account for 45 percent of I/O requests. Spyglass [7] reports that the locality ratios are below 1 percent in many traces, meaning that correlated files are contained in less than 1 percent of the directory space. A workload study on a large-scale file system [9] demonstrates that fewer than 1 percent clients issue 50 percent file requests and over 60 percent re-open operations occur within one minute. A recent study [6] shows that local write operations concentrate on 22 percent files in a five-year period. The existence of access locality facilitates the performance optimization in many computer system designs.

Selecting appropriate attributes is non-trivial due to two challenging constraints, i.e., the curse of dimensionality and dimensionality heterogeneity. First, when the dimensionality exceeds about 10, traversing the existing data structures based on space partitioning becomes slower than the brute-force linear-scan approach. This slowdown phenomenon is often called the "curse of dimensionality" [22]. We hence need to reduce the dimensionality in order to decrease operational complexity. Second, dimensionality heterogeneity, which means that two items that are close-by in one space might be far away in another space with a different dimensionality, is another great challenge. The data correlation is sensitive to the observation space selected. Two items that are correlated when observed in one attribute subset might be totally uncorrelated in another attribute subset.

Locality versus *affinity.* Extensive previous studies have proven that file system workloads have strong access locality, i.e., requested data are often temporally or spatially clustered together, as described briefly above. In this paper, we focus on the affinity of access and files. In fact, locality is a special affinity that describes temporal and spatial correlations among files. In this paper, we consider more generic affinity. Affinity in the context of this research refers to the semantic correlation derived from multi-dimensional file attributes, which include but are not limited to the temporal or spatial locality. For example, the typical attributes include path name, file name, file size, created time and modification time. In practice, simply exploring locality is insufficient in capturing affinity due to two reasons. First, physically closely located data can be semantically unrelated. For instance, the directories $c:\backslash windows$ and $c:\backslash users$ are functionally unrelated most of time. Accessing them through a shared ancestor directory will be neither helpful to users nor beneficial to the system performance. Second, affinity can in fact be implied by or embedded in the attributes that are not necessarily time or space oriented and are far from the temporal and spatial dimensions. Therefore, while in some cases locality may be part of the affinity and simple exploitation of locality may help scale up the performance to some extent, locality alone often falls short of overcoming the weaknesses.

Semantic correlations. Semantic correlations measure the affinity among files and we use correlations to estimate the likelihood that all files in a group are of great interest to a user or to be accessed together within a short period of time. We derive this measure from multiple attributes of files. To put things in perspective, linear brute-force search approaches use no correlation at all, which we call zero-dimensional correlation. Spatial-/temporal-locality based approaches, such as Spyglass [7] and SmartStore [15], use limited-dimensional correlations either in access time or reference space, which can be considered a special case of our proposed approach. The main benefit of measuring semantic correlations in multi-dimensional attribute space is that the affinity among files can be more accurately identified.

Semantic correlation can be exploited to optimize system performance. Metadata prefetching algorithms, Nexus [23] and FARMER [24], are proposed, in which both file access sequences and semantic attributes are considered in the evaluation of the correlation among files to improve file metadata prefetching performance. The probability of inter-file access is found to be up to 80 percent when considering four typical file system traces. Our preliminary results based on these and the *HP* [25], *MSN* [26], *EECS* [27] and Google [28] traces further show that exploiting semantic correlation of multi-dimensional attributes can help prune up to 99.9 percent search space [15].

Semantic correlation extends conventional temporal and spatial locality and can be quantitatively defined as follow. Assuming that there are $t \geq 1$ groups $\{G_i | 1 \leq i \leq t\}$, where the size of group G_i is denoted as $|G_i|$ and G_i contains files $f_j, (1 \leq j \leq |G_i|)$. Hence, the semantic correlation among the files of these groups can be measured by the minimum of $\sum_{i=1}^{t} \sum_{f_j \in G_i} (f_j - C_i)^2$ where C_i is the centroid of group G_i, i.e., the average values of multi-dimensional attributes. The value of $(f_j - C_i)^2$ represents the euclidean distance in the multi-dimensional attribute space. Since the computational costs for all attributes are unacceptably high in practice, we use a simple but fast tool, i.e., locality-sensitive hashing [22] to efficiently cluster semantically correlated files into groups, as detailed in Section 3.

2.2 Problem Statement

The problem we aim to solve in this paper concentrates on how to efficiently identify the t nearest neighbors of a given individual file. The answer to this question allows us to quickly represent the namespace of each file. Here t is an adjustable parameter that controls the size of the group that represents the access affinity and namespace of a file, and the correlation can be measured by the distance in a multi-dimensional space, such as euclidean distance.

For a given file, its per-file namespace can be formally defined by using semantic correlations as follows.

Definition 1. *The semantic-aware per-file namespace of file f consists of t files $(f_1, f_2, \ldots, f_t)$ that are the most strongly correlated with f based on p predefined semantics attributes, i.e., $(a_1, a_2, \ldots, a_p)$. The correlation degrees, as a quantitative representation of semantic correlation, are $(d_1, d_2, \ldots, d_t)$, respectively. The semantic-aware namespace of f is denoted by a t-tuple*

$$Namespace_t(f) = \{(f_1, d_1), (f_2, d_2), \ldots, (f_t, d_t)\},$$

where $f \notin Namespace_t(f)$, $d_i = 1 - \frac{E_i}{D}, (1 \leq i \leq t)$, D is a large constant, and E_i is euclidean distance between file f and file f_i in the p-dimensional attribute space.

The semantic-aware namespace on a per-file basis in SANE is a flat representation of a manageable size of t file

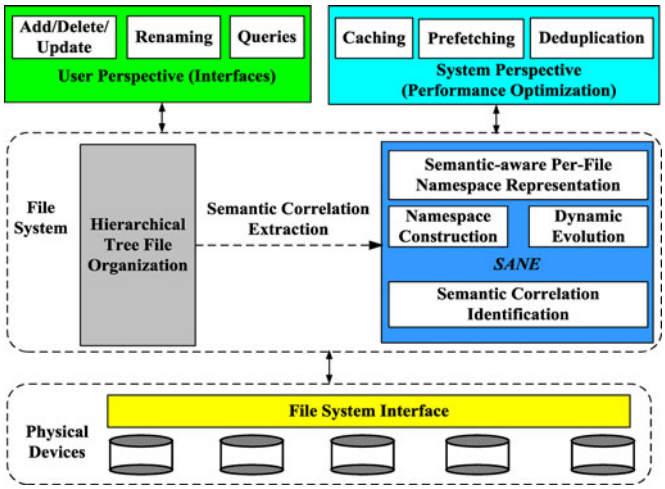


Fig. 2. SANE as a middleware in file systems.

members. The namespace construction entails identifying t nearest neighbors of a file in the multi-dimensional attribute space. The rationales behind this are two-fold. First of all, close neighbors, i.e., strongly semantically correlated files, should be arranged in the same or adjacent storage server nodes in a distributed file system or can be stored contiguously on disks in a centralized file system. Hence, performing a top-t query will quickly find correlated files belonging to the namespace of this file with a smaller amount of message exchanges in a distributed file system or fewer small random disk accesses in a centralized file system. We will discuss how to set an appropriate value for t, i.e., namespace size, in the following section. Second, the per-file namespace scheme in file systems can effectively provide a unique namespace for each file. Our design can avoid namespace collisions and the details are presented in Section 3.4.

