Spring 4-17-2010

Rapport: Semantic-sensitive Namespace Management in Large-scale File Systems

Yu Hua
Huazhong Univ. of Sci. & Tech., csyhua@hust.edu.cn

Hong Jiang
University of Nebraska-Lincoln, jiang@cse.unl.edu

Yifeng Zhu
University of Maine, zhu@eece.maine.edu

Dan Feng
Huazhong Univ. of Sci. & Tech., dfeng@hust.edu.cn

Rapport: Semantic-sensitive Namespace Management in Large-scale File Systems

Yu Hua
Huazhong Univ. of Sci. & Tech.
Wuhan, China
csyhua@hust.edu.cn

Hong Jiang
Univ. of Nebraska-Lincoln
Lincoln, NE, USA
jiang@cse.unl.edu

Yifeng Zhu
Univ. of Maine
Orono, ME, USA
zhu@eece.maine.edu

Dan Feng
Huazhong Univ. of Sci. & Tech.
Wuhan, China
dfeng@hust.edu.cn

***Abstract*—Explosive growth in volume and complexity of data exacerbates the key challenge to effectively and efficiently manage data in a way that fundamentally improves the ease and efficacy of their use. Existing large-scale file systems rely on hierarchically structured namespace that leads to severe performance bottlenecks and renders it impossible to support real-time queries on multi-dimensional attributes. This paper proposes a novel semantic-sensitive scheme, called Rapport, to provide dynamic and adaptive namespace management and support complex queries. The basic idea is to build files' namespace by utilizing their semantic correlation and exploiting dynamic evolution of attributes to support namespace management. Extensive trace-driven experiments validate the effectiveness and efficiency of our proposed schemes. To the best of our knowledge, this is the first work on semantic-sensitive namespace management for ultra-scale file systems.**

I. Introduction

One key function of namespace management in a file system is to facilitate file identification and lookup for end users. The namespace as an information-organizing infrastructure is fundamental to system quality and performance such as scalability and ease of use, especially for large-scale file systems. Current file systems, unfortunately, are mostly based on hierarchical directory-based structures designed more than forty years ago [1] and few changes have been made since. The basic hierarchical namespace has become restrictive, difficult to use, and limited in scalability especially for modern large-scale systems. On the other hand, modern technologies have made it possible to quickly collect vast amounts of data, such as media data (images, music, videos), network data (web, social network, emails), genomic data and medical data (personal biometrics). However, these data are often ill-organized in existing file systems via hierarchical directories. As a result, users often have to resort to manually navigating hierarchies of billions of files in a modern ultra-scale file system in order to locate target files. This slow manual query process can frustrate users while overwhelming the systems. The root cause of this problem lies in the fact that hierarchical namespace for ExtaByte-scale or larger systems requires the management capability that goes far beyond the abilities of both file system users and administrators.

The hierarchical directory-tree bottleneck problem will only worsen as the demand for concurrent access to data by parallel programs/threads accelerates with the growth of chip multiprocessors' (CMPs) processing capacity at the Moore's Law's pace, due to extra unnecessary accesses to shared parent directories. For instance, */system/storage* and */system/compute*, two otherwise functionally unrelated directories, will likely require accesses through a shared parent directory. The higher the directory tree is, the slower the access speed will be. More specifically, the existing hierarchical namespace faces the following four main challenges, resulting from its weak scalability and inefficient performance.

The first challenge is how to represent and organize explosively growing data and simultaneously satisfy increasingly demanding file lookup and search operations. The second challenge is how to provide real-time query responses while avoiding linear brute-force search on massive storage systems. The exponential increase in file numbers renders the structure of centralized namespace an access bottleneck, which cannot provide real-time responses to system calls, such as queries, insert/delete and update. The third challenge is the potential mismatches between heterogeneous application requirements and allocated homogeneous resources for files. The hierarchical namespace needs to provide the guarantee for strong consistency, which requires high costs [2], [3], and treats each file equally, whatever its importance and correlation, by allocating similar system resources, such as storage space and processed time, thus grossly mismatching the requirements of real-world applications with precious system resources. The fourth challenge is the lack of multi-dimensional attributes used for file management in each directory. All files within one directory are displayed in one-dimensional order, despite of the availability of useful multi-dimensional file attributes, such as filename, size, type and creation/modification time.

While there is a body of existing research in the literature, such as Semantic File System (SFS) [4], quFiles [5], Perspective [6], Spyglass [7], DiFFS [3], Copernicus [8], Magellan [9] and SmartStore [10], that have attempted to solve these problems by using semantic grouping techniques, most of them nevertheless inherit the fundamental concept of the hierarchical directory tree structure that limits the scalability and functionality of systems with trillions of files.

To address these challenges and problems and break away from the limiting notion of hierarchical directory tree for namespace management, we propose a novel approach to managing namespace, called Rapport, that represents a file by using semantic correlation, not conventional static filename, to accurately describe the dynamic evolution of file-oriented behaviors that can guide system optimization. The Rapport scheme can be made compatible with or orthogonal to other existing namespace schemes to further improve their performance. More specifically, Rapport is a virtual middleware that

can be deployed/embedded in any file systems as long as they provide multi-dimensional attributes of stored files, which can be used to allow users to obtain queried results of "*what they want*" (attribute description), besides "*where the file is located*" (full pathname in file systems). Transparent to users, Rapport can provide automatic placement and organization for massive files.

Different from the namespace schemes in existing file systems, Rapport can support multiple types of queries, including the conventional point query and complex queries (such as range and top-k). More importantly, users can obtain not only accurate queried results but also the correlated file information along with the query results in the form of a correlated and sanitized small flat namespace. Specifically, the namespaces of files are built by their semantic correlation, not simple IDs or filenames, to facilitate fast indexing and dynamic update according to the changes of file attributes.

Example 1: As shown in the right part of Figure 1, if a file named "*Myson.jpeg*" is identified to be correlated with other 6 files in a multi-dimensional attribute space, Rapport uses a correlation to represent this file's semantic-sensitive small flat namespace as "*Myson.jpeg=R(WithMonther.jpeg, WithFather.jpeg, WithFamily.jpeg, Say-mama.mp3, Birthday.txt, 1month.rmvb)*". A conventional hierarchically structured namespace would likely require navigating through more than two different branches of the directory tree to locate these otherwise correlated files, as shown in the left part of Figure 1.

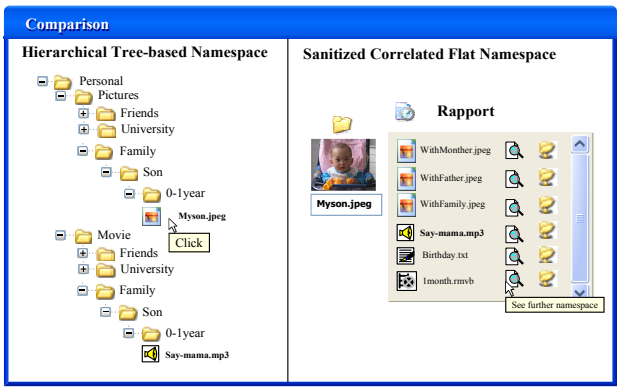


Fig. 1. An envisioned interface of Rapport.

The proposed Rapport semantic-sensitive namespace serves as an infrastructure to provide a number of benefits for both the users and the systems, as summarized in the following main contributions of Rapport.

