SmartStore: A New Metadata Organization Paradigm with Metadata Semantic-Awareness for Next-Generation File Systems

Yu Hua
Huazhong Univ. of Sci. and Tech.
csyhua@gmail.com

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

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

Dan Feng
Huazhong University of Science and Technology
dfeng@hust.edu.cn

Lei Tian
University of Nebraska-Lincoln
tian@cse.unl.edu

Abstract

Existing data storage systems based on hierarchical directory tree do not meet scalability and functionality requirements for exponentially growing datasets and increasingly complex metadata queries in large-scale file systems with billions of files and Exabytes of data. This paper proposes a novel decentralized semantic-aware metadata organization, called SmartStore, which exploits metadata semantics of files to judiciously aggregate correlated files into semantic-aware groups by using information retrieval tools. The decentralized design of SmartStore can improve system scalability and reduce query latency for both complex queries (including range and top-k queries), which is helpful to construct semantic-aware caching, and conventional filename-based point query. The key idea of SmartStore is to limit search scope of a complex metadata query to a minimal number of semantically related groups and avoid or alleviate brute-force search in entire system. Extensive experiments based on real-world traces show that SmartStore significantly improves system scalability and reduces query latency by more than one thousand times faster than current database approaches. To the best of our knowledge, this is the first paper addressing complex queries in large-scale file systems.

1. Introduction

Fast and flexible metadata retrieving is critical in the next-generation data storage systems. 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 [1–4] 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.

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% of all file system operations [10] 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 [16] such as "*How to efficiently extract useful knowledge from an ocean of data?*", "*How to manage the enormous number of files that have multi-dimensional or increasingly higher dimensional attributes?*", and "*How to effectively and expeditiously extract small but relevant subsets from large datasets to construct accurate and efficient data caches to facilitate high-end and complex applications?*".

For the above problems, first, while a high-end or next-generation storage system can provide a Petabyte-scale or even Exabyte-scale capacity containing an ocean of data, what the users really require to satisfy many of their application needs is some knowledge about the data's behavioral and structural properties that can in turn effectively facilitate the applications. Thus, we need to deploy and organize these files according to semantic correlation of file metadata in a way that would easily expose such properties. Second, files with multi-dimensional attributes produce prohibitively expensive storage and indexing costs when using conventional file systems based on a directory-based hierarchy. Third, 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 sets of complex data-intensive applications. It is thus our belief that semantic-aware caching, which leverages metadata semantic correlation and combines pre-processing and pre-fetching based on range and top-k queries, will be sufficiently effective in reducing the working sets and increase cache hit rates.

At the same time, new I/O interfaces are of great necessity to allow users to flexibly locate target files in a large-scale storage system. Of particularly desirable interfaces are range query and top-k query, where the former identifies files whose attribute value is within a given range, while the latter locates *k* files whose attributes are closest to given values. For example, a user may wish to obtain the answers to `Range Query:` "*Which experiments did I run yesterday that took less than 30 minutes and generated files larger than 2.6GB?*" and `Top-K Query:` "*I can not accurately remember a previously created file but I know that its file size is around 300MB and it was last visited around Jan.1, 2008. Can the system show 10 files that are closest to this description?*".

In a small-scale storage system, conventional directory-tree based design and I/O interfaces may support these complex file queries through exhaustive or brute-force searches. However, in an Exabyte-scale storage system, complex queries need to be judiciously supported in a scalable way since exhaustive searches can result in prohibitively high overheads. Furthermore, the inherent performance bottleneck imposed by the tree structure in conventional file system design can become unacceptably severe in an Exabyte-scale system. While database techniques may arguably offer alternative solutions for complex queries, the nature of databases dictates that they manage an enormous amount of ordered tables and indexed lists for each single metadata update [13] ignoring access locality and skewed distribution of file metadata and resulting in unacceptable user performance. In addition, databases provide over-enforced consistency control and thus can cause unnecessary overhead.

This paper proposes a novel decentralized semantic-aware metadata organization scheme, called *SmartStore*, to efficiently manage file metadata. SmartStore organizes files into a semantic R-tree through semantic analysis on file metadata, which enables efficient complex queries and focuses on methods for metadata organization and query in a decentralized environment. Semantic R-tree is a multi-dimensional structure to organize file metadata and represent their multi-dimensional attributes, while supporting point query and complex queries that include range and top-k queries to facilitate the construction of semantic-aware caching.