3 Design and Implementation

We aim to exploit semantic correlations among files to define one flat, small, but accurate namespace for each file. The namespace of a file, which we call *semantic-aware perfile-namespace* and *namespace* for short in the rest of this paper, can be simply viewed as a single-level dynamic virtual directory that evolves with the changes in files attributes and consists of a set of files that are most semantically correlated to this file. However, different from directory trees, per-file namespace is not hierarchical at all, i.e., one per-file namespace cannot be a child of another per-file namespace. A per-file namespace of a file also differs from a directory in that this per-file namespace does not include this file itself, as we will discuss later in details.

3.1 An Architectural Overview

To illustrate how SANE works, we briefly describe its overall architecture. SANE explores semantic correlations residing in files to build the namespace representation that can accurately identify a file and track the evolution of file attributes. SANE includes three key function modules, semantic correlation identification (SCI), namespace construction (NC) and dynamic evolution (DE), as shown in Fig. 2. To quickly identify correlated files, the SCI module performs the locality-sensitive-hashing computation based on multiple file attributes. Different compositions of attributes will produce semantic groups with different accuracies. However, it is non-trivial to select the optimal set of attributes that can most accurately define file correlations and best match access patterns. After the semantic correlations are identified, the NC module aggregates semantically correlated files into groups by performing nearest-neighbor searches for each file. This process determines which files belong to the namespace of a given file. The namespace of each file is represented as a t-tuple vector, consisting of the t most correlated files and their correlation degrees to this file. Finally, since the attributes' values of files and their correlations may change over time, DE helps it accurately adapt to such changes and makes speedy namespace updates. SANE exploits dynamically evolving correlations to create an accurate semantic-aware namespace in a very large-scale file system with a small performance overhead.

From the viewpoints of both end-users and file systems, SANE offers a transparent and context-aware abstraction to serve user requests and improve system performance. Specifically, for end-users, a customized flat and small namespace allows them to quickly navigate and identify target data files. A renaming operation is interpreted as a membership change to the t-tuple file set that constitutes a file's namespace. SANE can support three types of queries, namely, point, range and top-k queries. These small file sets that represent the namespaces of individual files differ from directories in conventional file systems in that, for the same file, a semantic-aware per-file namespace is a dynamic logic view to a set of files and it changes over time based on semantic contexts, while a directory always returns the same logic view to a fixed set of files. Furthermore, in systems with hierarchical directories, users and applications need to be aware of directory path to locate data. In contrast, SANE is transparent to users and applications and exposes a semantic-aware per-file namespace for a given file. On the other hand, for file systems, the semantic-aware namespace in SANE contains correlated files to facilitate efficient file caching and prefetching and data dedulplication, which is conducive to the overall performance improvement.

3.2 Semantic Correlation Identification

SANE uses locality sensitive hashing [22] to identify semantic correlations. LSH has the advantages of both locality preservation and fast identification.

LSH. We briefly introduce LSH and explain how it is used in fast semantic identification. LSH is an efficient tool that maps similar items into the same hash buckets.

Definition 2. *Given a distance R, approximation ratio $c > 1$, and two probability values P_1 and P_2 such that $1 > P_1 > P_2 > 0$, a function $h(.)$ is said to be (R, cR, P_1, P_2) locality sensitive for distance function $\|,\|$ if for points u_1 and u_2, it satisfies both conditions below:*

- *If $\|u_1, u_2\| \leq R$, then $Pr[h(u_1) = h(u_2)] \geq P_1$,*
- *If $\|u_1, u_2\| > cR$, then $Pr[h(u_1) = h(u_2)] \leq P_2$.*

In LSH, items close to each other will have a higher probability of colliding than items that are far apart [29]. Specifically, the closeness depends on the R value that can be obtained empirically by sampling real-world data

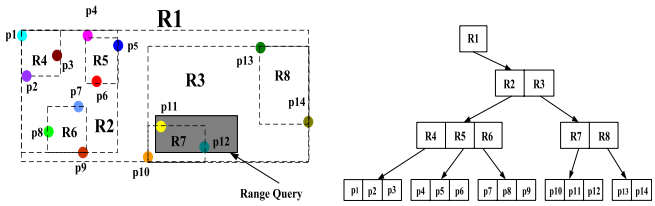
sets [30]. The first condition above shows that, for a given R, two close-by files that have a distance smaller than R will be hashed into the same bucket of a hash table with a high probability greater than or equal to P_1. The second condition states that two irrelevant files can always be hashed into the same bucket with a low probability equal to or smaller than P_2.

3.3 Data Storage Structure

LSH works well in identifying correlated data but suffers from the space-overhead problem as the amount of data increases. Since LSH requires many hash tables to maintain correlated data, the space overhead becomes a potential performance bottleneck. When dealing with massive amounts of data, data structure can easily overflow the main memory, leading to slow hard disk accesses and severe performance degradations. Furthermore, while the form of hash tables works for point-based queries (e.g., point and top-k queries), it may not efficiently support range queries that must obtain queried results within given intervals when the hash table fails to maintain the interval information.

We make use of the R-tree [31] structure to replace the original hash tables, store the correlated data, and represent their multi-dimensional attributes in the R-tree nodes. The *root* node (e.g., R1) represents domain ranges of all possible attributes. Let N be the maximum number of children of a node. Each *internal* node can contain r ($\frac{N}{2} \leq r \leq N$) child nodes. We set a lower bound on r to prevent tree degeneration and to ensure an efficient storage utilization. Whenever the number of children drops below r, the node will be deleted and its children will be re-distributed among sibling nodes. The upper bound N can guarantee that each tree node in fact can be stored exactly on one disk page. Each internal node contains entries in the form of $(I, Pointer)$ where $I = (I_0, I_1, \ldots, I_{p-1})$ is a p-dimensional bounding box, representing a minimumbounding rectangle (MBR). I_i is a bounded interval, which can cover items in the ith dimensional space. $Pointer$ is the address of a child node.