- **Semantic namespace construction.** The proposed Rapport is the first attempt at providing semantic namespace in large-scale file systems, to the best of our knowledge. Different from existing namespace schemes, the basic idea behind Rapport is to use semantic correlation relationship of multi-dimensional attributes, rather than one-dimensional attribute such as simple physical address or file names, to represent files to overcome the problems of a hierarchical namespace. A user can hence obtain the required files in an efficient way, without having to navigate in the data ocean.
- **Simple correlation identification.** Rapport makes use of the simple computation of locality-sensitive hashing (LSH) to accurately and efficiently identify semantically correlated files. It then represents each file based on the semantic correlation. Since the relationship among the files often evolves, Rapport can fast identify their changes to update the namespace by exploiting the particular file semantic of provenance.
- **Flexible query services.** Rapport can support conventional point query and complex queries (range and top-k) within a constant-scale computation complexity and also provide a sanitized and small flat namespace in the form of the files correlated with the queried results. The available namespace can guide and provide useful hints to users to accurately express and select which files they actually want.

The rest of this paper is organized as follows. Section II presents the problem description and research backgrounds of locality sensitive hashing. We show the design of Rapport scheme in Section III. Section IV discusses semantic computation and further improvements to support complex query services. Section V shows the namespace management and query operations. We present the details of system implementation in Section VI. Section VII shows the performance evaluation of Rapport. Section VIII shows the related work. We conclude our paper in Section IX.

II. Problem Description and Background

An accurate and flexible semantic-sensitive namespace aims to facilitate the management of trillions of files, arguably significantly benefiting both users and system managers.

A. Problem Description

Most large-scale file systems are networked and distributed, thus making storage system management more complex and generating high performance overhead if the 40+-year-old hierarchically structured namespace design is used. A fast, flexible, user-friendly, and scalable file management scheme is needed in large-scale file systems to facilitate ad hoc file queries on multi-dimensional attributes. These queries involve indexing file metadata, such as inode fields and extended attributes, to allow point, range and top-k queries over file attributes. This new management scheme should help users and administrators understand some fundamental questions. For example, "*what do a given set of files represent (semantic)? Where are they located (position)? How are they used (behaviors)? How do they evolve (provenance)? What other files are closely and meaningfully related to them (correlation)?*"

The key issue is how to accurately extract "useful knowledge" in a light way from a huge data ocean. The useful knowledge here is interpreted as semantic representation of correlated files that can be iteratively aggregated into correlated groups to form semantic-sensitive namespaces. As shown in Example 1, a file "*Myson.jpeg*"'s namespace can be represented as "*=R(WithMonther.jpeg, WithFather.jpeg, WithFamily.jpeg, Say-mama.mp3, Birthday.txt, 1month.rmvb)*" by

using the semantic correlation, where the files in the namespace can collectively and uniquely represent "*Myson.jpeg*". A file's semantic-sensitive namespace essentially is a set of files that are most correlated to "*Myson.jpeg*". The correlation is measured by the distance in a metric space, such as Euclidean distance. The problem we aim to solve in this paper then becomes one of how to find the t nearest neighbors for each file to represent, where t is the number of correlated files that can uniquely represent a given file.

B. Observation

Optimizing file system organization and query process requires us to well understand the characteristics of file metadata stored and accessed. By studying file systems workloads and snapshot traces, many studies in the literature have found that most accesses in file systems exhibit strong locality.

- Study [11] shows that on average 78% files are locally unwritten in a five-year period. More than 75% directories have a directory depth larger than 7. The number of small-capacity file system volumes has decreased and in particular, systems of size 4 GB or less accounted for only 4% of all examined file systems, including NTFS, FAT32 and FAT, that had a total of 4 billion files and 700TB file data.
- Spyglass [7] reports that the locality ratios are below 1% in many traces, meaning that correlated files are contained in less than 1% of the directory space. In other words, correlated files are widely dispersed across the directory space.
- Filecules [12] examines a large set of real traces and concludes that files can be classified into correlated groups since 6.5% of files account for 45% of I/O requests based on its findings.
- Measurement of large-scale network file system workloads [13] further verifies that fewer than 1% clients issue 50% file requests and over 60% re-open operations take place within one minute.

The above facts reveal strongly skewed, Zipfian-like distributions of metadata and access patterns, which imply that only a small subset of files may be frequently accessed and thus performing semantic file grouping may prove to be effective and efficient.

C. Locality Sensitive Hashing

Locality Sensitive Hashing (LSH) [14] is a memory-efficient tool that maps similar items into the same hash buckets. The LSH function family has the property that items close to each other will have a higher probability of colliding than items that are far apart. The closeness depends upon the R value that is obtained practically from the sampling approach in real-world applications [15], [16]. In this paper, we use LSH to achieve fast similarity search. For a given request, we first hash query point q into buckets in multiple hash tables, then union all items in those chosen buckets by ranking them according to their distances to the query point q, and finally select the closest items to a queried one.

We define S to be the domain of items and $||*||$ to be the distance between two items.

Definition 1: LSH function family, i.e., $\mathbb{H} = \{h : S \rightarrow U\}$ is called (R, cR, P_1, P_2)-sensitive for distance function $||*||$ if for any $p, q \in S$

- If $||p,q|| \leq R$ then $Pr_{\mathbb{H}}[h(p) = h(q)] \geq P_1$,
- If $||p,q|| \geq cR$ then $Pr_{\mathbb{H}}[h(p) = h(q)] \leq P_2$.

To allow the similarity search, we set $c > 1$ and $P_1 > P_2$. In practice, multiple hash functions are used to increase the gap between P_1 and P_2. The distance functions $||*||$ correspond to different LSH families of l_s norms based on s-stable distribution $h_{a,b}(v) = \lfloor \frac{a \cdot v + b}{\omega} \rfloor$, where a is a d-dimensional random vector with chosen entries following an s-stable distribution and b is a real number chosen uniformly from the range $[0, \omega)$.

The intuition behind such a hash function is that, if two points are close to each other, then with a high probability their shifted projections (on line a) will fall in the same interval. On the other hand, two faraway points are very likely to be projected into two different intervals. The value of P_1 describes the acceptable quality of queried results. Given a dataset under a specified metric space, we can determine the P_1 value based on pre-defined parameters [17].

Observation 1: **(P_1 Principle)** Let $\rho = \frac{\ln 1/P_1}{\ln 1/P_2}$. If $\rho \leq 1/c$, LSH can guarantee to run with time complexity $O(n^{\rho})$, which is sub-linear to n.

LSH implements the locality-sensitive approach by using multiple hash tables, in order to produce a higher probability of containment within one bucket for proximate items. For example, if query point q is a close neighbor to point p_1, they are stored in one bucket of hash tables with a high probability, say, they are in the same bucket in the first and second hash tables. On the other hand, point p_3 has a very low probability of being placed together with point q into one bucket due to their much longer Euclidean distance.

Two key parameters of an LSH-based structure are M, the capacity of a function family $\mathbb{G}$, and L, the number of hash tables. We first define a function family $\mathbb{G} = \{g : S \rightarrow U^M\}$ such that, for a d-dimensional vector v, $g(v) = (h_1(v), \cdots, h_M(v))$, where $h_j \in \mathbb{H}$ for $1 \leq j \leq M$. The $g(v)$ hence becomes the concatenation of M LSH functions. Second, we randomly select L functions $g_1, \cdots, g_L$ from $\mathbb{G}$, each of which, g_i $(1 \leq i \leq L)$, is associated with one hash table, thus requiring L hash tables. A vector v will be further hashed into a bucket (positioned by $g_i(v)$) in each hash table. Since the total number of hash buckets may be large, we can only maintain non-empty buckets by using the regular hashing in practice. The optimal values of M and L depend on the distance to the nearest neighbors.

