Semantic-Aware Metadata Organization Paradigm in Next-Generation File Systems

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

Abstract—Existing data storage systems based on the hierarchical directory-tree organization do not meet the scalability and functionality requirements for exponentially growing data sets and increasingly complex metadata queries in large-scale, Exabyte-level file systems with billions of files. This paper proposes a novel decentralized semantic-aware metadata organization, called SmartStore, which exploits semantics of files' metadata to judiciously aggregate correlated files into semantic-aware groups by using information retrieval tools. The key idea of SmartStore is to limit the search scope of a complex metadata query to a single or a minimal number of semantically correlated groups and avoid or alleviate brute-force search in the entire system. The decentralized design of SmartStore can improve system scalability and reduce query latency for complex queries (including range and top-k queries). Moreover, it is also conducive to constructing semantic-aware caching, and conventional filename-based point query. We have implemented a prototype of SmartStore and extensive experiments based on real-world traces show that SmartStore significantly improves system scalability and reduces query latency over database approaches. To the best of our knowledge, this is the first study on the implementation of complex queries in large-scale file systems.

Index Terms—File systems, metadata management, scalability, performance evaluation.

1 Introduction

Fast and flexible metadata retrieving is a critical requirement in the next-generation data storage systems serving high-end computing. As the storage capacity is approaching Exabytes and the number of files stored is reaching billions, directory-tree based metadata management widely deployed in conventional file systems [1] can no longer meet the requirements of scalability and functionality. For the next-generation large-scale storage systems, new metadata organization schemes are desired to meet two critical goals: 1) to serve a large number of concurrent accesses with low latency and 2) to provide flexible I/O interfaces to allow users to perform advanced metadata queries, such as range and top-k queries, to further decrease query latency.

Although existing distributed database systems can work well in some real-world data-intensive applications, they are inefficient in very large-scale file systems due to four main reasons. First, as the storage system is scaling up rapidly, a very large-scale file system, the main concern of this paper, generally consists of thousands of server nodes, contains trillions of files, and reaches exabyte-data-volume (EB). Unfortunately, existing distributed databases fail to achieve efficient management of petabytes of data and thousands of concurrent requests [2]. Second, for heterogeneous execution environments, devices of file systems are heterogeneous, such as supercomputers, clusters of PCs via Ethernet, InfiniBand and Fibers, and cloud storage via Internet. Instead, DBMS often assumes homogeneous and dedicated high-performance hardware devices. Recently, the database research community has become aware of this problem and agreed that existing DBMS for general-purpose applications would not be a "*one size fit all*" solution [3]. This issue has also been observed by file system researchers [4]. Third, for heterogeneous data types, their metadata in file systems are also heterogeneous. The metadata may be structured, semistructured, or even unstructured, since they come from different operational system platforms and support various real-world applications. This is often ignored by existing database solutions. Last but not the least, existing file systems only provide filename-based interface and allow users to query a given file, which severely limits the flexibility and ease of use of file systems.

In the next-generation file systems, metadata accesses will very likely become a severe performance bottleneck as metadata-based transactions not only account for over 50 percent of all file system operations [5] but also result in billions of pieces of metadata in directories. Given the sheer scale and complexity of the data and metadata in such systems, we must seriously ponder a few critical research problems [6] such as "*How to efficiently extract useful knowledge from an ocean of data?*," "*How to manage the enormous number of files that have multidimensional or increasingly higher dimensional attributes?*," and "*How to effectively and expeditiously extract small but relevant subsets from large data sets to construct accurate and efficient data caches to facilitate high-end and complex applications?*." We approach the above problems by first postulating the following:

- *Y. Hua and D. Feng are with Wuhan National Laboratory for Optoelectronics, the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China. E-mail: {csyhua, dfeng}@hust.edu.cn.*
- *H. Jiang and L.Tian are with the Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588-0150. E-mail: {jiang, tian}@cse.unl.edu.*
- *Y. Zhu is with the Department of Electrical and Computer Engineering, University of Maine, Room 271, 5708 Barrows Hall, Orono, ME 04469-5708. E-mail: zhu@eece.maine.edu.*

Manuscript received 15 Oct. 2010; revised 19 Apr. 2011; accepted 2 May 2011; published online 25 May 2011.
Recommended for acceptance by Y. Hu.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2010-10-0613.
Digital Object Identifier no. 10.1109/TPDS.2011.169.

First, while a high-end or next-generation storage system can provide a Petabyte-scale or even Exabyte-scale storage capacity containing an ocean of data, what the users really want for their applications is some knowledge about the data's behavioral and structural properties. Thus, we need to deploy and organize these files according to semantic correlations of file metadata in a way that would easily expose such properties.

Second, in real-world applications, cache-based structures have proven to be very useful in dealing with indexing among massive amounts of data. However, traditional temporal or spatial (or both) locality-aware methods alone will not be effective to construct and maintain caches in large-scale systems to contain the working data sets of complex data-intensive applications. It is thus our belief that semantic-aware caching, which leverages metadata semantic correlation and combines preprocessing and prefetching that is based on range queries (that identify files whose attributes values are within given ranges) and top-k Nearest Neighbor (NN) queries (that locate k files whose attributes are closest to given values), will be sufficiently effective in reducing the working sets and increasing cache hit rates.