Figure 1 shows an example that compares SmartStore against conventional file systems. The basic idea behind SmartStore is that files are grouped and stored according to their metadata semantics, instead of directory namespace. This 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 bottleneck and single-point failure since it renders the metadata organization fully decentralized whereby most operations, such as insertion/deletion and queries, can be executed within a given group.

Some related research has been conducted by other researchers. Data space [11] from the database research field is proposed as a data co-existence approach to organizing and operating collections of heterogeneous and partially unstructured data, by employing automatic methods to extract semantic relationships of data to obtain approximate query results. Database techniques generally cannot take full advantage of important characteristics of file systems, such as access locality and "hot spot" data, to enhance system performance. On the other hand, Spyglass [12] exploits the locality of file namespace and skewed distribution of metadata to map namespace hierarchy into a multi-dimensional K-D tree and uses multi-level versioning and partitioning to maintain consistency. It partitions the namespace hierarchy with increasing versioning overheads to support filename-based point query services. In contrast, SmartStore leverages semantics of multi-dimensional attributes, of which namespace is only a part, to produce distributed semantic R-tree adaptively based on metadata semantics and support complex queries with high reliability and fault tolerance.

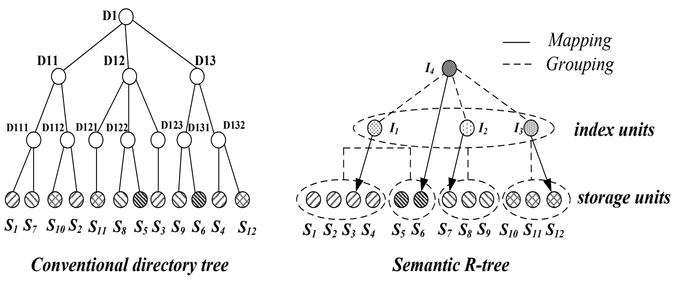

D1
D11
D12
D13
D111
D112
D121
D122
D123
D131
D132
S1 S7 S10 S2 S11 S8 S5 S3 S9 S6 S4 S12
Conventional directory tree
Mapping
Grouping
I4
I1
I2
I3
index units
storage units
S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12
Semantic R-tree

Figure 1. Comparisons with conventional file system.

This paper makes the following main contributions.

- **Decentralized metadata semantic-aware file system organization scheme**: Although partition may simplify the management over mass data [16, 17], intelligent and semantic-aware design can further optimize and enhance system performance by leveraging the inherent and abundant correlations among files. This paper presents a decentralized semantic-aware metadata organization scheme to support complex multi-query services and improve system performance by judiciously exploiting the metadata semantic information of files and effectively utilizing semantic analysis tools, i.e., Latent Semantic Indexing (LSI) [19]. 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 inefficiency. 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.
- **Practical implementation**: We have implemented the proposed SmartStore by aggregating correlated files into

several groups based on their metadata semantics, which can be iteratively aggregated into groups at higher levels until a single group is formed at the highest level, thus producing a semantic R-tree structure. We maintain the complete index information that contains multi-dimensional attributes of stored files in multiple decentralized storage servers through mapping all nodes in the semantic R-tree onto actual storage servers, thus generating a decentralized index structure. Index units, like folders in the conventional directory tree structure, that maintain the summary of multi-dimensional attribute information are now deployed among the storage units in a decentralized way to facilitate multi-query services. Based on experimental results, SmartStore prototype is more than one thousand times faster and 20 times smaller than current database methods with a very small false probability.

- **Multi-query services**: To the best of our knowledge, this is the first study to design and implement 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 the 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.

The rest of the paper is organized as follows. Section 2 describes the SmartStore system design. Section 3 presents details of design and implementation. Section 4 shows key issues for discussions. Section 5 presents the extensive experimental results. Section 6 presents the related work. Section 7 concludes our paper.

2. SmartStore System

2.1 Overview

The SmartStore system has three components: 1) Based on LSI tool, SmartStore semantically represents and groups metadata into storage and index units; 2) The units iteratively construct semantic R-tree structure that runs in a distributed environment; 3) In the semantic R-tree, SmartStore supports insertion, deletion and multi-query services.