III. Design of Rapport

This section presents the design of Rapport and its main functional components to facilitate the semantic-sensitive namespace representation and to enable complex queries on multiple attributes within the new non-directory-based namespace. Examples of complex queries studied in this paper include point, range and top-k.

Definition 2: **Semantic-Sensitive Namespace Representation**. In Rapport, a file f of p attributes $A(f) = (a_1, a_2, \cdots, a_p)$

uses a subset of attributes $A^*(A^* \subseteq A)$ to identify t semantically correlated files $S_{A^*}(f) = (f_1, f_2, \cdots, f_t)$ with the correlation degrees $(d_1, d_2, \cdots, d_t)$. The semantic-sensitive namespace of f is then represented by the correlation R_{A^*}, as follows.

$$File(f) = R_{A^*}\{(f_1,d_1),(f_2,d_2),\cdots,(f_t,d_t)\} \quad (1)$$

$$d_i = 1 - \frac{E_{f,i}}{D} \quad (2)$$

where $1 \leq i \leq t$, D is a large constant, and $E_{f,i}$ is the Euclidean distance between file f and file i in the semantic namespace. As the file system evolves, the correlated file set A^* is dynamically adjusted (see Section V for more details).

A. Design Principles for Semantic-Sensitive Namespace

Rapport makes use of semantic correlation existing among files to build the namespace representation that can accurately reflect dynamic changes of file attributes and also provide fast complex query. One fundamental research issue is whether we can exploit the dynamically evolving correlation representation described above to create an accurate semantic-sensitive namespace in a very large-scale file system with a small performance overhead.

In order to achieve the accuracy and scalability, our design follows the basic principles described below.

1) **Unique**. A namespace scheme should provide a unique representation for each file to uniquely identify a file entity and quickly differentiate it from other files.
2) **Correlated**. An ideal namespace should be able to accurately identify the files that are strongly semantically correlated to facilitate queries.
3) **Adaptable**. An adaptable namespace can automatically adapt the file name representation to the dynamic evolution of file attributes and file correlation.
4) **Scalable**. Large-scale file systems require a scalable namespace to support efficient representation of differentiated files and parallel operations.

Rapport carries out semantic grouping among massive data and provides complex queries within the new and small flat semantic-sensitive namespace. As shown in Figure 2, Rapport consists of four basic functional modules that execute the four main Rapport tasks respectively, namely, semantic correlation identification, namespace management, complex queries, and dynamic evolution. In the first task, the LSH-based computation on multi-dimensional file attributes is performed to quickly identify correlated files. Different attribute sets may lead to semantic groups of various accuracies. However, it is non-trivial to select the optimal set of attributes that most accurately define file correlations and best match access patterns. The second task then aggregates semantically correlated files into one or a small number of groups, in which Rapport performs nearest neighbor search for each file to identify those files belonging to its semantic namespace. Within the newly generated namespace Rapport's third task carries out complex queries, which are different from existing work, by providing correlation space to help user obtain a better understanding on their file data. Finally, since the attributes of files and their correlation may change from time to time, Rapport's fourth task helps it accurately adapt to dynamic evolution and make necessary adjustments.

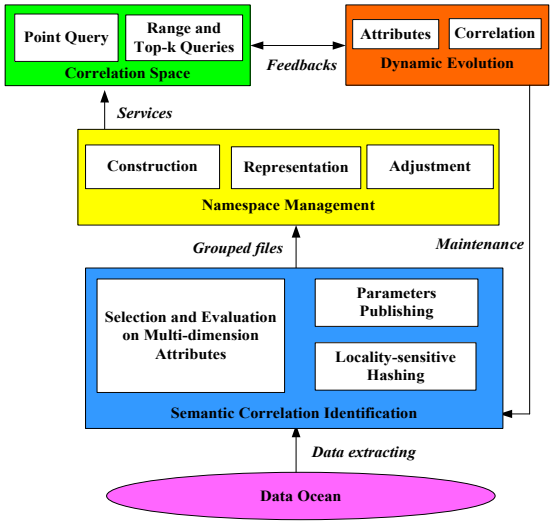


Fig. 2. Rapport architecture.

IV. Semantic Correlation Computation

Rapport makes use of the LSH computation to probabilistically group semantically correlated files into the same or adjacent buckets. Existing LSH approaches face a dilemma between two conflicting goals of space/query efficiency and approximation guarantee. Specifically, if the accuracy of the retrieved neighbor is crucial, a large amount of space is then needed, resulting in a high query cost. On the other hand, in order to meet a tight space or stringent query time requirement, one must sacrifice the accuracy guarantee of LSH.

A. Bounded LSH Computation

Bounded LSH (B-LSH) [18] is used in our design to provide fast search and reduce space overhead. Figure 3 shows the comparison between basic and bounded LSH structures containing the same number of clustered data objects. In basic LSH, too many "hot spot" objects close to each other are likely placed into the same or adjacent buckets. As a result, these buckets become overloaded, leading to low search efficiency and accuracy. Bounded LSH deals with this issue by actively migrating extra data into adjacent buckets. It is also observed that bounded LSH uses much fewer hash tables than basic LSH since the buckets of bounded LSH are load-balanced, while guaranteeing that the data in close-by buckets are correlated.

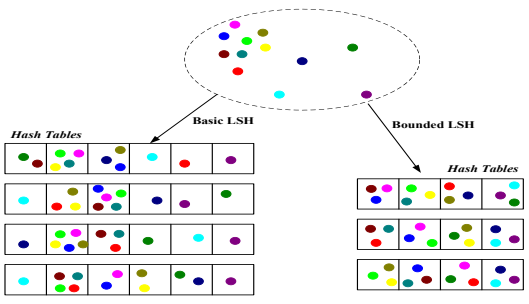


Fig. 3. Comparisons between basic and bounded LSH structures.

LSH maps data objects to hash buckets, each of which contains approximately the same number of objects. The bucket size is an important parameter in bounded LSH. A space-efficient hashing algorithm proposed in [19] allocates $(dn + n\log^{O(1)} n)$ space, which almost matches the lower bound for hash-based algorithm recently obtained in [20]. In bounded LSH, we define each bucket size as $\lceil (dn + n\log^{O(1)} n)/(L \cdot T) \rceil$ for n d-dimension objects stored in L hash tables, each of which maintains at most T buckets. In practice, the optimal values of L and T are determined by two steps, which first determine the bounds on L and T that guarantee the design correctness and then, within those bounds, choose the values of L and T that can achieve the best expected tradeoff between query running time and search accuracy. We carry out a series of operations to build Rapport, including *Initialization*, *Free Hashing*, *Local Ranking* and *Ordered Overflowing* as shown in [18]. Bounded LSH constructs L hash tables, each of which contains M LSH functions that follow the 2-stable Gaussian distribution for the Euclidean distance. Meanwhile, each bucket in a hash table can contain at most $\triangle_{B-LSH} = \lceil (dn + n\log^{O(1)} n)/(L \cdot T) \rceil$ objects. Further operations depend upon the number of vectors contained in a bucket, i.e., $Number(Bucket_{g_i(v)})$, to which $g_i(v)$ points.