Semantic correlation [7] comes from the exploitation of high-dimensional attributes of metadata. The main benefit of using semantic correlation is the ability to significantly narrow the search space and improve system performance. In this paper, we propose a novel decentralized semantic-aware metadata organization, called *SmartStore* [8], to effectively exploit semantic correlation to enable efficient complex queries for users and to improve system performance in real-world applications. This paper makes the following key contributions:

Decentralized semantic-aware organization scheme of file system metadata. SmartStore is designed to support complex query services and improve system performance by judiciously exploiting semantic correlation of file metadata and effectively utilizing semantic analysis tools, i.e., Latent Semantic Indexing (LSI) [9]. The new design is different from the conventional hierarchical architecture of file systems based on a directory-tree data structure in that it removes the latter's inherent performance bottleneck and thus can avoid its disadvantages in terms of file organization and query efficiency. Additionally and importantly, SmartStore is able to provide the existing services of conventional file systems while supporting new complex query services with high reliability and scalability. Our experimental results based on a SmartStore prototype implementation show that its complex query performance is more than 1,000 times higher and its space overhead is 20 times smaller than current database methods with a very small false probability.

Multiquery services. To the best of our knowledge, this is the first study on the design and implementation of a storage architecture that supports complex queries, such as range and top-k queries, within the context of ultra-large-scale distributed file systems. More specifically, our SmartStore can support three query interfaces for point, range, and top-k queries. Conventional query schemes in small-scale file systems are often concerned with filename-based queries that will soon be rendered inefficient and ineffective in next-generation large-scale distributed file systems. The complex queries will serve as an important portal or browser, like the web or web browser for Internet and city map for a tourist, for query services in an ocean of files. Our study is a first attempt at providing support for complex queries directly at the file system level.

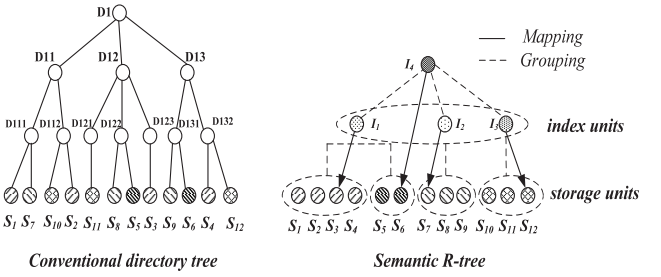


Fig. 1. Comparisons with conventional file system.

The rest of the paper is organized as follows: Section 2 describes the SmartStore system design. Section 3 presents semantic grouping. The version-based design is discussed in Section 4. The performance of SmartStore is evaluated through extensive trace-driven experiments in Section 5. Section 6 presents related work. The paper is concluded in Section 7.

2 SmartStore System

The basic idea behind SmartStore is that files are grouped and stored according to their metadata semantics, instead of directory namespace, as shown in Fig. 1 that compares the two schemes. This is motivated by the observation that metadata semantics can guide the aggregation of highly correlated files into groups that in turn have higher probability of satisfying complex query requests, judiciously matching the access pattern of locality. Thus, query and other relevant operations can be completed within one or a small number of such groups, where one group may include several storage nodes, other than linearly searching via brute force on almost all storage nodes in a directory namespace approach. On the other hand, the semantic grouping can also improve system scalability and avoid access bottlenecks and single-point failures since it renders the metadata organization fully decentralized whereby most operations, such as insertion/deletion and queries, can be executed within a given group.

2.1 Overview

A semantic R-tree as shown on the right of Fig. 1 is evolved from classical R-tree [10] and consists of index units (i.e., nonleaf nodes) containing location and mapping information and storage units (i.e., leaf nodes) containing file metadata, both of which are hosted on a collection of storage servers. One or more R-trees may be used to represent the same set of metadata to match query patterns effectively. SmartStore supports complex queries, including range and top-k queries, in addition to simple point query. Fig. 2 shows a logical diagram of SmartStore that provides multiquery services for users while organizes metadata to

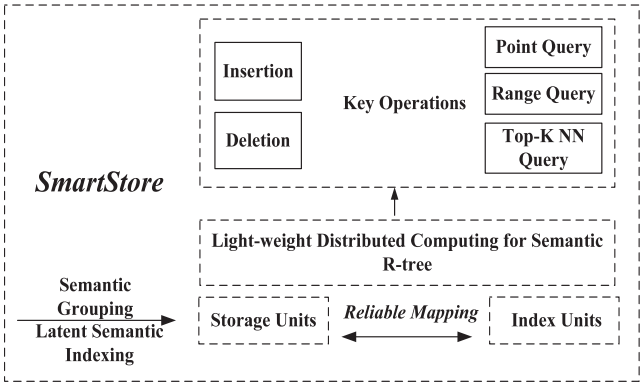


Fig. 2. SmartStore system diagram.

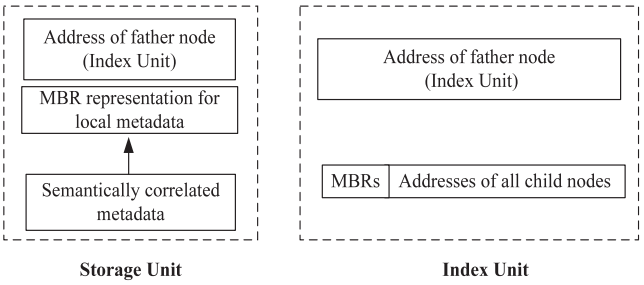


Fig. 3. Storage and index units.

enhance system performance by using decentralized semantic R-tree structures.