The flat namespace as a proper middleware will not become the performance bottleneck. After identifying correlated files by using LSH, we then organize all groups of closely correlated files into R-trees that are deployed in multiple distributed metadata servers. An R-tree is a data structure organized in aheight-balanced tree that is often used to represent and indexspatial data. An R-tree can split data space into hierarchicallynested bounding boxes that may contain several data entitieswithin the bounding box. It can efficiently support point, rangeand top-k queries by maintaining index records in its leaf nodes. An index record is a reference pointer to data. R-treesuse solid minimum bounding rectangles, i.e., bounding boxes, to indicate the queried regions. An MBR ineach dimension denotes an interval of all enclosed data witha lower and an upper bound. In our design, we exploit its special capability for supporting query services by modifying its structure to store the semantically aggregated files. R-tree isa completely dynamic index structure that is able to efficiently support data updates and provide efficient query services by visiting only a small number of nodes in a spatial search. Thusin our design, we use R-tree to construct the namespace.



(a) The data representation in geometric space. (b) The data storage structure.

Fig. 3. R-tree based representation.

As shown in Fig. 3, we first identify the correlated items via LSH based hashing computation. The correlated items are represented in the geometric space as shown in Fig. 3a. Fig. 3b further exhibits the R-tree based data structure to store the correlated items. LSH identifies correlated data items that are further represented in geometric space via minimum bounded rectangles that correspond to the groups. MBRs can be further aggregated iteratively until the R-tree root node. Each non-leaf node in an R-tree contains the ranges of multi-dimensional attributes of stored files as well as the pointers to their child nodes. The leaf nodes maintain the pointers to the actual files. Moreover, the namespace of each file can be built by selecting the members from the groups that this file belongs to. For a range query, it contains the query requests that are also represented as rectangles in the geometric space. The covered data can be fast identified and considered as query results.

3.4 Namespace Construction

SANE offers a scalable way to construct semantic-aware per-file namespaces for the file system. SANE can provide the constructed namespaces of files in any storage area, such as portions of the main memory, and/or portions of the secondary storage of SSD or HDD. Here, take the main memory for an example, SANE is initialized by first carrying out LSH-based hash computation to cluster into groups files. These files are semantically correlated with the files already in the main memory of the file system, in which SANE is installed. SANE then organizes these groups into an R-tree structure. A nearest-neighbor query over the R-tree can identify the member files of the namespace of an individual file. This process repeats when a new file is accessed and loaded into the main memory. When the main memory is full, the namespace of a file will be replaced and flushed to the secondary memory based on a proper replacement policy, say, LRU. In other words, the main memory stores the per-file namespaces of the "hot" or "popular" files at any given time while "cold" files' namespaces are stored in the secondary memory.

We use the LSH-based R-trees described above to build the semantic aware per-file namespace. Specifically, for each file, its namespace is derived from the results of a top-t query that identifies the t nearest neighbors in the attribute space. These most closely correlated neighbors constitute the namespace of this file.

The size of the per-file namespace depends on the parameter t. If t is too small, i.e., having a very small number of members, it is difficult to differentiate files. But a very large t often involves some files that might not be strongly

correlated, resulting in a decrease in the semantic correlation in a namespace and a potentially higher cost for namespace update. Therefore, we need to strike a good balance between the differentiated representation and the semantic correlation guarantee. In SANE, we determine an appropriate value for t by maximizing the mean and standard deviation (MSD), defined as $MSD(f) = \overline{d} + \alpha\sqrt{\frac{\sum_{i=1}^{t}(d_i-\overline{d})^2}{t}}$, where $\overline{d} = \frac{\sum_{i=1}^{t} d_i}{t}$ and α is the correlation factor. The correlation factor is obtained from history sample records and can be adjusted according to specific requirements in real-world applications (e.g., the traces used in Section 4 for performance evaluation). Our rationale behind using MSD is that the mean of all correlated degrees faithfully describes the correlation between a given file and all files in its namespace. The standard deviation allows the namespace to select differentiated member files to guarantee the unique representation. We further make a suitable assumption that the namespace of each file does not contain the file itself.

MSD quantitatively controls the construction of a semantic-aware per-file namespace in an iterative manner. Initially, we start with one file ($t = 1$), i.e., the nearest neighbor, and calculate the initial $MSD(f)$. If the addition of the next most closely correlated file to the namespace can increase the value of $MSD(f)$, this file is then considered a member of file f's namespace and t is increased by 1 and $MSD(f)$ is updated accordingly. This process repeats until the addition of the next most closely correlated file results in a decrease in $MSD(f)$.

We use the correlation degree d_i as a metric to evaluate the correlation between two files. The metric d_i is the distance between two files in the attribute space. This metric can also help differentiate the file names and guarantee the uniqueness of file representation. For instance, assuming that both files A and B are correlated with file C, SANE thus considers file C as a member of namespaces $Namespace_t(A)$ and $Namespace_t(B)$, which may potentially produce the same naming representation. When using the correlation degree, e.g., $(C, 0.7)$ and $(C, 0.5)$ respectively in $Namespace_t(A)$ and $Namespace_t(B)$, we can easily differentiate their namespaces due to different correlation degrees. In the worst case, it might happen that the representations of two files are exactly the same, i.e., they have the same file members and correlation degrees. Although it occurs rarely, we solve this representation collision by increasing the namespace size until we obtain a unique representation.

3.5 Dynamic Evolution

SANE leverages a two-set representation of namespace for speedy update. The semantic naming scheme adapts to the dynamic evolution of file attributes and correlations, which is one of the most salient features distinguishing SANE from most existing tree-based schemes.

In SANE, we use a two-set design to achieve fast update on staleness and reduce the overhead of maintaining information consistency. For each file f, we maintain two membership sets, i.e., the set $Namespace_t(f)$ that contains the t files most semantically correlated to file f, and the set $Member(f)$ whose elements represent

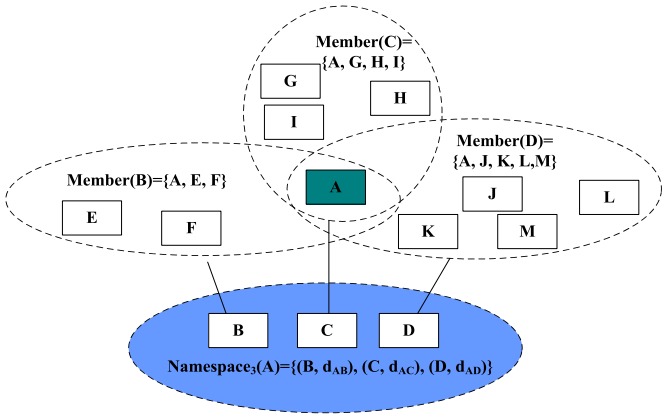


Fig. 4. Two-set representation for speedy update.

all files whose namespaces contain file f. For example, set $Namespace_3(A) = \{(B, d_{AB}), (C, d_{AC}), (D, d_{AD})\}$ keeps the namespace member files B, C and D of file A. On the other hand, file B is also a member of other files' namespaces (say, E and F), thus, $Member(B) = \{A, E, F\}$. When the attributes' values of f change, we first execute a new top-t query to re-build its namespace by finding the t nearest neighbors. These new neighbors form the updated $Namespace_t(f)$ set. File f further conveys its new attribute values to the members of set $Member(f)$ to update their namespaces.