B. Improvements upon LSH Structure

Although bounded LSH works well in identifying correlated data objects while balancing load, it inherits the space-inefficiency disadvantage of the LSH-based structures. Since bounded LSH requires multiple hash tables to store correlated data, the space overhead becomes a potential performance bottleneck when dealing with massive amounts of data as the corresponding indexing structure overflows the main memory, leading to slow hard disk accesses and thus performance degradation. Furthermore, while the form of hash tables in bounded LSH may work 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. In light of these performance issues, we make use of the R-tree [21] structure to replace the original hash tables, store the correlated data, and represent their multi-dimensional attribute information in R-tree nodes.

An R-tree [21] is a tree-based data structure that, similar to B-tree [22], is often used to represent and index spatial multi-dimensional data. An R-tree can split data space into hierarchically nested bounding boxes that can contain several data entities within certain ranges. The R-tree structure [21] can efficiently support point, range and top-k queries by maintaining index records in its leaf nodes containing pointers to their data. This completely dynamic index structure is able to provide efficient query service by visiting only a small number of nodes in a spatial search [10]. The path length from the root to any leaf node is identical since the index structure of R-tree is height-balanced, which is called the R-tree height. The R-tree index structure uses solid Minimum Bounding Rectangles (MBRs), i.e., bounding boxes, to indicate the queried regions. An MBR in each dimension denotes an interval of the enclosed data with a lower and an upper bound [23]. In our design, we exploit its special capability for supporting range and top-k queries by modifying its structure appropriately to store the semantically grouped files.

V. Namespace Management and Query Services

This section presents the semantic-sensitive namespace management scheme in Rapport to provide unique and correlated naming for files and support complex queries in large-scale file systems. The Rapport namespace takes into account the dynamic evolution of file attributes and correlation.

A. Namespace Construction

A file in Rapport is represented in the form of semantic correlation, as defined in Equation 1. We make use of the LSH computation to fast and accurately identify correlated files that are then placed into the same or adjacent nodes in R-tree as discussed in Section IV to build a semantic-sensitive namespace. Specifically, for each file within an R-tree node, its namespace is derived from the results of a top-t query that can find the t nearest neighbors in the multi-dimensional attribute space. These most correlated neighbors constitute the namespace of this file. For example, if a file *Myson.jpeg* is located under an R-tree node, we carry out a top-6 query to search its 6 nearest neighbors, such as ($WithMonther.jpeg, WithFather.jpeg, WithFamily.jpeg, Say-mama.mp3, Birthday.txt, 1month.rmvb$) by using a top-6 algorithm as presented in Section V-D. In this way, the semantic-sensitive namespace of the file *Myson.jpeg* is represented as $R\{(WithMonther.jpeg, 0.85), (WithFather.jpeg, 0.79), \cdots\}$.

If the node holding the file *Myson.jpeg* contains a smaller number of files than t, we can execute similar nearest neighbor search operation upon adjacent nodes that may contain correlated files with a high probability according to the LSH definition [14].

The basic idea behind the namespace construction is to include the t nearest neighbors of a file in the multi-dimensional attribute space. The rationale behind this is that potential neighbors in the multi-dimensional attribute space should reside in the same or adjacent nodes. Hence, performing a top-t query should find correlated files belonging to the namespace of this file. We further 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 multi-dimensional attribute space. This metric can also help differentiate the file names and guarantee the unique representation. For example, assuming that both files A and B are correlated with file C, Rapport thus considers file C as a member of namespaces for $R(A)$ and $R(B)$ respectively, which may potentially produce the same naming representation. When using the correlation degree, i.e., $(f_C, 0.7)$ and $(f_C, 0.5)$ respectively in $R(A)$ and $R(B)$, we can easily differentiate their namespace due to different values of their correlation degrees. In the worst case, the representation of two files may be exactly the same, i.e., they may have the same namespace members and correlation degrees. Although it occurs rarely, we solve this representation

collision by increasing the namespace size until we obtain a unique representation.

The parameter t is a key value in deciding the namespace size of each file in the Rapport scheme. If the t value is too small, i.e., having a very small number of members, it is difficult to differentiate the represented files. On the other hand, a large value often involves some not-strongly-correlated files that potentially decrease the semantic correlation in a namespace, possibly also requiring high cost for namespace update. Therefore, we need to strike a tradeoff between the unique representation and semantic correlation guarantee. Rapport interprets the above quality requirements as a simple optimization function by examining the Mean and Standard Deviation (MSD) as follows.

$$\max\{MSD(f)\} = \max\{\bar{d} + \alpha\sqrt{\frac{\sum_{i=1}^{t}(d_i - \bar{d})^2}{t}}\}$$

where $\bar{d} = \frac{\sum_{i=1}^{t} d_i}{t}$ and α is the correlation factor. The correlation factor is obtained from the result of sampling historical records and can also be adjusted according to the specific requirements in real-world applications. The rationale behind this interpretation is that the mean of all correlated degrees faithfully describes the correlation between a given file and the members of its namespace and the standard deviation allows the namespace to select differentiated members to guarantee the unique representation. The MSD model hence becomes one of the several feasible approaches to quantifying the qualitative requirements of constructing a semantic-sensitive namespace. Specifically, a file f takes into account $t = 1$ file, i.e., the nearest neighbor, to determine the initial $MSD(f)$ value. If the addition of the next most correlated file to the namespace can increase the $MSD(f)$ value, this file is then considered to be a member of file f namespace and t values is also increased by 1. This process repeats until the addition of the next most correlated file decreases the $MSD(f)$ value.

B. Dynamic Evolution

The semantic naming can potentially adapt to the dynamic evolution of file attributes and correlation, which is one of the most salient features of Rapport that distinguish it from most existing schemes. In Rapport, a file f locally maintains the membership information of two sets, i.e., the set $File(f)$ of files constituting the namespace of file f and the set $Member(f)$ of files whose namespaces include file f as a member. For example, the file $Myson.jpeg$ keeps its namespace members, i.e., $File(Myson.jpeg) = (WithMonther.jpeg, WithFather.jpeg, WithFamily.jpeg, Say-mama.mp3, Birthday.txt, 1month.rmvb)$. On the other hand, the file $Myson.jpeg$ may be also used for representing other files and serving as a member of other files' namespaces, e.g., $Member(Myson.jpeg) = (toy.doc, kindergarten.pdf)$. This two-set design can facilitate fast update on staleness and maintain information consistency.

When the values of a file f's attributes change, it first executes a new top-t query to re-build its namespace by finding the t nearest neighbors in the multi-dimensional space. These new neighbors form the updated $File(f)$ set. File f further transfers its new attribute values to the members of set $Member(f)$ to update their namespaces, which may trigger re-computations for semantic correlations.

C. Publishing LSH Parameters

A user query specifies the attributes of the target files by indicating a file name for point query, multi-dimensional attribute intervals for range query, and a given attribute (e.g., file name) and k value for top-k query. Rapport can efficiently provide correlated files, which constitute the correlation space and are obtained from the namespace of these results. Although providing correlated results may not be very difficult for a web search engine, it is quite challenging for file systems since there are not enough attributes from an unstructured file organization. We address this challenge by using simple hashing computations to meet the real-time requirement in large-scale file systems. Specifically, each client must first obtain the LSH computation parameters, i.e., M and L, as described in Section II-C. M is the capacity of a function family $\mathbb{G}$ and L is the number of hash tables. In order to carry out an accurate LSH computation, Rapport configures each client with these two parameters to fast locate correlated files. Publishing updates from the file system to the clients/applications allows real-time updates of the LSH computation parameters.