SmartStore has three key functional components: 1) the grouping component that classifies metadata into storage and index units based on the LSI semantic analysis; 2) the construction component that iteratively builds semantic R-trees in a distributed environment; and 3) the service component that supports insertion, deletion in R-trees, and multiquery services.

2.2 User View

A query in SmartStore works as follows: initially, a user sends a query to a randomly chosen storage unit, i.e., a leaf node of semantic R-tree. The chosen storage unit, called *home* unit for this request, then retrieves semantic R-tree to locate the corresponding R-tree node. Specifically, for a point query, the home unit checks Bloom filters [11] stored locally in a way similar to the group-based hierarchical Bloom-filter array approach [12] and, for a complex query, the home unit checks the Minimum Bounding Rectangles (MBR) [10] to determine the membership of queried file within checked servers. An MBR represents the minimal rectangle of the enclosed data set by using multidimensional intervals of the attribute space, showing the lower and the upper bounds of each dimension. After obtaining query results, the home unit returns them to the user.

2.3 System View

The most critical component in SmartStore is semantic grouping, which efficiently exploits metadata semantics, such as file physical and behavioral attributes, to classify files into groups iteratively. These attributes exhibit different characteristics. For example, attributes such as access frequency, file size, volume of "read" and "write" operations are changed frequently, while some other attributes, such as filename and creation time, often remain unchanged. SmartStore identifies the correlations between different files by examining these and other attributes, and then places strongly correlated files into groups. All groups are then organized into a semantic R-tree. These groups may reside in multiple metadata servers. By grouping correlated metadata, SmartStore exploits their affinity to boost the performance of queries.

Fig. 3 shows the basic steps in constructing a semantic R-tree. Each metadata server is a leaf node in our semantic R-tree and can also potentially hold multiple nonleaf nodes of the R-tree. In the rest of the paper, we refer to the semantic R-tree leaf nodes as *storage units* and the nonleaf nodes as *index units*.

2.4 Configuration to Match Query Patterns

The objective of the semantic R-tree constructed by examining the semantic correlation of metadata attributes is to match the patterns of complex queries from users. Unfortunately, in real-world applications, the queried attributes will likely exhibit unpredictable characteristics, meaning that a query request may probe an arbitrary d-dimensional $(1 \leq d \leq D)$ subset of D-dimensional metadata attributes. For example, we can construct a semantic R-tree by leveraging three attributes, i.e., *file size, creation time, and last modification time*, and then queries may search files according to their *file size*, *file size and creation time*, or other combinations of these three attributes. Although using a single semantic R-tree can eventually lead to the queried files, the system performance can be greatly reduced as a result of more frequently invoking the brute-force-like approach after each failed R-tree search. The main reason is that a single semantic R-tree representing three attributes may not work efficiently if queries are generated in an unpredictable way.

In order to efficiently support complex queries with unpredictable attributes, we develop an *automatic configuration* technique to adaptively construct one or more semantic R-trees to improve query accuracy and efficiency. More R-trees with each being associated with a different combination of multidimensional attributes provide much better query performance, but require more storage space. The automatic configuration technique thus must optimize the trade-off between storage space and query performance. Our basic idea is to configure one or more semantic R-trees to adaptively satisfy complex queries associated with an arbitrary subset of attributes.

Assume that D is the maximum number of attributes in a given file system. The automatic configuration first constructs a semantic R-tree according to the available D-dimensional attributes to group file metadata, and counts the number of index units, $NO(I_D)$, generated in this R-tree. It then constructs another semantic R-tree using a subset (i.e., d attributes) and records the number of generated index units, $NO(I_d)$. When the difference in the number of index units between the two semantic R-trees, $|NO(I_D) - NO(I_d)|$, is larger than some predetermined threshold, we conjecture that these two semantic R-trees are sufficiently different, and thus are saved to serve future queries.

3 Semantic-Aware Design and Analysis

3.1 Semantic Grouping

Statement 1 (Semantic Grouping of Metadata). *Given file metadata with D attributes, find a subset of d attributes $(1 \leq d \leq D)$, representing special interests, and use the correlation measured in this subset to partition similar file metadata into multiple groups so that:*

- *A file in a group has a higher correlation with other files in this group than with any file outside of the group.*
- *Group sizes are approximately equal.*

Semantic grouping is an iterative process. In the first iteration, we compute the correlation between files and cluster all files whose correlations are larger than a predetermined admission constant $\varepsilon_1 (0 \leq \varepsilon_1 \leq 1)$ into groups. The admission constant depends on different distributions of file data. We use a sampling method to facilitate the group construction. Specifically, we first randomly choose data from traces to compose a sample data set. By examining the data set, we then adjust various admission constants to select a near-optimal value to guarantee approximate load balance.

All groups generated in the first iteration are used as leaf nodes to construct a semantic R-tree. The composition of the selected d-dimensional attributes produces a `grouping predicate`, which serves as the grouping criteria. The semantic grouping process can be recursively executed by aggregating groups in the $(i-1)$st-level into the ith-level nodes of the semantic R-tree with the correlation value $\varepsilon_i (0 \leq \varepsilon_i \leq 1, 1 \leq i \leq H)$, until the root is reached, where H is the depth of the constructed R-tree.