Figure 2 shows a diagram of SmartStore that provides multi-query services for users and meanwhile organizes metadata to enhance system performance by using decentralized semantic R-tree structure. Section 3 describes the design details of each component and here we discuss the overall functions respectively from user and system views.

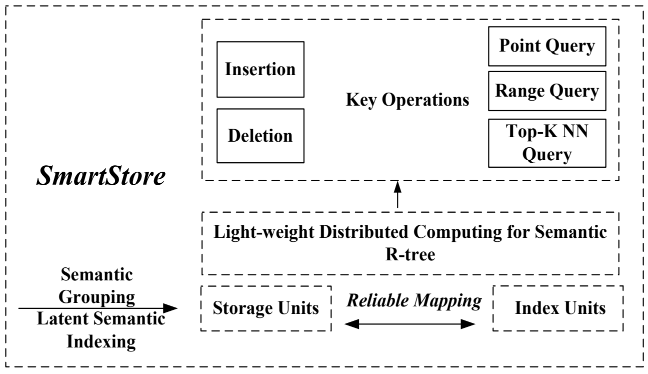


Figure 2. SmartStore system and components.

2.2 User View

SmartStore supports flexible multi-query services for users and these queries follow similar query path. In general, users initially send a query request to a randomly chosen server that is also represented as storage unit that is a leaf node of semantic R-tree. The chosen storage unit, also called *home* unit for the request, then retrieves semantic R-tree nodes by using an on-line multicast-based or off-line pre-computation approach to locating a query request to its correlated R-tree node. After obtaining query results, the home unit returns them to users. The key difference between point query and complex queries including range or top-k is that a point query checks Bloom filters [20] and a complex query checks minimum bounding rectangles (MBR) [7].

2.3 System View

Semantic grouping is one of the key components in the system design, which efficiently utilizes metadata semantic information, such as file physical and behavior attributes, to carry out the semantic-based metadata grouping. These multi-dimensional attributes exhibit different characteristics. For example, attributes such as access frequency, file size, amount of "read" and "write" operations are changed frequently, while some other attributes, such as filename and initial creation time, remains untouched most of time. SmartStore combines these attributes to conjecture their semantic correlations and then organizes correlated metadata into groups that are in turn structured in a semantic R-tree. These metadata groups are stored within a distributed environment with multiple metadata servers. By grouping correlated metadata, SmartStore exploits their affinity to boost the performance of both point query and complex queries.

Storage and index units as shown in Figure 3 constructing semantic R-tree come from semantic grouping by exploiting and leveraging metadata semantics. Each metadata server is a leaf node in our semantic R-tree and potentially holds multiple non-leaf 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 non-leaf nodes as *index units*.

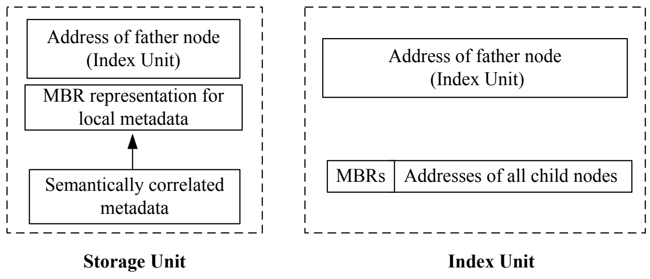

Address of father node (Index Unit)
MBR representation for local metadata
Semantically correlated metadata
Storage Unit
Address of father node (Index Unit)
MBRs
Addresses of all child nodes
Index Unit

Figure 3. Storage and index units.

3. Design and Implementation

In this section, we present SmartStore components including semantic grouping, system reconfigurations such as node insertion and deletion, and point and complex queries.

3.1 Semantic Grouping

3.1.1 Statement and Tool

We define semantic metadata grouping and then show our approaches to achieving this grouping.

Statement 1 (**Semantic Metadata Grouping**). *Given file metadata with D-dimensional attributes, find d-dimensional* $(1 \leq d \leq D)$ *attributes, representing special interests, to partition similar file metadata into multiple groups so that:*

- *A file in a group has higher correlations with the 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 SmartStore. In the first iteration of grouping, a predetermined constant value $\varepsilon_1 (0 \leq \varepsilon_1 \leq 1)$ is used as an admission threshold. 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 includes a `grouping predicate`, which serves as grouping criteria. The semantic grouping process can be recursively executed by aggregating groups in the $(i-1)$th-level into the ith-level nodes of the semantic R-tree with the association value $\varepsilon_i (0 \leq \varepsilon_i \leq 1), (1 \leq i \leq H)$, until reaching the root, where H is the depth of the constructed R-tree.