D. Query Algorithms

Although successful for databases, R-tree-based research has not been directly conducted in large-scale distributed file systems, especially for supporting complex queries. Rapport attempts to exploit the R-tree structure for semantic grouping of correlated files to significantly reduce the costs of key operations in complex queries. Since point query is relatively easy to implement by probing the multi-dimensional ranges from the root to the leaf nodes of an R-tree, we only present the algorithms of complex queries, i.e., range and top-k queries, respectively shown in Figure 4 and 5.

Range_Query

Input: Range query request Q
Output: Files covered by Q

1: $NodeList := Null$
2: Target:= CheckContextPic(Q)
3: **while** $Q \cap MBR(Target) = \emptyset$ **and** $Target \neq Root$ **do**
4: Target:=ParentNode(Target)
5: **end while**
6: **if** $Target \neq Root$ **then**
7: Add target node into $NodeList$
8: $Target := SilbingNode(Target)$
9: **while** $Q \cap MBR(Target) \neq \emptyset$ **do**
10: Add target node into $NodeList$
11: $Target := SilbingNode(Target)$
12: **end while**
13: **end if**
14: Check target nodes in $NodeList$
15: Return files covered by Q

Fig. 4. Algorithm for range query.

Top-k_Nearest_Neighbor_Query

Input: A query point q and the number of nearest neighbors k
Output: Files satisfying query request

1: $MinD := \infty, FileList := Null$
2: TargetNode:= CheckContextPic(q, k)
3: $FileList := LocalQuery(TargetNode, q, k)$
4: $Update(MinD, FileList)$
5: $TargetNode := SilbingNode(TargetNode)$
6: **while** $Distance(q, TargetNode) < MinD$ **do**
7: Local query on $TargetNode$
8: Update $FileList$ and $MinD$
9: **if** $Visited(SilbingNodes)$ **then**
10: $TargetNode := FatherNode(TargetNode)$
11: **else**
12: $Target := SilbingNode(Target)$
13: **end if**
14: **end while**
15: Return $FileList$

Fig. 5. Algorithm for querying top-k nearest neighbors.

VI. System Implementation

A. Environments

We have implemented a prototype of Rapport in Linux kernel 2.4.21 and performed experiments in a cluster of 60 server nodes, each with Intel Core 2 Duo CPU and 2GB memory. An RPC-based interface to WAFL (Write Anywhere File Layout) gathers the changes of file attributes, driven by our snapshot-based traces, to evaluate the Rapport performance in terms of "rename" operation when taking into account the dynamic evolution of the entire file system. The prototype contains the four basic functional components of Rapport, i.e., supporting LSH-based correlation identification, constructing the semantic-sensitive namespace, handling complex queries, and managing dynamic evolution of attributes and correlation. These components are implemented in the user space. A client captures the file system operations from the user traces and then delivers the query requests to the servers. Both clients and servers use multiple threads to exchange messages and data via TCP/IP. The IP encapsulation technique helps forward the query requests among multiple servers. We carry out the experiments for 30 runs each to validate the results according to the evaluation guidelines of file and storage systems [24]. The file attributes considered in this evaluation exhibit access locality and skewed distribution especially for multi-dimensional attributes.

B. Traces and Queries

We use three representative traces, i.e., the HP file system trace [25], the MSN trace [26] and the EECS NFS server (EECS) trace at Harvard [27] to drive Rapport performance evaluation. In order to emulate the I/O behaviors of large-scale file systems for which no realistic traces exist, we scaled up the existing I/O traces of current storage systems both spatially and temporally. 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 time of all sub-traces is 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 presents a heavier workload (higher intensity). The number of sub-traces replayed concurrently is denoted as the *Trace Intensifying Factor* (TIF) as shown in Table I, II and III. Similar workload scale-up approaches have also been used in other studies [10], [28].

While point queries based on file names are very common in most file system workloads, no file system I/O traces representing requests for complex queries are publically available. In this paper, we use a synthetic approach to generating complex queries within the multi-dimensional attribute space. The key idea of synthesizing complex quires is to statistically generate random queries in a multi-dimensional space. The file static attributes and behavioral attributes are derived from the available I/O traces. More specifically, a range query is formed by points along multiple attribute dimensions and a top-k query must specify the multi-dimensional coordinate of a given point and the k value. For example, a range query aiming to find all the files that were revised between times 2:00 to 5:10, with the amount of "read" data ranging from 2MB to 10MB, and the amount of "write" data ranging from 6MB to 9MB, can be represented by two points in a 3-dimensional attribute space, i.e., (2:00, 2, 6) and (5:10, 10, 9). Similarly, a top-k query in the form of (8:30, 16.5, 58.2, 5) represents a search for the top-5 files that are closest to the description of a file that is last revised at time 8:30, with the amounts of "read" and "write" data being approximately 16.5MB and 58.2MB, respectively. Therefore, it is reasonable and justifiable for us to utilize random numbers as the coordinates of queried points that are assumed to follow either the Uniform, Gauss, or Zipf distribution to comprehensively evaluate the complex query performance. Due to space limitation, we mainly present the results based on the Zipf distribution.

Query requests are generated from the attribute space of the above representative traces and are randomly selected by considering the different distributions. We further set the Zipfian parameter H to be 0.8. We select these query requests to constitute the query set and examine the query accuracy and latency. We compare Rapport with SmartStore [10] and the baseline system, a popular DBMS-based database approach that uses a B^+ tree [29] to index each metadata attribute without considering any database optimization and is denoted as Baseline.

C. Parameter Selection

The performance of Rapport is sensitive to the parameter settings. One of the key parameters is the metric R that regulates the measure of approximate membership. The LSH-based structures can work well if R is roughly equivalent to the distance between the queried point q and its exact nearest neighbor. Unfortunately, identifying an optimal R value is non-trivial due to the uncertainties and probabilistic nature of LSH. Either too large or too small R value may possibly result in unsatisfactory query results [18]. Worse still, the optimal R value in fact may not exist at all since it often exhibits an unpredictable behavior, e.g., an R value working well for some cases may perform very poorly for others [16]. A good choice

TABLE I
SCALED-UP HP .

	Original	TIF=400
request (million)	94.7	37880
active users	32	12800
user accounts	207	82800
active files (million)	0.969	387.6
total files (million)	4	1600

TABLE II
SCALED-UP MSN .

	Original	TIF=400
# of files (million)	1.25	500
total READ (million)	3.30	1230
total WRITE (million)	1.17	468
duration (hours)	6	2400
total I/O (million)	4.47	1788

TABLE III
SCALED-UP EECS .

	Original	TIF=400
total READ (million)	0.46	184
READ size (GB)	5.1	2040
total WRITE (million)	0.667	266.8
WRITE size (GB)	9.1	3640
total operations (million)	4.44	1776

of the R value must well balance between the query efficiency and the quality guarantee of approximation.