More than one predicate may be used to construct semantic groups. Thus, multiple semantic R-trees can be obtained and maintained concurrently in a distributed manner in a large-scale distributed file system where most files are of interests to arguably only one or a small number of applications or application environments. In other words, each of these semantic R-trees may possibly represent a different application environment or scenario. Our objective is to identify a set of predicates that optimize the query performance.

In the semantic R-tree, each node represents all metadata that can be accessed through its child nodes. Each node can be summarized by a geometric centroid of all metadata it represents. The attributes used to form semantic vectors can be either physical ones, such as creation time and file size, or behavioral ones, such as process ID and access sequence.

3.2 Probabilistic Latent Semantic Analysis (PLSA) for Complexity Reduction

The basic idea behind Probabilistic Latent Semantic Analysis [13] is to execute a mixture decomposition derived from a latent class model, rather than performing an SVD operation over co-occurrence tables based on linear algebra like Latent Semantic Indexing [9]. In order to avoid overfitting, a statistic model based on the Expectation Maximization (EM) algorithm [14], [15] is used to guarantee the maximum likelihood estimation. The EM algorithm computes posterior probabilities for latent variables with expected E steps and updates them within maximum M steps.

The difference in computational complexity between LSI and PLSA is mainly due to the fact that different objective functions are used to determine the optimal decomposition/approximation. LSI makes use of the L2 or Frobenius norm, which corresponds to an implicit additive Gaussian noise assumption on counts, to facilitate the squared deviation on the matrix. On the other hand, PLSA uses the likelihood function of multinomial sampling to show an explicit maximization of the predictive power of the latent space model. This technique allows PLSA to minimize the cross entropy or Kullback-Leibler divergence between the empirical distribution and the used model, thus significantly decreasing the computational complexity [16]. Note that PLSA can reduce the computational complexity and improve runtime performance and obtain approximate accuracy as LSI. We compare their performance in terms of semantic-aware system construction in Section 5.

3.3 Complexity Analysis

The computational complexities of the *offline* and *online* system constructions are significantly different in our design. The offline method builds the semantic groups in advance, while its online counterpart builds the semantic groups dynamically and allows real-time updates.

The complexity of offline LSI mainly stems from the singular value decomposition. Given certain constraints of computational precision, LSI needs to calculate the optimal number of iteration steps based on the reduced z-dimensional square matrix, thus resulting in z iterations. In our experiments, we found that between 20 and 80 iterations are usually sufficient. Within each iteration, matrix decomposition of the correlation vectors for n files takes $O(n^2)$. Therefore, the initial system construction requires $O(zn^2)$. In practice, the typical value of z is between 3 and 10, much smaller than the number of rows or columns of the original attribute matrix.

The complexity of offline PLSA mainly depends on the calculation of E-step and M-step. Each E-step requires the computation of the optimal probability, which consists of γ numbers, and a constant number of arithmetic operations to be computed on n files, thus resulting in a complexity of $O(\gamma n)$. The M-step accumulates the posterior probability for each rating to facilitate the new estimate, thus requiring a complexity of $O(\gamma n)$. Therefore, the PLSA approach has a computational complexity of $O(\gamma n)$.

The online computational complexity is more important in real systems. In the standard LSI approach, each arriving request, represented as a query vector, requires the identification of the correlation with n existing files by using either the angle (e.g., cosine similarity) or distance (e.g., euclidean) measure, thus leading to a complexity of $O(n)$. In practice, many applications would identify the most correlated item by comparing the correlation degrees of all n items, which also results in a complexity of $O(n)$. Thus, the computational complexity of online LSI is $O(n)$. On the other hand, PLSA requires the computation of the prediction probability in the statistical model, amounting to 2γ arithmetic operations. The value of γ is independent of the numbers of n files and m attributes. As a result, the complexity of online PLSA is $O(\gamma)$.

4 Consistency Guarantee via Versioning

In SmartStore, a recently created version attached to its correlated replica can temporarily maintain the aggregated changes that however have not updated the corresponding

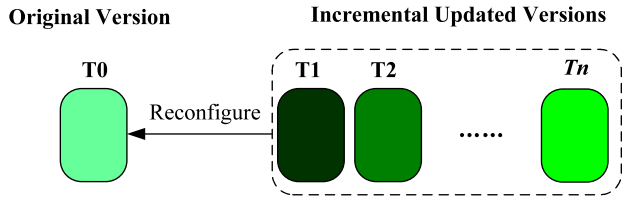


Fig. 4. Multiversion structure.

original replica. This method eliminates many small, random, and frequent visits to the index and has been widely used in most versioning file systems [4], [17]. In the following, we first present the *standard* scheme for multiversion management and then the *improved* solution to obtain space efficiency and support fast update.

4.1 Multiversion Management

In order to maintain both semantic correlation and locality, SmartStore creates versions for every group, represented as the first-level index unit that has been replicated to other index units. At the time instant t_0, SmartStore sends the replicas of the original index units to other index units and from t_{i-1} to t_i, updates are aggregated into the t_ith version that is attached to its correlated index unit as shown in Fig. 4. These updates include insertion, deletion, and modification of file metadata, which are appropriately labeled in the versions. In order to adapt to the system changes, SmartStore allows the groups to have different numbers and sizes of attached versions.