One benefit of using the two-set representation is to significantly reduce the complexity of the "*rename*" operation. A rename operation will likely change the namespace representation of a file to a new one, in which the key issue is how to guarantee the uniqueness of a new representation without a brute-force checking over the entire file system. With the aid of the two-set representation, a rename operation can be performed with a small performance overhead.

Specifically, we allow the verification for uniqueness to occur in the files shared by all "Member" sets of the renamed file. For instance, as shown in Fig. 4, when file A is renamed, we execute a simple intersection operation upon the three "Member" sets, i.e., $Member(B), Member(C), Member(D)$. If the intersection results in only file A, the uniqueness is guaranteed since no other file's namespace simultaneously contains files B, C, D. Otherwise, we must check whether the namespaces of files other than A in the intersection of the three "Member" sets contain the same member files and correlation degrees as the namespace of file A. If another file in the intersection has an identical namespace as the renamed file A, then we re-compute semantic correlations for A to generate another representation of A by increasing the namespace size. In the unlikely case that the newly computed namespace of A collides with another file, this process repeats until a unique namespace for the renamed file A is obtained as guaranteed (see Lemma 1). The simple intersection operation reduces the number of files to be checked. In addition, since two files being the same is defined as their respective namespaces having identical member files and correlation degrees, the probability of such a collision and re-computation to generate a new filename is extremely small.

A renamed file is likely a member of other files namespaces. After a file is renamed, it may potentially introduce further updates on its correlated files represented in its "Member" set. Specifically, we examine whether the new correlation degrees between the renamed file and its correlated files become larger than some pre-defined threshold. The threshold range is between 0 and 1. If it is 0, all files must be updated no matter what the correlation degree is. On the other hand, if it is 1, only the renamed file itself needs to be updated due to the guarantee of uniqueness in SANE. In practice, by leveraging an accurate pre-defined quantitative threshold of semantic correlation (e.g., through empirical analysis), the renaming of a file typically only leads to the updates of a limited number of correlated files. The update overhead due to the rename operations is shown indeed to be very small by our experimental evaluation.

4 Performance Evaluation

4.1 Experimental Setup

Platform. We have implemented a prototype of SANE in Linux kernel 2.6.28 and performed experiments on a cluster of 80 server nodes, each with Intel Core 2 Duo CPU and 2 GB memory. An RPC-based interface to WAFL (Write Anywhere File Layout) [32], [33] gathers the dynamic changes of file attributes, driven by our snapshot-based traces in the performance evaluation. SANE use three metrics, namely, *cost-effectiveness, searchability* and *scalability*. All experiments have taken into account the dynamic evolution of file systems, such as file creations and deletions. In addition, the user interfaces of SANE for namespace representation, renaming and query service are also implemented in our prototype. We design and implement most modules at the user space level so that our prototype can run on many existing systems. The prototype has implemented the three basic function components of SANE discussed in the previous section, namely, LSH-based semantic correlation identification, namespace construction, and dynamic evolution of attributes and correlations.

Intensified Traces. We use four representative traces, of which three collected from the industry and one from the academia. These traces include HP file system trace [25], MSN trace [26], EECS NFS server (EECS) trace at Harvard [27] and Google clusters trace [28], which drive the SANE performance evaluation. Moreover, for the sizes of these traces, Google cluster contains 75 five-minute reporting intervals. There are a total of 3,535,029 observations, 9,218 unique jobs and 176,580 unique tasks. HP contains 94.7 million requests for a total of 4 million files from 32 users. MSN has 1.25 million files and records 4.47 million operations, in which there are 3.3 million read and 1.17 million write operations. EECS contains 4.44 million operations. The number and size of read operations are respectively 0.46 million and 5.1 GB. Those of write operations are respectively 0.667 million and 9.1 GB. Due to their relatively small sizes, we use a scaled-up method to intensify them.

In order to emulate the I/O behaviors of large-scale file systems for which no realistic traces are publicly available, we scaled up the existing I/O traces of current storage systems both spatially and temporally. This method has been successfully used in Glance [12] and SmartStore [15]. Specifically, a trace is first decomposed into sub-traces. We then add a unique sub-trace ID to all files to intentionally increase the working set. The start times of all sub-traces are set to zero so that they are replayed concurrently. The chronological order among all requests within a sub-trace is faithfully preserved. The combined trace contains the same histogram of file system calls as the original one but represents a heavier workload (higher intensity). The number of sub-traces replayed concurrently is denoted as *Trace Intensifying Factor* (TIF) and in our experiments the default TIF value is 400. Moreover, the multi-dimensional attributes chosen for this evaluation are faithfully extracted from the original traces. In addition, in order to obtain reasonable R values for the LSH computation (Section 3.2) in our experiments, we use the sampling method, which has been verified through practical applications [29], [34]. We determine the R values to be 1,200, 800, 1,000 and 950, respectively for the intensified *HP*, *MSN*, *EECS* and Google traces. In fact, the SANE system can be dynamically configured according to the requirements of users or systems, such as query accuracy and latency, available space, bandwidth and computing resources.

All I/O requests in the intensified file system traces are issued from client machines to server machines. Both clients and servers use multiple threads to exchange messages and data via TCP/IP. IP encapsulation technique facilitates fast addressing in the networks by encapsulating an IP header with an outer IP header for tunneling. It helps forward the query requests among multiple servers. We repeat each experiments 30 runs to validate the results according to the evaluation guidelines of file and storage systems [35].

In our performance evaluation, we intentionally choose specific comparison schemes in order to make the comparison relevant and fair. We compare SANE with the target schemes only in the relevant aspects such as namespace construction, query performance for users, and file prefetching and data deduplication for systems. To evaluate *cost-effectiveness*, we choose to compare SANE with the Ext4 file system (Linux kernel 2.6.28) that serves as a classic representative of the conventional namespace schemes based on hierarchical directory trees. In order to support directory indexing, Ext4 examines a hashed B-tree that uses a hash table of the filenames. For *Searchability*, we select some state-of-the-art comparable systems, including Spyglass [7] and SmartStore [15]. Both systems support various types of queries in file systems with competitive performance. Note that since there is no open source code available for Spyglass, we implemented its main components, such as the crawler, multiple partitions and versions, and K-D tree, according to the descriptions presented in [7].