In order to obtain reasonable R values for our experiments, we use the sampling method proposed in the LSH statement [15] and through practical applications [16]. We use "*proximity measure* $\chi = ||p_1^\star - q||/||p_1 - q||$" to evaluate the top-1 query quality for queried point q, where $p_1^\star$ and p_1 respectively represent the actual and searched nearest neighbors of point q by computing their Euclidean distance. We generally can obtain an optimal R value that only works for a given case, but not for all cases. Identifying optimal R values hence requires case-by-case study. Selecting R values essentially depends upon the distribution of measured points as shown in Ref. [16], [18]. We determine R to be 1200, 800 and 1000, as shown in Figure 6, respectively for the intensified *HP*, *MSN* and *EECS* traces. In addition, we use $L = 8$ LSH to support queries with $\omega = 0.75,\ M = 12$.

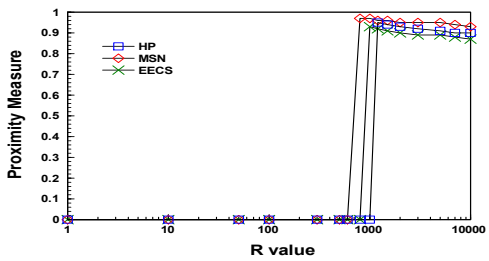


Fig. 6. R values for used traces.

D. Evaluation Metric

To the best of our knowledge, there is no specified evaluation methodology, i.e., metric, to examine the performance of naming schemes, despite of the existing research on file naming. We here present an approximate but simple performance metric, called the "*rename*" cost, to evaluate a naming scheme by executing a "*rename*" operation on a given file and then measuring the time it takes to update all filenames in response to this rename operation. The rationale behind using this metric is that one of the main costs for managing a namespace is to maintain filename consistency among multiple distributed servers.

VII. EVALUATION RESULTS

In this section, we evaluate the efficiency and effectiveness of Rapport in terms of initial namespace construction time, "*rename*" operation latency, accuracy and latency of complex queries and system scalability.

A. Initial Namespace Construction Time

The Rapport system is initialized by first carrying out the LSH-based hash computation to identify correlated files that are then organized into an R-tree structure, in which each file executes a top-t query to identify the members of its namespace. This is the process by which initial namespaces of files are constructed. Figure 7 shows this construction time under the three traces introduced previously. The construction time includes the latencies of all the above operations. We also compare Rapport with SmartStore in this measure and observe that Rapport requires much smaller time since it executes the simple hashing computation, while SmartStore uses the time-consuming LSI tool [30] to decompose a large matrix, thus incurring higher computation overhead.

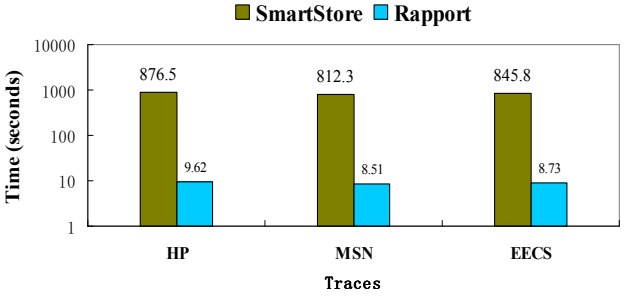


Fig. 7. Construction time.

B. Latency of File Rename Operation

We use the latency of the file rename operation to evaluate and compare the performance of the namespace management schemes of Rapport and SmartStore. As shown in Figure 8, while the average rename latency of Rapport is 0.08*s*, 0.05*s*, and 0.06*s* for the HP, MSN and EECS traces, respectively, SmartStore incurs an average rename latency of 0.23*s*, 0.17*s*, and 0.19*s* respectively. Nevertheless, both Rapport and SmartStore significantly outperform the baseline approach (i.e., DBMS). The main reason is that both Rapport and SmartStore use semantic grouping to aggregate the correlated files that tend to be accessed together. The *rename* operation hence can be completed within one or a very small number of file groups, avoiding checking any irrelevant files. Furthermore, Rapport achieves a smaller latency than SmartStore since the former explicitly exploits the two-set design of $Member(f)$ and $File(f)$, described in Section V-B, that significantly expedites the update process.

C. Query Accuracy

Rapport can provide complex query services, such as point, range and top-k queries, while also showing a small correlated flat namespace of any individual file. We measure the hit rate for point query and the "recall" metric for range and top-k queries. We adopt the "recall" metric as a measure for complex query quality from the field of information retrieval [31]. Given a query q, we denote $T(q)$ the ideal set of k nearest objects and $A(q)$ the actual neighbors reported by Rapport. We define *recall* as $recall = \frac{|T(q) \cap A(q)|}{T(q)}$.

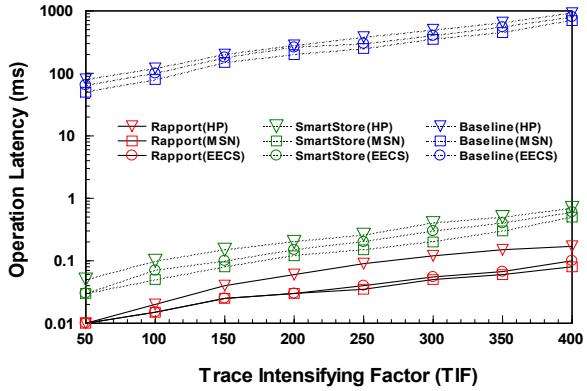


Fig. 8. "rename" operation latency.

Figure 9 shows the hit rate of point query under the intensified traces of $TIF = 300$. The average hit rate in Rapport is 93.7%, 95.2% and 94.6%, respectively for the HP, MSN and EECS traces, visibly outperforming SmartStore (88.6%, 91.1% and 90.2%). The main reasons behind Rapport's superiority to SmartStore are twofold. First, the former exploits the LSH functions that can significantly alleviate the impact of stale information. Second, the two-set design behind the semantic-sensitive namespace of the former makes it possible to accurately and timely search the updated results. Tables IV and V present the recall measures of range and top-k queries in Rapport when the query requests follow the uniform and Zipf distributions respectively. The experimental results show that Rapport can provide accurate queried results.

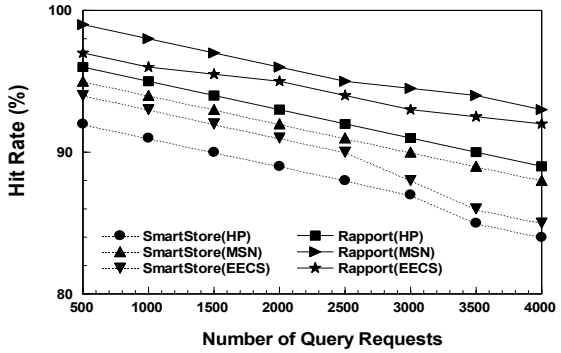


Fig. 9. Average hit rate for point query.

TABLE IV
QUERY ACCURACY (RECALL) OF RANGE AND TOP-K QUERIES THAT FOLLOW A UNIFORM DISTRIBUTION.