More than one predicate may be used to construct the 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 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.

An item a with D-dimensional attributes can be represented as a **semantic vector** $S_a = [S_1, S_2, \cdots, S_D]$. Similarly, a point query can also be abstracted as $S_a = [S_1, S_2, \cdots, S_d]$ $(1 \leq d \leq D)$. In the semantic R-tree, each node covers all metadata that can be accessed through its children nodes. Each node can also be represented by a geometric centroid of all metadata it covers. When attributes are represented as semantic vectors, they can be either physical ones, such as time of creation and file size, or behavioral ones, such as corresponding process ID and access pattern. Our previous work [9] shows that combining physical and behavioral attributes can improve the identification of file correlations, which help improve cache hit rate and prefetch accuracy.

We propose to use Latent Semantic Indexing (LSI) [19] as a tool to measure semantic similarity. LSI is a technique based on the Singular Value Decomposition (SVD) [18] to measure semantic similarity. SVD reduces a high-dimensional vector into a low-dimensional one by projecting the large vector into a semantic subspace. Specifically, SVD decomposes an attribute-file matrix A, whose rank is r, into the product of three matrices, i.e., $A = U \Sigma V^T$, where $U = (u_1, \ldots, u_r) \in R^{t \times r}$ and $V = (v_1, \ldots, v_r) \in R^{d \times r}$ are orthogonal, $\Sigma = diag(\sigma_1, \ldots, \sigma_r) \in R^{r \times r}$ is diagonal, and σ_i is the i-th singular value of A. V^T is the transpose of matrix V. LSI utilizes an approximate solution by representing A with a rank-k matrix to delete all but k largest singular values, i.e., $A_k = U_k \Sigma_k V_k^T$.

A metadata query for attribute i can also be represented as a semantic vector of size k, i.e., the i-th row of $U_k \in R^{t \times k}$. In this way, LSI projects a query vector $q \in R^{t \times 1}$ into the k-dimensional semantic space in the form of $\hat{q} = U_k^T q$ or $\hat{q} = \Sigma_k^{-1} U_k^T q$. The inverse of the singular values is used to scale the vector. The similarity between semantic vectors is measured as their inner product. Due to space limitation, this paper only gives a basic introduction to LSI and details can be found in [19].

3.1.2 Basic Grouping Procedures

We first use LSI to group metadata according to their semantic similarity. Next we present how to further formulate and organize the groups into a semantic R-tree.

First, we calculate the correlations among these servers, each of which is represented as a leaf node (i.e., storage unit). Given N nodes storing D-dimensional metadata, a semantic vector with d attributes $(1 \leq d \leq D)$ is constructed by using LSI to represent each of the N metadata nodes. Then using the semantic vectors of these N nodes as input of the LSI tool, we obtain the semantic correlation value between any two nodes, x and y, among these N nodes.

The next procedure is to build parent nodes, i.e., the first-level non-leaf node (index unit), in the semantic R-tree. Nodes x and y are aggregated into a new group if their correlation value is larger than a predefined admission threshold ε_1. When a node has correlation values larger than ε_1 with more than one another node, the one with the largest correlation value will be chosen. These groups are recursively aggregated until all of them form a single one that is the root of R-tree. In the semantic R-tree, each tree node uses Minimum Bounding Rectangles (MBR) to represent all metadata that can be accessed through its children nodes.

The above procedures aggregate all metadata into a semantic R-tree. For complex queries, the query traffic is very likely bounded within one or a small number of tree nodes due to metadata semantic correlations and similarities. If each tree node is stored on a single metadata server, such query traffic is then bounded within one or a small number of metadata servers. Therefore, the proposed SmartStore can effectively avoid or minimize brute-force searches that must be used in conventional directory-based file systems for point and complex queries.

3.2 System Reconfigurations

3.2.1 Insertion

When a storage unit is inserted into a semantic group of storage units, the semantic R-tree is adaptively adjusted to balance the workload among all storage units within this group. An insertion operation involves two steps: group location and threshold adjustment. Both steps only access a small fraction of semantic R-tree in order to avoid message flooding in the system.