In the experiments, for a given file f, SANE first chooses its nearest neighbor file ($t = 1$) as the member of its namespace. We then obtain the value of $MSD(f)$. If the value of $MSD(f)$ increases after adding its next most closely correlated file to the namespace, the file is considered a member of file f's namespace and t is increased by 1, while $MSD(f)$ is updated accordingly. Otherwise, the namespace construction finishes. The bounds (minimum and maximum) and average values of t in four traces are respectively (min-max: 22-121; average: 31.6) in MSN, (min-max: 11-31; average:

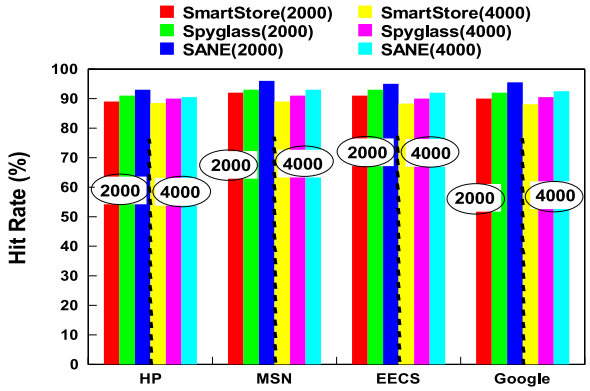


Fig. 5. Average hit rate for point query.

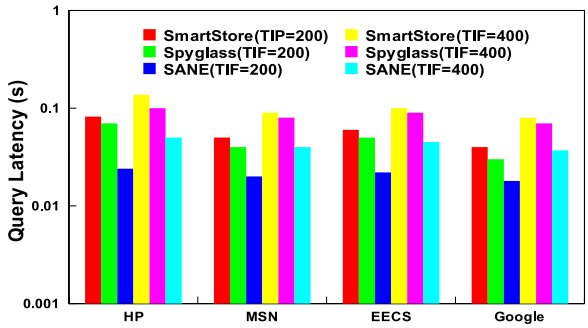


Fig. 6. Point query latency.

15.2) in EECS, (min-max: 17-53; average: 23.8) in Google and (min-max: 27-172; average: 38.4) in HP.

In general, filename-based point query is very popular in most file system workloads. There are no file system I/O traces for both point and complex queries (range and top-k) requests. In order to address this issue, we leverage a synthetic approach to generating not only point query, but also complex queries within the multi-dimensional attribute space. The basic idea is to statistically generate random queries in a multi-dimensional space. We study the file static attributes and behavioral attributes from the available I/O traces. For example, a point query in the form of (17:50, 85.2, 36.5) represents a search for the files that are closest to the description of a file that is last revised at time 17:50, with the amounts of "read" and "write" data being approximately 85.2 and 36.5 MB. Moreover, a range query aiming to find all the files that were revised between time 8:20 to 10:50, with the amount of "read" data ranging from 22 to 40 MB, and the amount of "write" data ranging from 7 to 12 MB, can be represented by two points in a three-dimensional attribute space, i.e., (8:20, 22, 7) and (10:50, 40, 12). Similarly, a top-k query in the form of (9:30, 17.2, 75.8, 5) represents a search for the top-five files that are closest to the description of a file that is last revised at time 9:30, with the amounts of "read" and "write" data being approximately 17.2 and 75.8 MB, respectively.

In addition, the semantic correlations identified in SANE create new opportunities for system designers to implement system optimizations from a totally new perspective. In this paper, we take data deduplication to illustrate potential benefits of SANE from a system perspective. We have leveraged semantic correlations identified by SANE to optimize system function of deduplication.

4.2 Results and Discussions

4.2.1 Searchability

We compare SANE with Spyglass [7] and SmartStore [15] in terms of accuracy and latency of point and complex queries. Note that both Spyglass and SmartStore can obtain exact-matching results by using brute-force-like approaches and increasing the amount of data that must be read from the disk. Here, hit rates in Spyglass and SmartStore represent cache hits of the accessed partitions in Spyglass and semantic groups in SmartStore respectively.

Query Accuracy. Fig. 5 shows the hit rates for the 2,000 and 4,000 point query requests. The hit rate of SANE is 93.7, 95.2, 94.6, and 94.8 percent, respectively for the HP, MSN, EECS and Google traces, visibly outperforming Spyglass (90.5, 92.3, 91.9 and 90.2 percent) and SmartStore (89.6, 91.1, 90.2 and 89.3 percent). The main reasons behind SANE's superiority to SmartStore and Spyglass are twofold. First, the former leverages the LSH functions that can significantly mitigate the adverse impact of stale information. Second, SANE's two-set design behind its semantic-aware namespace makes it possible to accurately and timely search updated results.

We adopt the "recall" metric from the field of information retrieval to measure the quality of complex-queries. For a given query q, we denote $T(q)$ the ideal set of k nearest objects and $A(q)$ the actual neighbors reported by SANE. We define $recall = \frac{|T(q) \cap A(q)|}{T(q)}$. Table 1 presents the *recall* measures of range and top-k queries in SANE. The experimental results show that the query results returned by SANE are reasonably accurate.

Query Latency. Fig. 6 compares the latencies of point-queries among SmartStore, Spyglass and SANE. The experimental results show that SANE outperforms SmartStore, by up to 58.2, 52.5, 54.7 and 49.6 percent, and Spyglass, by up to 32.8, 26.7, 28.2 and 24.6 percent, under the HP, MSN, EECS and Google traces respectively. This again is attributed to the fact that SANE uses the fast LSH-based hash computation while SmartStore uses the slow matrix-based LSI tool and Spyglass depends on subtree partitions in a hierarchical tree structure. Table 2 compares the latency measures of range and top-k queries of these three schemes. For complex queries, SANE consistently and significantly outperforms SmartStore and Spyglass by at least one order of magnitude.

4.2.2 Scalability

We examine the system scalability by measuring the average latencies of query and update requests as well as the number of required network messages as a function of the system size. The results are shown in Fig. 7.