Traces	TIF	Range Query Number			Top-k Query Number		
		1000	2000	3000	1000	2000	3000
HP	150	89.5	87.3	85.6	93.7	92.8	91.2
	350	88.2	86.5	84.1	90.3	89.5	88.7
	450	84.7	82.1	80.3	88.6	86.5	84.9
MSN	150	91.2	90.3	89.6	95.8	94.2	91.8
	350	90.3	88.9	86.5	93.7	92.4	90.6
	450	89.2	87.3	85.5	90.2	89.1	86.9
EECS	150	92.9	91.7	90.4	97.2	96.5	93.9
	350	90.5	89.2	87.7	94.2	92.6	90.7
	450	88.3	86.2	83.6	91.3	90.1	89.5

D. Query Latency

Figure 10 compares the query latencies of point and complex queries carried out by the Rapport, SmartStore and Baseline approaches. The results show that Rapport incurs much smaller query latencies than SmartStore, up to 58.2%, 52.5% and 54.7% smaller in the HP, MSN and EECS traces respectively. This again is attributed to the fact that Rapport uses the faster LSH-based hash computation while SmartStore uses the slower LSI tool. Table VI summarizes the latency measures of range and top-k queries. For these complex queries, Rapport and SmartStore vastly outperform the baseline approach, by four and three orders of magnitude respectively.

TABLE V
QUERY ACCURACY (RECALL) OF RANGE AND TOP-k QUERIES THAT FOLLOW A ZIPF DISTRIBUTION.

Traces	TIF	Range Query Number			Top-k Query Number		
		1000	2000	3000	1000	2000	3000
HP	150	93.2	90.6	87.9	96.2	95.1	94.3
	350	90.6	88.2	87.8	93.5	92.1	90.8
	450	87.4	85.3	82.6	91.2	89.6	86.7
MSN	150	94.5	93.2	91.9	98.7	97.9	94.5
	350	92.8	91.1	88.7	96.3	95.2	93.1
	450	91.7	90.6	88.4	93.1	92.5	89.7
EECS	150	95.5	94.4	93.5	98.6	97.2	96.3
	350	93.5	92.2	90.9	97.1	94.8	93.5
	450	90.6	88.7	85.2	94.2	92.6	92.2

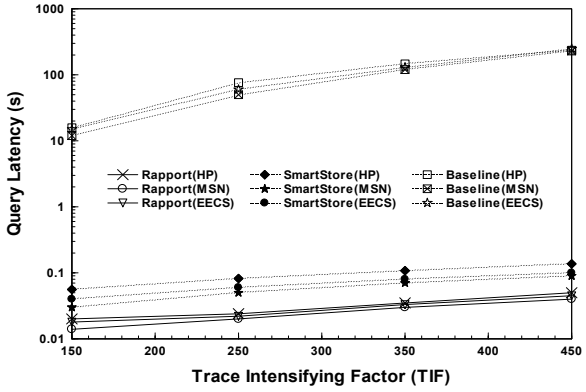


Fig. 10. Point query latency.

TABLE VI
COMPARISONS OF RAPPORT, SMARTSTORE AND BASELINE IN TERMS OF THE RANGE AND TOP-K QUERY LATENCY (IN SECONDS).

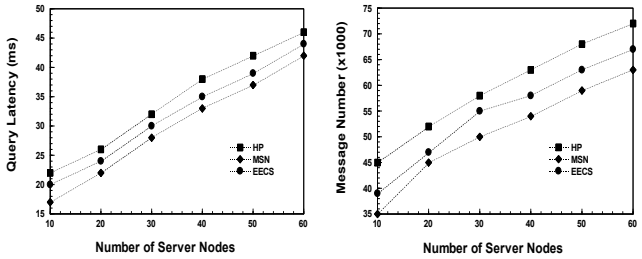
Traces	TIF	Range Query			Top-k Query		
		Rapport	SmartStore	Baseline	Rapport	SmartStore	Baseline
HP	150	0.11	4.12	964.2	0.26	4.79	1035.6
	350	0.35	10.61	3027.5	0.82	12.37	3581.6
	450	0.62	21.57	6281.9	1.57	29.83	8736.1
MSN	150	0.08	3.06	725.3	0.15	3.65	976.2
	350	0.27	8.52	2138.5	0.62	11.73	3056.8
	450	0.46	18.61	5432.7	1.26	25.83	7125.3
EECS	150	0.06	2.76	648.1	0.11	3.02	749.2
	350	0.21	7.58	1947.6	0.47	9.52	2681.9
	450	0.39	16.47	4865.2	0.51	20.86	6351.5

E. System Scalability

We examine system scalability by measuring the average query latency and number of required network messages as a function of the system size when executing 1500 requests composed of 500 point, 500 range and 500 top-k queries under the traces intensified by a TIF factor of 350, as show in Figure 11.

We observe that the average query latencies are 38.6ms, 35.2ms and 36.9ms, respectively for the HP, MSN and EECS

traces as shown in Figure 11(a). The latency measure scales steadily and smoothly with the total number of server nodes that increases from 10 to 60. Rapport 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 the system has only very limited impact on the organization of the correlated files, thus resulting in strong system scalability. In addition, Figure 11(b) shows that the number of messages required for query services also scales steadily and smoothly with the number of server nodes. Since query operations run within one or a very small number of correlated file groups, Rapport often avoids probing any irrelevant nodes and hence reduces network overhead.



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

Fig. 11. System scalability in terms of query latency and network overhead.

VIII. Related Work

Hierarchical directory-based namespace schemes generally impose a performance cost to maintain metadata consistency and avoid false operations/results. Conventional DBMSs often support on-line transaction processing workloads by means of the locking technique [32], [33]. In order to keep replica consistent, FARSITE [34] replicates at the granularity of a "directory group", which resembles a volume, but with a dynamically defined boundary. Both Slice [35] and Archipelago [36] use hash-based approaches to placing groups of objects to certain partitions, restricting the flexibility of these systems. Pangaea [2], [37] maintains the consistency of the namespace in file systems by leveraging pervasive and optimistic replication policies.

Semantic File System (SFS) [4] is one of the first file systems that extend the traditional file system hierarchies by allowing users to navigate the file system while searching file attributes. SFS dynamically creates virtual directories on demand. quFiles [5] is another semantic aware file system that encapsulates different physical representations of the same logical data and selects the best representations depending on the context in which it is accessed. Perspective [6] makes use of *view*, which is a semantic description of a set of files, specified as a query on file attributes, and the ID of the device on which they are stored. 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. DiFFS [3] places files and directories independently on a cluster of servers, where the physical location of files is independent of their position in the namespace hierarchy. In order to replace conventional directory hierarchy, Copernicus [8] provides a searchable namespace by leveraging a dynamic graph-based index, in which it clusters semantically related files into vertexes and allows inter-file relationships to form edges between them. Magellan [9] further builds metadata search functionality directly into the file system and it uses K-D tree, a single data structure, to index all clustered metadata. The rationale there is that files with similar attributes are often clustered together in the namespace. SmartStore [10] uses LSI tool [30] to aggregate semantically correlated files into groups and support complex queries. However, because of LSI's high computational complexity, it is difficult for SmartStore to provide real-time update to changes made to attributes as a result of file system evolution. Although these studies give valuable insights into the current and future searchable file system designs by using the existing locality in sequential accesses, they essentially provide new metadata organization schemes and basically still inherit the fundamentals of the conventional hierarchical directory tree, thus limited in scalability and functionality for ultra-scale file systems.

Our Rapport design that, to the best of our knowledge, is the first work that provides semantic-sensitive namespace management in large-scale file systems. The novelty of our approach lies in the fact that it exploits the file semantic correlation in a multidimensional attribute space to construct a semantic-sensitive namespace for a given file that is small and flat, thus allowing for fast and accurate complex query services and dynamic adaptation to the evolution of file system.