When inserting a storage unit as a leaf node of the semantic R-tree, we need to first identify a group that is most closely related to this unit. Semantic correlation value between this new node and a randomly chosen group is computed by using the LSI analysis over their semantic vectors. If the value is larger than certain admission threshold, the group then accepts the storage unit as a new member. Otherwise, the new unit will be forwarded to adjacent groups for admission checking. After a storage unit is inserted into a group, the MBR will be updated to cover the new unit.

The admission threshold is one of the key design parameter to balance load among multiple storage units within a group. It directly determines the semantic correlation, membership, and size of a semantic group. The initialized value of this threshold is determined by the sampling analysis. After inserting a new storage unit into a semantic group, the threshold is dynamically adjusted to keep the semantic R-tree balanced.

3.2.2 Deletion

The deletion operation in the semantic R-tree is similar to a deletion in a conventional R-tree [7]. Deleting a given node entails adjusting the semantic association of that group, including the value of group vector and the multi-dimensional MBR of each group node. If a group contains too few storage units, the remaining units of this group are merged into its sibling group. When a group becomes a child node of its former grandparent in the semantic R-tree as a result of becoming the only child of its father due to group merging, its height adjustment is propagated upward if necessary.

3.3 On-line Queries

We present the intuitive on-line queries that locate a query request to correlated node by multicasting messages.

3.3.1 Range Query

A range query is to find files satisfying multi-dimensional range constraints. A range query can be easily supported in the semantic R-tree that contains an MBR on each tree node with a time complexity bounded by $O(\log N)$ for N storage units. A range query request can be initially sent to any storage unit that then multicasts query messages to its father and sibling nodes in semantic R-tree to identify correlated target nodes, which may contain results with high probability.

3.3.2 Top-K Query

A top-k query aims to identify K files with attribute values that are closest to the desired one, query point q. The main operations are similar to those of a range query. After a storage unit receives a query request, it first checks its father node, i.e., an index node, to identify a target node in the semantic R-tree that is most closely associated with the query point q. After checking the target node, we obtain a *MaxD* that is used to measure the maximum distance between the query point q and all currently obtained results, serving as a threshold for improving the queried results. The *MaxD* value can be updated when obtaining better results. By multicasting query messages, the sibling nodes of the target node are further checked to verify whether the current *MaxD* represents the smallest distance to the query point, i.e., determining whether there are still better results. Until the parent node of the target node cannot find files with smaller distance than *MaxD*, we can return the top-k query results to the request.

3.3.3 Point Query

Filename-based indexing is very popular in existing file systems and will likely remain popular in future file systems. A point query for filenames is to find some specific file, if it exists, among storage units. A simple but bandwidth-inefficient solution is to send the query request to a sequence of storage units to ascertain the existence and location of the queried file following the semantic R-tree directly. This method suffers from long delays and high bandwidth overheads.

In SmartStore, we deployed a different approach for point query. Specifically, Bloom filters [20], which are space-efficient data structures for membership queries, are embedded into storage and index units to support fast filename-based query services. A Bloom filter is built for each leaf node to represent the file names of all files whose metadata are stored locally. The Bloom filter of an index unit is obtained by the logical union operations of the Bloom filters of its child nodes, as shown in Figure 4. A filename-based query will be routed along the path on which the corresponding Bloom filters report positive hits. A false positive is solved by broadcasting to all leaf nodes.

The above multi-query operations have to suffer from heavy messages to locate most correlated nodes that contain

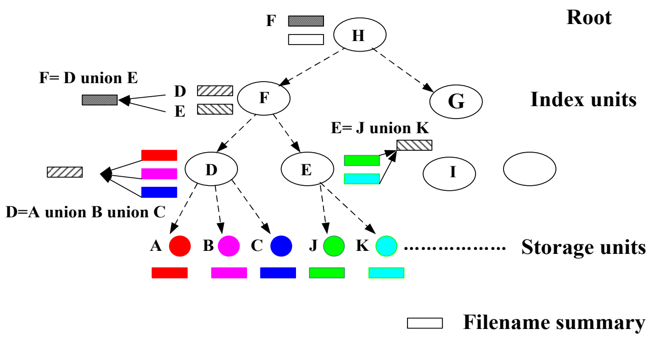


Figure 4. Bloom filters used for filename-based query.

queried files with high probability, since a query request is randomly allocated to a storage unit that is possibly not correlated with the request. The query performance can be further improved by off-line pre-processing.