We observe from Fig. 7a that the latency measure scales steadily and smoothly as the number of server nodes

TABLE 1
Accuracy of 4,000 Range and 4,000 Top-k ($k = 7$) Queries

	HP			MSN			EECS			Google		
TIF	150	350	450	150	350	450	150	350	450	150	350	450
Range	88.5	87.8	86.7	92.5	91.6	90.3	93.7	92.8	91.6	95.4	94.2	93.7
Top-k	95.7	93.4	92.6	97.1	95.9	94.2	97.7	95.8	94.2	96.2	95.6	94.3

TABLE 2
Range & Top-k (k = 7) Latency (Seconds)

Traces	TIF	Range Query			Top-k Query		
		SANE	SmartS.	Spyglass	SANE	SmartS.	Spyglass
HP	150	0.11	4.12	2.87	0.26	4.79	3.02
	350	0.35	10.61	7.62	0.82	12.37	9.36
	450	0.62	21.57	12.86	1.57	29.83	18.62
MSN	150	0.08	3.06	2.17	0.15	3.65	2.39
	350	0.27	8.52	5.71	0.62	11.73	6.82
	450	0.46	18.61	11.72	1.26	25.83	14.95
EECS	150	0.06	2.76	1.26	0.11	3.02	2.18
	350	0.21	7.58	4.32	0.47	9.52	5.69
	450	0.39	16.47	11.72	0.51	20.86	15.81
Google	150	0.05	2.48	1.17	0.09	2.65	2.08
	350	0.18	7.26	4.15	0.32	8.61	4.92
	450	0.32	15.81	10.28	0.42	17.62	12.75

increases from 10 to 80. SANE only needs to carry out simple hashing computation to accurately find the queried results that are placed together in a small and flat search space. Therefore, the upward scaling of system size has very limited impact on the organization of correlated files, thus resulting in strong system scalability.

Fig. 7b shows the latency of the update operation introduced in Section 3.5, indicating a near linear scaling. We issue a total of 1 million update requests and measure the average latency of each request under different system scales. Fig. 7c shows that the number of messages required for query services also scales reasonably with the number of server nodes. Since query operations run within one or a very small number of correlated file tuples, SANE avoids probing any irrelevant nodes and hence reduces the network overhead.

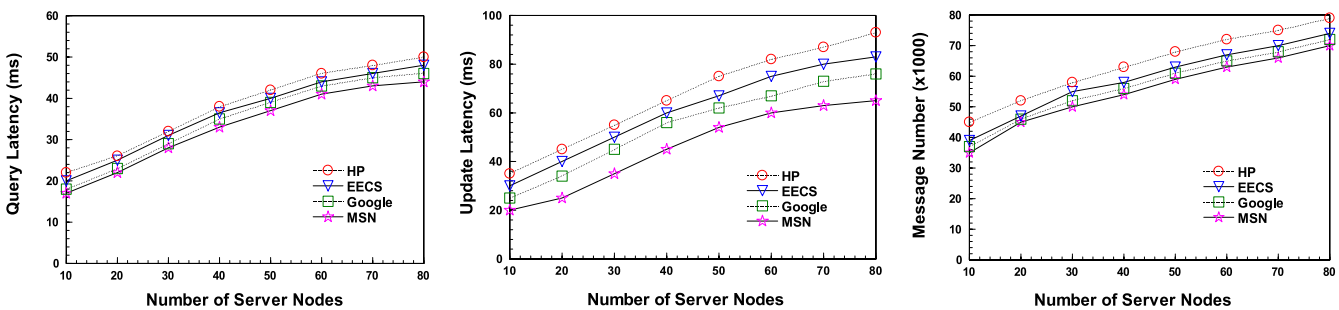
Projecting the scalability trends in an ultra-large scale system is difficult, if not impossible, to implement. In this study, we conduct simulations by exponentially scaling the number of server nodes from 100 up to 1,000. The simulation results in terms of latencies of query and update requests show that SANE can maintain near-linear scalability as the system scales up exponentially. We recognize that these simulations running under ideal conditions may overestimate the scalability of SANE since in the simulations we might have underestimated or ignored some potential bottlenecks, such as network bandwidth. However, the experimental results do at least show potentials of the proposed schemes in the future exascale systems.

5 Related Work

It is worth noting that the essential difference in SANE from existing work is its flat, rather than hierarchical, namespace for data-intensive file systems. Semantic file system (SFS) [36] is one of the first file systems that extend the traditional file system hierarchies by allowing users to search customized file attributes. SFS creates virtual directories based on demand. quFiles [13] provides a simple and unified view of different copies of files that are optimized for different access contexts, such as network bandwidth. SANE uses a new approach that exploits semantic correlations among files to create a dynamic per-file namespace to speed up file lookups when full pathnames are not available. Our approach differs from SFS and quFiles in that we take into consideration the semantic context implicitly and explicitly represented in file metadata when serving complex queries. Our approach is particularly helpful in avoiding brute-force search, which is time-prohibitive in large file systems.

Hadoop [21] has emerged to be a popular platform for large-scale data analysis but its namespace management suffers from the single-name-node limitation. The name-node stores the entire file system namespace in the main memory and can become a performance bottleneck, thus limiting the system scalability. In order to overcome the limitation of Hadoop Distributed File System (HDFS), Ceph [11] and its demonstration system [37] use dynamic subtree partition to avoid metadata-access hot spots and support filename-based query. Google file system (GFS) [20] logically represents its namespace as a lookup table mapping full pathnames to metadata. Although using a single master makes the overall metadata design simple and easily implementable, the single master can become a potential performance bottleneck and single point of failure. Haystack [16], used in Facebook, tries to avoid disk operations when accessing metadata by leveraging network attached storage appliances over NFS, and thus performs all metadata lookups in the main memory. Unlike SANE, these systems still inherit many of the innate features of the conventional hierarchical directory-tree methodology, thus limiting the scalability and functionality for large-scale file systems.

In order to handle the scalability problem of file system directories, GIGA+ [38] proposed a POSIX-compliant scalable directory design to efficiently support hundreds of thousands of concurrent mutations per second, in particular in terms of file creations. An extendible hashing-based



(a) Query Latency. (b) Update Latency. (c) Number of Messages.

Fig. 7. Study of system scalability based on prototype implementation.

method [39] is used to dynamically partition each directory to support metadata management for a trillion files. Moreover, with a goal to scale metadata throughput with the addition of metadata servers, the Ursa Minor distributed storage system [17] handles metadata operations on items stored in different metadata servers by consistently and atomically updating these items. Dynamic subtree partition [40] offers adaptive management for hierarchical metadata workloads that evolve over time. In SANE, our focus is not on how to store a large number of files within a directory. Instead, we aim to design a new approach that helps quickly locate target files in a file system with potentially billions or trillions of files.

Among searchable file systems, Spyglass [7] exploits the locality of file namespace and skewed distribution of metadata to map the namespace hierarchy into a multi-dimensional K-D tree and uses multilevel versioning and partitioning to maintain consistency. Glance [12], a just-in-time sampling-based system, can provide accurate answers for aggregate and top-k queries without prior knowledge. SmartStore [15] uses the latent semantic indexing (LSI) tool [41], [42] to aggregate semantically correlated files into groups and support complex queries. SANE improves the performance of the query functionalities significantly and provides real-time responses to metadata queries in semantic-aware namespaces.