Versioning may introduce extra performance overhead due to the need to check on the attached versions in addition to the original information when executing a query. However, since the versions only maintain changes that require small storage overheads and can be fully stored in memory, the extra latency of searching is usually small. In practice, we propose to roll the version changes backward, rather than forward as in Spyglass [4]. A query first checks the original information and then its versions from t_i backward to t_0. The direct benefit of checking backward is to timely obtain most recent changes since version t_i usually contains newer information than version t_{i-1}.

SmartStore removes attached versions when reconfiguring index units. The frequency of reconfiguration depends on the user requirements and environment constraints. Removing versions entails two operations. We first apply the changes of a version into its attached original index unit that will be updated according to these changes in the attached versions, such as inserting, deleting, or modifying file metadata. On the other hand, the version is also multicast to other remote index units that have stored the replicas of original index unit, and then these remote index units carry out the similar operations for local updating. Since the attached versions only need to maintain changes of file metadata and maintain small size, SmartStore may multicast them as replicas to other remote servers to guarantee information consistency while requiring not too much bandwidth to transmit small-size changes as shown in Section 5.4.

4.2 Space-Efficient Improvements

SmartStore improves the space efficiency of multiversion scheme by leveraging space-efficient Bloom filters to decrease space overhead and provide fast identification of stale versions. The basic idea is to maintain a single copy of

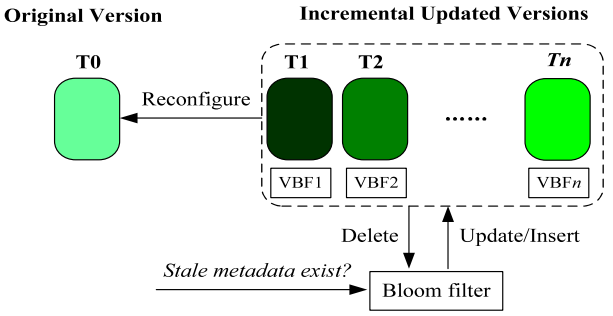


Fig. 5. Improved versioning structure.

metadata of each updated file in the version. Fig. 8 shows the improved versioning structure that consists of original and incremental versions together with space-efficient Bloom filters to identify stale data.

Compared with the standard versioning as previously shown in Fig. 4, the improved versioning structure in Fig. 5 deploys two-level Bloom filters to carry out multiversion management and support fast query services. Specifically, the first-level Bloom filter is a counting Bloom filter [18] that uses the counters, rather than bits in standard Bloom filter [11], to support deletion operations. Newly updated data are first hashed into the counters in the counting Bloom filters. If all hit counters are nonzero, we can say that the data are a member of multiple versions with high probability, meaning that there is the stale copy of the newly arriving data.

Although the membership queries on the counting Bloom filters do not produce false negatives due to local data updates, hash collisions in Bloom filters can possibly result in false positives. The penalty of a false hit is a waste of probing operation on the second-level Bloom filters, called Version Bloom Filter (VBF). Each VBF corresponding to a version is a counting Bloom filter and maintains the memberships of data stored in that version.

After the first-level, Bloom filter indicates that there is the stale copy in the multiversion structure, the update data then probe the VBFs to identify which version contains the stale copy to facilitate further updating. Once the stale version is identified, we will update it with new data. Therefore, the final verification on the versions occurs when both levels of Bloom filters produce false positives. However, such probability is very small. On the other hand, if the first-level counting Bloom filter indicates that the updated data have no stale copies in the versions, we then directly insert them into the versions according to their temporal sequence. In addition, when a data item is deleted from a version, we decrease the corresponding counters by 1 in the two-level Bloom filters. If the incremental updated versions are reconfigured into the baseline version, all metadata are removed in the updated versions and we thus initialize the two-level Bloom filters by setting all counters to zero.

The versioning system essentially introduces the extra checking on the versions when carrying out query operations since the versions may contain the latest data. We use the Bloom filters to simplify the point queries by first hashing queried items into the Bloom filters and if both levels of Bloom filters say hits, we then check the incremental updated versions and if not, the queried results only come from the baseline versions. On the other hand, the Bloom filters, which only support point queries, do not work for complex queries, such as the range and top-k

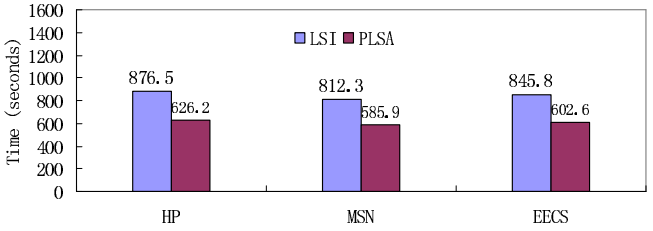


Fig. 6. Initial construction time by using LSI and PLSA.

queries that require the comparisons on range bounds. We thus carry out the standard operations as previously presented in Section 4.1.

5 Performance Evaluation

This section evaluates SmartStore through its prototype by using representative large file system-level traces, including *HP* [19], *MSN* [20], and *EECS* [21]. We compare SmartStore against two baseline systems that use database techniques. The evaluation metrics considered are query accuracy, query latency, and communication overhead. Due to space limitation, additional performance evaluation results are omitted but can be found in our conference paper [8], including recalls of range and top-k queries using *MSN* and *EECS*, average query latency, and message overhead based on preprocessing approach, grouping efficiency, and system scalability.