3.4 Queries Accelerated by Off-line Pre-processing

To further accelerate queries, we utilize duplicate-based approach to performing off-line pre-processing. Specifically, each storage unit locally maintains a copy of the semantic vectors of all index units to speed up the queries. The basic idea of the off-line pre-processing is that each storage unit maintains semantic information of a limited number of index units to strike a tradeoff between accuracy and maintenance costs as shown in Figure 5. We deploy the replicas of first-level index units, e.g., D, E, I, in storage units to obtain a good tradeoff, which is further verified in Section 5.4. Any arriving request thus can first be formulated into a request vector based on its multi-dimensional attributes. We then use the LSI tool over the request vector and semantic vectors of existing index units to check which index unit is the most semantically associated with the request, i.e., discovering the target index unit that has the largest probability of successfully serving the request. The request can be further forwarded directly to the target index unit, in which a local search is performed. In this way, we can efficiently reduce the communication costs for positioning target units using off-line pre-processing, for point, range and top-k queries.

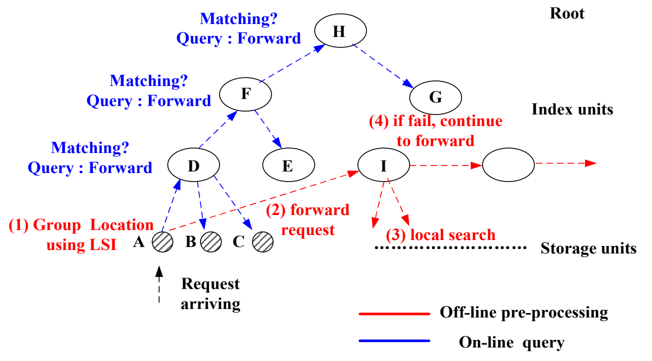


Figure 5. On-line and off-line queries. Off-line pre-processing can quickly locate target index unit that has high probability to satisfying requests.

4. Key Issues

We discuss key issues in SmartStore in terms of node split/merge, units mapping and attribute updating.

4.1 Node Split and Merge

The operations of splitting and merging nodes in semantic R-tree follow the standard algorithms in R-tree [7], in which a node will be split when the number of its child nodes is larger than a predetermined threshold M and a node is merged with its adjacent neighbor when the number of its child nodes is smaller than another predetermined threshold m. In our design, the parameter m and M can be defined as $m \leq \frac{M}{2}$. The parameter m can be varied in real-world applications by experimental tuning to obtain load balance.

4.2 Multi-Mapping for Index Units

Since index units are stored in storage units, it is necessary and important to map the former into the latter in a way that balances the load among storage units while enhances system reliability. Our mapping is based on a simple bottom-up approach that iteratively applies the random selection and labeling operations, as shown in Figure 6 by an example. The example shows the process that maps index units to storage units. An index unit in the first level can be first randomly mapped to one of its child nodes in the R-tree (i.e., a storage unit from the covered semantic group). Each storage unit that has been mapped by an index node is labeled to avoid being mapped by another index node. After all the first-level index units have been mapped into storage units, the same mapping process is applied to the second-level index units that are mapped to the remaining storage units. This mapping process repeats iteratively until the root node of the semantic R-tree is mapped. In practice, also verified by experiments in Section 5.4, the number of storage units is generally much larger than that of index units and thus each index unit can be mapped to a different storage unit.

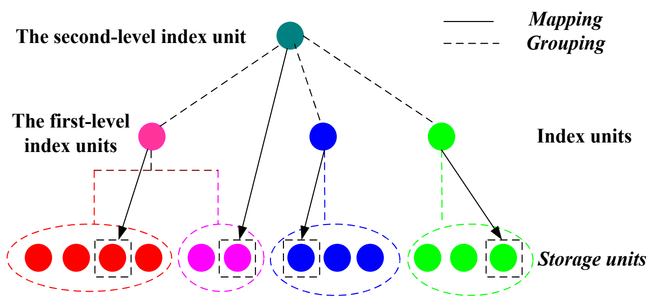


Figure 6. Mapping operations for index units.