6 Conclusion and Future Work

This paper proposes a new namespace management scheme, called SANE, that exploits semantic correlations among files to create a flat, small, and accurate semantic-aware namespace for each file. The per-file namespace is a flat structure without an internal hierarchy. For a given file, its namespace consists of a certain number of the most closely correlated files. We design an efficient method to identify semantic correlations among files by using a simple and fast LSH-based lookup. For each lookup operation, SANE cost-effectively presents users' files that might be of interests. We have implemented SANE as a middleware that can run on top of most existing file systems, orthogonally to directory trees, to facilitate file lookups. In addition, the semantic correlation accurately identified in SANE can be used to improve some system functions, such as data deduplication and file prefetching. SANE is a valuable tool for both system developers and users. We intend to release it for public use in the near future.

Acknowledgments

This work was supported in part by National Natural Science Foundation of China (NSFC) under Grant 61173043; National Basic Research 973 Program of China under Grant 2011CB302301; NSFC under Grant 61025008, 61232004; Fundamental Research Funds for the central universities, HUST, under grant 2012QN098; US National Science Foundation (NSF) under Grants NSF-IIS-0916859, NSF-CCF-0937993, NSF-CNS-1016609, NSF-CNS-1116606, NSF IIS 091663, CNS 1117032, EAR 1027809, CCF 0937988, CCF 0621493, and EPS 0904155. Part of the work was done while Yu Hua was at University of Nebraska-Lincoln. The authors greatly appreciate anonymous reviewers for constructive comments.

References

[1] I. Gorton, P. Greenfield, A. Szalay, and R. Williams, "Data-Intensive Computing in the 21st Century," *Computer*, vol. 41, no. 4, pp. 30-32, 2008.

[2] I.D.C. (IDC),"2010 Digital Universe Study: A Digital Universe Decade - Are You Ready?" http://gigaom.files.wordpress.com/2010/05/2010-digital-universe-iview., 2013.

[3] "Symantec. 2010 State of the Data Center Global Data.," http://www.symantec.com/content/en/us/about/media/pdfs/Symantec_DataCenter10_Report _Global.pdf, Jan. 2010., 2013.

[4] M. Seltzer and N. Murphy, "Hierarchical File Systems are Dead," *Proc. 12th Conf. Hot Topics in Operating Systems (HotOS '09)*, 2009.

[5] R. Daley and P. Neumann, "A General-Purpose File System for Secondary Storage," *Proc. Fall Joint Computer Conf., Part I*, pp. 213-229, 1965.

[6] N. Agrawal, W. Bolosky, J. Douceur, and J. Lorch, "A Five-Year Study of File-System Metadata," *Proc. USENIX Conf. File and Storage Technologies (FAST)*, 2007.

[7] A.W. Leung, M. Shao, T. Bisson, S. Pasupathy, and E.L. Miller, "Spyglass: Fast, Scalable Metadata Search for Large-Scale Storage Systems," *Proc. Seventh USENIX Conf. File and Storage Technologies (FAST)*, 2009.

[8] S. Doraimani and A. Iamnitchi, "File Grouping for Scientific Data Management: Lessons from Experimenting with Real Traces," *Proc. 17th Int'l Symp. High Performance Distributed Computing (HPDC '08)*, 2008.

[9] A. Leung, S. Pasupathy, G. Goodson, and E. Miller, "Measurement and Analysis of Large-Scale Network File System Workloads," *Proc. USENIX Ann. Technical Conf. (ATC '08)*, 2008.

[10] A. Ames, C. Maltzahn, N. Bobb, E. Miller, S. Brandt, A. Neeman, A. Hiatt, and D. Tuteja, "Richer File System Metadata Using Links and Attributes," *Proc. Mass Storage Systems and Technologies (MSST)*, 2005.

[11] S. Weil, S.A. Brandt, E.L. Miller, D.D.E. Long, and C. Maltzahn, "Ceph: A Scalable, High-Performance Distributed File System," *Proc. Seventh Symp. Operating Systems Design and Implementation (OSDI)*, 2006.

[12] H. Huang, N. Zhang, W. Wang, G. Das, and A. Szalay, "Just-In-Time Analytics on Large File Systems," *Proc. Ninth USENIX Conf. File and Storage Technologies (FAST)*, 2011.

[13] K. Veeraraghavan, J. Flinn, E.B. Nightingale, and B. Noble, "quFiles: The Right File at the Right Time," *Proc. USENIX Conf. File and Storage Technologies (FAST)*, 2010.

[14] Z. Zhang and C. Karamanolis, "Designing a Robust Namespace for Distributed File Services," *Proc. IEEE 20th Symp. Reliable Distributed Systems (SRDS)*, pp. 162-173, 2001.

[15] Y. Hua, H. Jiang, Y. Zhu, D. Feng, and L. Tian, "SmartStore: A New Metadata Organization Paradigm with Semantic-Awareness for Next-Generation File Systems," *Proc. ACM/IEEE Supercomputing Conf. (SC)*, 2009.

[16] D. Beaver, S. Kumar, H. Li, J. Sobel, and P. Vajgel, "Finding a Needle in Haystack: Facebooks Photo Storage," *Proc. Ninth USENIX Conf. Operating Systems Design and Implementation (OSDI)*, 2010.

[17] S. Sinnamohideen, R. Sambasivan, J. Hendricks, L. Liu, and G. Ganger, "A Transparently-Scalable Metadata Service for the Ursa Minor Storage System," *Proc. USENIX Ann. Technical Conf.*, 2010.

[18] D. Hildebrand and P. Honeyman, "Exporting Storage Systems in a Scalable Manner with pNFS," *Proc. 22nd IEEE / 13th NASA Goddard Conf. Mass Storage Systems and Technologies (MSST)*, 2005.

[19] PVFS2. Parallel Virtual File System, Version 2, http://www.pvfs2.org, 2013.

[20] S. Ghemawat, H. Gobioff, and S. Leung, "The Google File System," *Proc. 19th ACM Symp. Operating Systems Principles (SOSP)*, 2003.

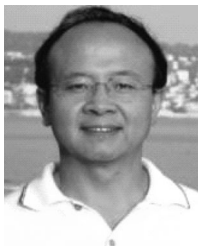
[21] Hadoop Project, http://hadoop.apache.org, 2013.

[22] P. Indyk and R. Motwani, "Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality," *Proc. 30th Ann. ACM Symp. Theory of Computing (STOC)*, 1998.

[23] P. Gu, Y. Zhu, H. Jiang, and J. Wang, "Nexus: A Novel Weighted-Graph-Based Prefetching Algorithm for Metadata Servers in Petabyte-Scale Storage Systems," *Proc. IEEE Sixth Int'l Symp. Cluster Computing and the Grid (CCGrid)*, 2006.