5.1 Prototype Implementation

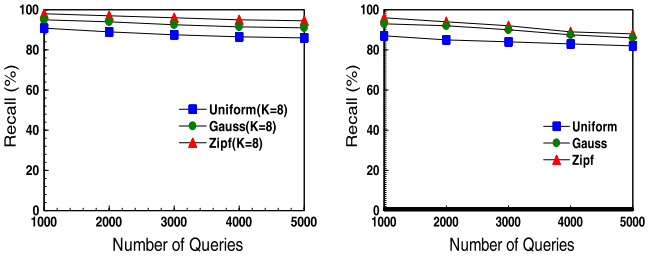
The SmartStore prototype is implemented in Linux and our experiments are conducted on a cluster of 60 storage units. Each storage unit has an Intel Core 2 Duo CPU, 2 GB memory, and high-speed network connections. We carry out the experiments for 30 runs each to validate the results according to the evaluation guidelines of file and storage systems [22]. The used attributes display access locality and skewed distribution especially for multi-dimensional attributes.

5.2 Construction Time

SmartStore needs to first build the offline semantic-aware system by using standard LSI and PLSA to support online complex queries as illustrated in Section 3. Recall that the LSI carries out the SVD operations on the original attribute matrix to identify the correlation among multiple files and the PLSA makes use of E-M algorithm to facilitate the maximized likelihood estimation to decrease the computation complexity. Fig. 6 shows the initial construction time of SmartStore system when taking into account three intensified real-world traces. We observe that the PLSA spends much less time than LSI, with an average reduction of 28.3 percent. The main reason is that PLSA deploys probabilistic model to extract the semantic correlation, thus avoiding the high complexity of matrix decomposition in LSI. These experimental results are consistent with the complexity analysis presented in Section 3.3.

5.3 Accuracy of Complex Queries

We adopt "Recall" as a measure for complex query quality from the field of information retrieval [23]. Given a query q, we denote $T(q)$ the ideal set of K nearest objects and $A(q)$ the actual neighbors reported by SmartStore. We define $recall$ as $recall = \frac{|T(q) \cap A(q)|}{T(q)}$.



(a) Top-8 NN query. (b) Range query.

Fig. 7. Recall of complex queries using *HP* trace.

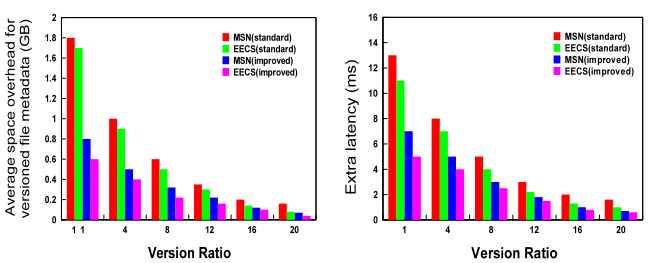
Fig. 7 shows recall values of complex queries, including range and top-k ($k = 8$) queries, for the *HP* trace. We observe that a top-k query generally achieves higher recall than a range query. The main reason is that top-k query in essence is a similarity search, thus targeting a relatively smaller number of files. We also notice that requests following a Zipf or Gauss distribution obtain much higher recall values than those following a uniform distribution. This is because under a Zipf or Gauss distribution, files are mutually associated with a higher degree than under uniform distribution.

5.4 Overhead and Efficiency of Versioning

Using versioning to maintain consistency among multiple replicas of the root and index nodes in the semantic R-tree, as described in Section 4, introduces some extra spatial and temporal costs.

Similar to evaluating the versioning file systems [17], we adjust the version ratio, i.e., file modification-to-version ratio, to examine the overhead introduced by versioning. Fig. 8 shows the versioning overhead in terms of required space and latency when checking the versions. Due to space limit, this paper only presents the performances under the MSN and EECS traces.

Fig. 8a shows the average required space in each index unit by using standard and improved versioning schemes. The space overhead is closely associated with the version ratio. If the ratio is 1, it is called a comprehensive versioning, and every change results in a version, thus requiring the largest storage space. When the ratio is increased, changes usually are aggregated to produce a version to reduce space overhead. We observe that the improved scheme requires much smaller space, averagely 41.25 percent decrements, than the standard solution. The main reason is that the former maintains a single copy of updated data in the



(a) Space overhead. (b) Extra latency.

Fig. 8. Versioning overhead in space and access latency.

incremental versions before configuring into the corresponding baseline version with the aid of fast checking of two-level Bloom filters, while the latter simply accumulates the update versions in the temporal sequence. In addition, although the used Bloom filters possibly increase the space overhead in the improved scheme, the larger benefits of keeping a single copy make the extra space overhead be trivial.

Fig. 8b shows the extra latency incurred when verifying query results in the versions. Note that we use the same number of point, range, and top-k queries. Compared with the entire query latency, the additional versioning latency is no more than 10 percent. The reason is that all versions only need to record small changes stored in memory and we use rolling backward to reduce unnecessary checking on stale information. In addition, the improved scheme significantly decreases the query latency on average by 21.6 percent, compared with the standard solution, since the point query only needs to execute fast hash computation.

6 Related Work

One of the most prevalent metadata queries is content-based query by examining the contents and pathnames of files, such as attribute-based naming in the Semantic file system [24] and content-based search tool in Google Desktop [25]. However, the efficiency of content-based search heavily depends on files that contain explicitly understandable contents, while ignoring file context that is utilized by most users in organizing and searching their data [26]. Furthermore, typical techniques successful for the web search, such as HITS algorithm [27] and Google search engine [28], leverage tagged and contextual links that do not inherently, let alone explicitly, exist in large-scale file systems.