Our semantic grouping scheme aggregates correlated metadata into semantic-aware groups that can satisfy query requests with high probability and the experimental results in Section 5 further show that most of requests can obtain query results by visiting one or a very small number of groups. The root node hence will not likely become a performance bottleneck.

However, the single point of failure of the root node may result in a serious threat to system reliability. Thus, we utilize a multi-mapping approach to enhancing system reliability through fault tolerance, by allowing the root node to be mapped to multiple storage units. In this multi-mapping of the root node, the root is mapped to a storage unit in each group of the storage units that cover a different subtree of the semantic R-tree, so that the root can be found within all different subtrees. Since each father node in the semantic R-tree maintains the MBR-based multi-dimensional attribute ranges of its child nodes while the root keeps the attribute bounds of files of the entire system (or application environment), a change on a file or metadata will not necessarily lead to an update on the root node representation, unless it results in a new attribute value that falls outside of any attribute bound maintained by the root. Thus, most changes to metadata in a storage unit will not likely lead to an update on the root node, which significantly reduces the cost of maintaining consistency among the multiple copies of the root node that must multicast any change to one copy of the root to all others.

This method of mapping the root node to all semantic "groups" at a certain level of the semantic R-tree also facilitates fast query services and improves system reliability. It can help speedup the query services by quickly eliminating an unsatisfiable query request through checking the root to determine if the query range falls outside of the root range. If a root in a group fails, the group can easily obtain a copy of the root index information from one of the adjacent groups to create a new mapping within the failing group and then multicast the newly mapped root to other groups.

4.3 Attribute Monotone for Updating

SmartStore can support updating function by analyzing the monotone of multi-dimensional attributes to further insert, delete or migrate changed files, obtaining system load balance and improving query accuracy.

SmartStore considers an attribute to be of monotone-sensitivity if its value always increases or decreases over time. One example is the time of creation of a file since the time is increased monotonically. The monotone-sensitivity attributes potentially leads to inefficiency of semantic R-tree representation since when adding a new file, its created time must often be outside of the existing represented MBR ranges, i.e., an outlier, thus resulting in frequent updates to maintain information consistency and query accuracy. In contrast, non-monotone-sensitivity attributes, e.g., file size, needs not to be always increased or decreased, hence leading to infrequent updating operations. The difference between monotone-sensitivity and monotone-insensitivity attributes is that the former may become an outlier of the root MBR and thus requires updating all nodes from the current node to the root, whereas the latter may still be covered by the current node's MBR and thus require no or limited updates.

Our scheme considers an aggregate-based approach for updating. After changing the value of a monotone-sensitivity attribute, SmartStore needs to re-computes the correlation of semantic vectors among changed files and all index units of the first-level node. If the file still maintains the most correlation with the current group, meaning that no migration is required, SmartStore sends the update message to the index unit of the group to extend its MBR, while the updated attribute information will be kept in the index unit as an aggregate. When the number of updated messages in the aggregate is larger than some threshold, we then update the MBR of all nodes from the current index unit to the root. The changed file might also be migrated into the group that is most correlated and the new group will carry out the similar operations to update higher-level nodes in an aggregate way. On the other hand, if the value of a monotone-insensitivity attribute is changed, we only need to update the related nodes that consider the changed value an outlier.

The feasibility of the aggregate-based update design for monotone-sensitivity attributes is that the root is infrequently visited and some update latency will not introduce much query errors, while our first-level index units that are frequently visited keep the up-to-date attribute information to guarantee query accuracy.

5. Performance Evaluation

We have implemented a prototype of SmartStore. This section evaluates SmartStore through the prototype by using representative large I/O traces, including *HP* [21], *MSN* [14], *EECS* [15]. We compare SmartStore against the database approach and the evaluation metrics considered are query accuracy, query latency and communication overhead.

5.1 System Implementation

The SmartStore prototype is implemented in the Linux environment and our experiments are conducted on a cluster of 60 storage units. Each storage unit has Intel Core 2 Duo CPU, 2GB memory, and high-speed networks. We carry out the experiments for 30 runs each to validate the results based on the evaluation guidelines of file and storage systems [1].