[24] P. Xia, D. Feng, H. Jiang, L. Tian, and F. Wang, "FARMER: A Novel Approach to File Access Correlation Mining and Evaluation Reference Model for Optimizing Peta-Scale File Systems Performance," *Proc. 17th Int'l Symp. High Performance Distributed Computing (HPDC)*, 2008.
[25] E. Riedel, M. Kallahalla, and R. Swaminathan, "A Framework for Evaluating Storage System Security," *Proc. USENIX Conf. File and Storage Technologies (FAST)*, pp. 15-30, 2002.
[26] S. Kavalanekar, B. Worthington, Q. Zhang, and V. Sharda, "Characterization of Storage Workload Traces from Production Windows Servers," *Proc. IEEE Int'l Symp. Workload Characterization (IISWC)*, 2008.
[27] D. Ellard, J. Ledlie, P. Malkani, and M. Seltzer, "Passive NFS Tracing of Email and Research Workloads," *Proc. USENIX Conf. File and Storage Technologies (FAST)*, pp. 203-216, 2003.
[28] J. L. Hellerstein, "Google Cluster Data," http://googleresearch.blogspot.com/2010/01/google-cluster-data.html, Jan. 2010.
[29] A. Andoni and P. Indyk, "Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions," *Comm. the ACM*, vol. 51, pp. 117-122, 2008.
[30] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni, "Locality-Sensitive Hashing Scheme Based on p-Stable Distributions," *Proc. Ann. Symp. Computational Geometry*, pp. 253-262, 2004.
[31] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *ACM SIGMOD Record*, vol. 1, pp. 47-57, 1984.
[32] D. Hitz, J. Lau, and M. Malcolm, "File System Design for an NFS File Server Appliance," *Proc. USENIX Winter Technical Conf.*, pp. 235-246, 1994.
[33] N.C. Hutchinson, S. Manley, M. Federwisch, G. Harris, D. Hitz, S. Kleiman, and S. O'Malley, "Logical vs. Physical File System Backup," *Operating Systems Rev.*, vol. 33, pp. 239-250, 1998.
[34] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-Probe LSH: Efficient Indexing for High-Dimensional Similarity Search," *Proc. VLDB*, pp. 950-961, 2007.
[35] A. Traeger, E. Zadok, N. Joukov, and C. Wright, "A Nine Year Study of File System and Storage Benchmarking," *ACM Trans. Storage*, vol. 4, pp. 1-56, 2008.
[36] D.K. Gifford, P. Jouvelot, M.A. Sheldon, and J.W.O. Jr, "Semantic File Systems," *Proc. Symp. Operating Systems Principle (SOSP)*, 1991.
[37] C. Maltzahn, E. Molina Estolano, A. Khurana, A. J. Nelson, S. A. Brandt, and S. Weil, "Ceph as a Scalable Alternative to the Hadoop Distributed File System," *login: The USENIX Magazine*, vol. 35, pp. 38-49, Aug. 2010.
[38] S. Patil and G. Gibson, "Scale and Concurrency of GIGA+: File System Directories with Millions of Files," *Proc. Ninth USENIX Conf. File and Storage Technologies (FAST)*, 2011.
[39] J. Xing, J. Xiong, N. Sun, and J. Ma, "Adaptive and Scalable Metadata Management to Support a Trillion Files," *Proc. ACM/IEEE Supercomputing Conf. (SC)*, 2009.
[40] S. Weil, K. Pollack, S. Brandt, and E. Miller, "Dynamic Metadata Management for Petabyte-scale File Systems," *Proc. ACM/IEEE Supercomputing*, 2004.
[41] S. Deerwester, S. Dumas, G. Furnas, T. Landauer, and R. Harsman, "Indexing by Latent Semantic Analysis," *J. Am. Soc. for Information Science*, pp. 391-407, 1990.
[42] C. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala, "Latent Semantic Indexing: A Probabilistic Analysis," *J. Computer and System Sciences*, vol. 61, no. 2, pp. 217-235, 2000.

Yu Hua received the BE and PhD degrees in computer science from Wuhan University, China, in 2001 and 2005, respectively. He is currently an associate professor at the Huazhong University of Science and Technology, China. His research interests include computer architecture, cloud computing, network storage and cyber-physical systems. He has more than 50 papers to his credit in major journals and international conferences including *IEEE-TC*, *IEEE-TPDS*, USENIX ATC, INFOCOM, SC, ICDCS, ICPP and MASCOTS. He has been on the organizing and program committees of multiple international conferences, including INFOCOM, ICPP and IWQoS. He is a senior member of the IEEE, and a member of ACM and USENIX.



Hong Jiang received the BSc degree from the Huazhong University of Science and Technology, Wuhan, China, in 1982, the MASc degree from the University of Toronto, Canada, in 1987, and the PhD degree from the Texas A&M University, College Station, in 1991. He is Willa Cather Professor at the University of Nebraska-Lincoln. His research interests include computer architecture, computer storage systems and parallel/distributed computing. He serves as an associate editor of the *IEEE Transactions on Parallel and Distributed Systems*. He has more than 200 publications in major journals and international conferences, including *IEEE-TPDS*, *IEEE-TC*, USENIX-ATC, ISCA, MICRO, FAST, SC, ICS, HPDC, etc. He is a senior member of the IEEE and a member of the ACM and ACM SIGARCH.

Yifeng Zhu received the BSc degree from the Huazhong University of Science and Technology, Wuhan, China, in 1998, and the MS and PhD degrees from the University of Nebraska, Lincoln, in 2002 and 2005, respectively. He is an associate professor at the University of Maine. His research interests include parallel I/O storage systems, and energy-aware memory systems. He served as the program committee of international conferences, including ICDCS and ICPP. He received the Best Paper Award at IEEE CLUSTER 07. He is a member of the ACM, the IEEE, and the Francis Crowe Society.

Dan Feng received the BE, ME, and the PhD degrees in computer science and technology from the Huazhong University of Science and Technology (HUST), China, in 1991, 1994, and 1997, respectively. She is a professor and vice dean of the School of Computer Science and Technology, HUST. Her research interests include computer architecture, massive storage systems, and parallel file systems. She has more than 80 publications to her credit in journals and international conferences, including *IEEE Transactions on Parallel and Distributed Systems*, JCST, USENIX ATC, FAST, ICDCS, HPDC, SC, ICS and ICPP. She is a member of the IEEE.

Lei Xu received the BS degree in electronic science and technology from Wuhan University, China in 2005. He is working toward the PhD degree in the Department of Computer Science at the University of Nebraska Lincoln. His research interests include file system, manycore architecture, operating systems, distributed storage systems and storage class memories.