Subtree-partitioning-based approaches have been widely used in recent studies, such as Ceph [1], GIGA+ [29], Farsite [30], and Spyglass [4]. Ceph [1] maximizes the separation between data and metadata management by using a pseudorandom data distribution function to support a scalable and decentralized placement of replicated data. Farsite [30] makes the improvement on distributed directory service by utilizing tree-structured file identifiers that support dynamically partitioning on metadata at arbitrary granularity. GIGA+ [29] extends classic hash tables to build file system directories and uses bitmap encoding to allow hash partitions to split independently, thus obtaining high update concurrency and parallelism. Spyglass [4] 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. However, in its current form, Spyglass focuses on the indexing on a single server and cannot support distributed indexing on multiple servers.

7 Conclusion

The paper presents a new paradigm for organizing file metadata for next-generation file systems, called SmartStore, by exploiting file semantic information to provide efficient and scalable complex queries while enhancing system scalability and functionality. The novelty of SmartStore lies in it matches actual data distribution and physical layout with their logical semantic correlation so that a complex query can be successfully served within one or a small number of storage units. Specifically, a semantic grouping method is proposed to effectively identify files that are correlated in their physical attributes or behavioral attributes. SmartStore can very efficiently support complex queries, such as range and top-k queries, which will likely become increasingly important in the next-generation file systems. Our prototype implementation proves that SmartStore is highly scalable, and can be deployed in a large-scale distributed storage system with a large number of storage units.

Acknowledgments

This work was supported in part by National Natural Science Foundation of China (NSFC) under Grant 61173043, 61025008; National Basic Research 973 Program of China under Grant 2011CB302301; NSFC under Grant 60703046; Fundamental Research Funds for the central universities, HUST, under grant 2010MS043; National High Technology Research and Development 863 Program of China under Grant 2009AA01A401 and 2009AA01A402; the Program for Changjiang Scholars and Innovative Research Team in University under Grant IRT-0725; US National Science Foundation (NSF) under Grants NSF-CNS-1016609, NSF-CCF-0621526, NSF-CCF-0937993, NSF-CNS-1116606, NSF-CNS-1117032, NSF-IIS-0916859, NSF-IIS-0916663, NSF-CCF-0937988, NSF-CCF-0621493, NSF-CDI-1027809, and NSF-EPS-0904155. The authors greatly appreciate anonymous reviewers for constructive comments.

References

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

[2] Y. Hua, Y. Zhu, H. Jiang, D. Feng, and L. Tian, "Supporting Scalable and Adaptive Metadata Management in Ultralarge-Scale File Systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 22, no. 4, pp. 580-593, Apr. 2011.

[3] M. Stonebraker and U. Cetintemel, "One Size Fits All: An Idea Whose Time Has Come and Gone," *Proc. Int'l Conf. Data Eng. (ICDE)*, 2005.

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

[5] D. Roselli, J. Lorch, and T. Anderson, "A Comparison of File System Workloads," *Proc. USENIX Conf.*, pp. 41-54, 2000.

[6] M. Seltzer and N. Murphy, "Hierarchical File Systems Are Dead," *Proc. Conf. Hot Topics in Operating Systems (HotOS)*, 2009.

[7] D.K. Gifford, P. Jouvelot, M.A. Sheldon, and J.W. OToole, "Semantic File Systems," *Proc. ACM Symp. Operating Systems Principles (SOSP)*, 1991.

[8] 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.

[9] 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.

[10] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, 1984.

[11] B. Bloom, "Space/Time Trade-Offs in Hash Coding with Allowable Errors," *Comm. ACM*, vol. 13, no. 7, pp. 422-426, 1970.

[12] Y. Hua, Y. Zhu, H. Jiang, D. Feng, and L. Tian, "Scalable and Adaptive Metadata Management in Ultra Large-Scale File Systems," *Proc. Int'l Conf. Distributed Computing Systems (ICDCS)*, 2008.
[13] T. Hofmann, "Probabilistic Latent Semantic Indexing," *Proc. Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR)*, 1999.
[14] G. McLachlan and T. Krishnan, *The EM Algorithm and Extensions*. Wiley, 1997.
[15] A. Dempster et al., "Maximum Likelihood from Incomplete Data via the EM Algorithm," *J. Royal Statistical Soc. Series B (Methodological)*, vol. 39, no. 1, pp. 1-38, 1977.
[16] Z. Rached, F. Alajaji, and L. Campbell, "The Kullback-Leibler Divergence Rate between Markov Sources," *IEEE Trans. Information Theory*, vol. 50, no. 5, pp. 917-921, May 2004.
[17] C.A.N. Soules, G.R. Goodson, J.D. Strunk, and G.R. Ganger, "Metadata Efficiency in Versioning File Systems," *Proc. USENIX Conf. File and Storage Technologies (FAST)*, 2003.
[18] L. Fan, P. Cao, J. Almeida, and A. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," *IEEE/ACM Trans. Networking*, vol. 8, no. 3, pp. 281-293, June 2000.
[19] E. Riedel, M. Kallahalla, and R. Swaminathan, "A Framework for Evaluating Storage System Security," *Proc. USENIX Conf. File and Storage Technologies (FAST)*, 2002.
[20] 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.
[21] 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)*, 2003.
[22] A. Traeger, E. Zadok, N. Joukov, and C. Wright, "A Nine Year Study of File System and Storage Benchmarking," *ACM Trans. Storage*, vol. 4, no. 2, pp. 1-56, 2008.
[23] B. Piwowarski and G. Dupret, "Evaluation in (XML) Information Retrieval: Expected Precision-Recall with User Modelling (EPRUM)," *Proc. Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR)*, pp. 260-267, 2006.
[24] D.K. Gifford, P. Jouvelot, M.A. Sheldon, and J.W.O. Jr, "Semantic File Systems," *Proc. ACM Symp. Operating Systems Principles (SOSP)*, 1991.
[25] "Google Desktop," http://desktop.google.com/, 2011.
[26] C. Soules and G. Ganger, "Connections: Using Context to Enhance File Search," *Proc. ACM Symp. Operating Systems Principles (SOSP)*, 2005.
[27] J. Kleinberg, "Authoritative Sources in a Hyperlinked Environment," *J. ACM*, vol. 46, no. 5, pp. 604-632, 1999.
[28] "Google," http://www.google.com/, 2011.
[29] S. Patil and G. Gibson, "Scale and Concurrency of GIGA+: File System Directories with Millions of Files," *Proc. USENIX Conf. File and Storage Technologies (FAST)*, 2011.
[30] J.R. Douceur and J. Howell, "Distributed Directory Service in the Farsite File System," *Proc. Symp. Operating Systems Design and Implementation (OSDI)*, pp. 321-334, 2006.