IX. Conclusion

Conventional hierarchical namespace management requires users to navigate a data ocean constructed with billion-scale file hierarchies, potentially leading to misplacement and unacceptable indexing latency. This paper proposes a new namespace scheme, called Rapport, that makes use of semantic correlation to create a semantic-sensitive namespace. The semantic-sensitive namespace of a given file consists of the most closely correlated files and these files are identified by using a simple and fast LSH-based hash computation. Rapport can work on top of any existing file systems that provide multi-dimensional attributes of files. Rapport supports complex queries, such as point, range and top-*k* queries, while exporting the correlated small and flat namespace of queried results. Extensive trace-driven experiments demonstrate the efficiency and effectiveness of our proposed Rapport scheme.

Rapport, different from existing naming schemes, exhibits the correlated semantic-sensitive namespace of queried results that can further hint and inspire users who have no or little knowledge about the large-scale file systems to express what they really want to search and discover and then facilitate their subsequent searches and other file operations. Our future research work will collect and evaluate feedback from a wide range of users. At the same time, we will also investigate the impact of our design to file system users. We will view the correlated namespace as a cache and measure the probability that the next files to be accessed are successfully included in the namespaces of current files.

REFERENCES

[1] R. Daley and P. Neumann, "A general-purpose file system for secondary storage," *Proc. Fall Joint Computer Conference, Part I*, pp. 213–229, 1965.

[2] Y. Saito and C. Karamanolis, "Pangaea: a symbiotic wide-area file system," *Proceedings of the 10th workshop on ACM SIGOPS European workshop*, 2002.

[3] Z. Zhang and C. Karamanolis, "Designing a Robust Namespace for Distributed File Services," *Proc. SRDS*, pp. 162–173, 2001.

[4] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O. Jr, "Semantic file systems," *ACM Symposium on Operating System Principles*, pp. 16–25., 1991.

[5] K. Veeraraghavan, J. Flinn, E. B. Nightingale, and B. Noble, "quFiles: The Right File at the Right Time," *Proc. FAST*, 2010.

[6] B. Salmon, S. W. Schlosser, L. F. Cranor, and G. R. Ganger, "Perspective: Semantic Data Management for the Home," *Proc. FAST*, 2009.

[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. FAST*, 2009.

[8] A. Leung, A. Parker-Wood, and E. L. Miller, "Copernicus: A scalable, high-performance semantic file system," *University of California, Santa Cruz, UCSC-SSRC-09-06*, Oct, 2009.

[9] A. Leung, I. Adams, and E. L. Miller, "Magellan: A searchable metadata architecture for large-scale file systems," *University of California, Santa Cruz, UCSC-SSRC-09-07*, Nov, 2009.

[10] 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," *Proceedings of ACM/IEEE Supercomputing Conference (SC)*, 2009.

[11] N. Agrawal, W. Bolosky, J. Douceur, and J. Lorch, "A Five-Year Study of File-System Metadata," *Proc. FAST*, 2007.

[12] S. Doraimani and A. Iamnitchi, "File Grouping for Scientific Data Management: Lessons from Experimenting with Real Traces," *Proc. HPDC*, 2008.

[13] A. Leung, S. Pasupathy, G. Goodson, and E. Miller, "Measurement and analysis of large-scale network file system workloads," *Proc. USENIX Conference*, 2008.

[14] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," *Proc. ACM symposium on Theory of computing*, pp. 604–613, 1998.

[15] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," *Proc. Annual Symposium on Computational Geometry*, pp. 253–262, 2004.

[16] Y. Tao, K. Yi, C. Sheng, and P. Kalnis, "Quality and Efficiency in High-dimensional Nearest Neighbor Search," *Proc. SIGMOD*, 2009.

[17] A. Andoni and P. Indyk, "Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions," *Communications of the ACM*, no. 1, pp. 117–122, 2008.

[18] Y. Hua, B. Xiao, D. Feng, and B. Yu, "Bounded LSH for Similarity Search in Peer-to-Peer File Systems," *Proc. ICPP*, pp. 644–651, 2008.

[19] A. Andoni and P. Indyk, "Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions," *Proc. FOCS*, pp. 459–468, 2006.

[20] R. Motwani, A. Naor, and R. Panigrahy, "SLower bounds on locality sensitive hashing," *ACM Symposium on Computational Geometry*, 2006.

[21] A. Guttman, "R-trees: A dynamic index structure for spatial searching," *ACM SIGMOD*, pp. 47–57, 1984.

[22] D. Comer, "The ubiquitous B-tree," *Computing Survey*, 1979.

[23] C. Bohm, S. Berchtold, and D. A. Keim, "Searching in high-dimensional spaces index structures for improving the performance of multimedia databases," *ACM Computing Surveys*, vol. 33, no. 3, pp. 322–373, 2001.

[24] A. Traeger, E. Zadok, N. Joukov, and C. Wright, "A nine year study of file system and storage benchmarking," *ACM Transactions on Storage*, no. 2, pp. 1–56, 2008.

[25] E. Riedel, M. Kallahalla, and R. Swaminathan, "A framework for evaluating storage system security," *Proc. 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 International Symposium on Workload Characterization (IISWC)*, 2008.

[27] D. Ellard, J. Ledlie, P. Malkani, and M. Seltzer, "Passive NFS Tracing of Email and Research Workloads," *Proc. FAST*, pp. 203–216, 2003.

[28] Y. Hua, Y. Zhu, H. Jiang, D. Feng, and L. Tian, "Scalable and Adaptive Metadata Management in Ultra Large-scale File Systems," *Proc. ICDCS*, 2008.

[29] D. Comer, "The ubiquitous B-tree," *ACM Comput. Surv.*, vol. 11, no. 2, pp. 121–137, 1979.

[30] C. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala, "Latent Semantic Indexing: A Probabilistic Analysis," *Journal of Computer and System Sciences*, vol. 61, no. 2, pp. 217–235, 2000.

[31] B. Piwowarski and G. Dupret, "Evaluation in (XML) information retrieval: Expected precision-recall with user modelling (EPRUM)," *Proc. SIGIR*, pp. 260–267, 2006.

[32] M. Stonebraker, S. Madden, D. Abadi, S. Harizopoulos, N. Hachem, and P. Helland, "The end of an architectural era:(it's time for a complete rewrite)," *Proc. VLDB*, pp. 1150–1160, 2007.

[33] M. Astrahan, M. Blasgen, D. Chamberlin, K. Eswaran, J. Gray, P. Griffiths, W. King, R. Lorie, P. McJones, J. Mehl, *et al.*, "System R: relational approach to database management," *ACM Transactions on Database Systems (TODS)*, vol. 1, no. 2, pp. 97–137, 1976.

[34] A. Adya, W. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. Douceur, J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer, "FARSITE: Federated, available, and reliable storage for an incompletely trusted environment," *ACM SIGOPS Operating Systems Review*, vol. 36, 2002.

[35] D. Anderson, J. Chase, and A. Vahdat, "Interposed request routing for scalable network storage," *Proc.OSDI*, 2000.

[36] M. Ji, E. Felten, R. Wang, and J. Singh, "Archipelago: an island-based file system for highly available and scalable Internet services," *Proc. of the 4th USENIX Windows Systems Symposium*, 2000.

[37] Y. Saito, C. Karamanolis, M. Karlsson, and M. Mahalingam, "Taming aggressive replication in the Pangaea wide-area file system," *Proc. OSDI*, 2002.