Yu Hua received the BE and PhD degrees in computer science from Wuhan University, China, in 2001 and 2005, respectively. He joined the Department of Computing at the Hong Kong Polytechnic University as a research assistant in 2006. Now, he is an associate professor in the School of Computer Science and Technology, Huazhong University of Science and Technology, China. His research interests include computer architecture, cloud computing, network storage, and cyberphysical systems. He has more than 20 papers to his credit in major journals and international conferences including *IEEE Transactions on Computers* (TC), *IEEE Transactions on Parallel and Distributed Systems* (TPDS), USENIX ATC, INFOCOM, SC, ICDCS, ICPP, and HiPC. He has been on the program committees of multiple international conferences. He is a member of the IEEE and USENIX.

Hong Jiang received the BSc degree in computer engineering from Huazhong University of Science and Technology, Wuhan, China, in 1982, the MASc degree in computer engineering from the University of Toronto, Canada, in 1987, and the PhD degree in computer science from the Texas A&M University, College Station, in 1991. Since August 1991, he has been at the University of Nebraska-Lincoln (UNL), where he served as the vice chair of the Department of Computer Science and Engineering (CSE) from 2001 to 2007, and is a professor of CSE. At the UNL, he has graduated 10 PhD students, who upon their graduations either landed academic tenure-track positions (e.g., Stevens Institute of Tech., New Mexico Tech., Univ. of Maine, Univ. of Alabama, etc.) or were employed by major US IT corporations (e.g., Microsoft, Seagate, etc.). His present research interests include computer architecture, computer storage systems and parallel I/O, parallel/distributed computing, cluster and grid computing, performance evaluation, real-time systems, middleware, and distributed systems for distance education. He serves as an associate editor of the *IEEE Transactions on Parallel and Distributed Systems*. He has more than 180 publications in major journals and international conferences in these areas, including *IEEE TPDS*, *IEEE TC*, JPDC, USENIX-ATC, ISCA, MICRO, FAST, ICDCS, IPDPS, OOPLAS, ECOOP, SC, ICS, HPDC, ICPP, etc., and his research has been supported by the US National Science Foundation (NSF), DOD, and the State of Nebraska. He is a senior member of the IEEE and a member of the ACM and ACM SIGARCH.

Yifeng Zhu received the BSc degree in electrical engineering from Huazhong University of Science and Technology, Wuhan, China, in 1998, and the MS and PhD degrees in computer science from the University of Nebraska, Lincoln, in 2002 and 2005, respectively. He is currently an associate professor in the Department of Electrical and Computer Engineering at the University of Maine. His research interests include parallel I/O storage systems, supercomputing, energy-aware memory systems, and wireless sensor networks. He served as the program chair of IEEE NAS '09, SNAPI '07, guest editor of a special issue of IJHPCN, and the program committee of various international conferences, including ICDCS, ICPP, and NAS. He received the Best Paper Award at IEEE CLUSTER '07 and served as the PI or co-PI of several research grants awarded by the US National Science Foundation (NSF), including CDI, CCF, CSR, HECURA, ITEST, REU, and MRI. He is a member of the ACM and the Francis Crowe Society, and a senior member of the IEEE..

Dan Feng received the BE, ME, and PhD degrees in computer science and technology from 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* (TPDS), JCST, USENIX ATC, FAST, ICDCS, HPDC, SC, ICS, and ICPP. She is a member of the IEEE.

Lei Tian received the BE degree in computer science and technology from Huazhong University of Science and Technology (HUST), China, in 2001, and the ME and PhD degrees in computer architecture from HUST, in 2004 and 2010, respectively. His research interests include RAID-structured storage systems, distributed storage systems, and large-scale metadata management. He has more than 30 publications to his credit in journals and international conferences including *IEEE TC*, *IEEE TPDS*, *ACM TOS*, FAST, ICS, SC, HPDC, ICDCS, MSST, ICPP, IPDPS, and MASCOTS.