In order to emulate the I/O behaviors of the next-generation storage systems, we scaled up the I/O traces both spatially and temporally. Specifically, a trace is decomposed into subtraces. We add the unique subtrace ID to all files to intentionally increase the working set. The start time of all subtraces is set to zero so that they are replayed concurrently. The chronological order among all requests within a subtrace 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 subtraces replayed concurrently is denoted as the *Trace Intensifying Factor* (TIF) as shown in Table 1, 2 and 3. Similar workload scale-up approaches have also been used in other studies [5, 6].

Table 1. Scaled-up HP trace.

	Original	**TIF=80**
request (million)	94.7	7576
active users	32	2560
user accounts	207	16560
active files (million)	0.969	77.52
total files (million)	4	320

Table 2. Scaled-up MSN trace.

	Original	**TIF=100**
# of files (million)	1.25	125
total READ (million)	3.30	330
total WRITE (million)	1.17	117
duration (hours)	6	600
total I/O (million)	4.47	447

Table 3. Scaled-up EECS trace.

	Original	**TIF=150**
total READ (million)	0.46	69
READ size (GB)	5.1	765
total WRITE (million)	0.667	100.05
WRITE size (GB)	9.1	1365
total operations (million)	4.44	666

We compare SmartStore with two baseline systems. The first one is a popular database approach that uses a B^+ tree [8] to index each metadata attribute. The second one is a simple, non-semantic R-tree-based database approach that organizes each file based on its multidimensional attributes without leveraging metadata semantics.

While filename-based point query is very popular in most file systems 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 reasonable complex queries within the multi-dimensional attribute space. The file static attributes and behavioral attributes are derived from the available I/O traces. We randomly generate range values used for range query, and desired values used for top-k query according to three different statistic distributions, including Uniform, Gauss, and Zipf. Due to space limitation, we mainly present the results of Zipf distribution.

5.2 Grouping Efficiency

The grouping efficiency determines how effectively SmartStore can bound a query within a small set of semantic groups to improve the overall system scalability. Figure 7 shows that most operations, from 87.3% to 90.6%, can be served by one group, i.e., 0-hop routing distance. This confirms the effectiveness of our semantic grouping. In addition, since the semantic vector of one group, i.e., the first-level index unit in the semantic R-tree, can accurately present the aggregated metadata, these vectors are replicated to other storage units in order to perform fast and accurate queries locally as mentioned in Section 3.4. The observed results further prove the feasibility of the off-line pre-processing scheme, which can quickly direct a query request to the most correlated index units.

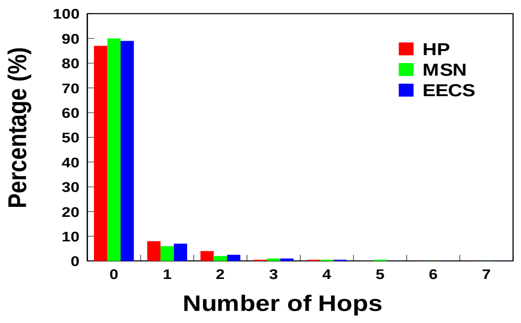


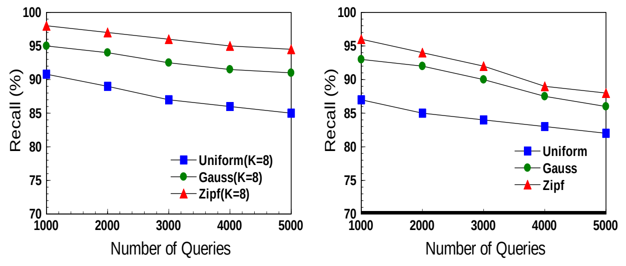
Figure 7. The number of hops of routing distance.

5.3 Query Accuracy

We evaluate the performance of complex queries by using "Recall" metric and point query by checking Bloom filters.

5.3.1 Complex Queries

We use "Recall" as a metric often used to measure search quality in the field of information retrieval, to evaluate the query accuracy. 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.

Figure 8. Recall of complex queries using *HP* trace.

Figure 8 shows the recall values of complex queries, including range and top-k (k=8) nearest neighbor (NN) 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 uniform distribution. This is because under a Zipf or Gauss distribution, files are mutually associated with a higher degree than under uniform distribution. Table 4 and 5 show the recalls of range and top-k queries, as a function of the query numbers, for *MSN* and *EECS* traces. Experimental results further confirm that SmartStore can achieve a high accuracy for complex queries.

5.3.2 Point Query

SmartStore can support point query, i.e., filename-based indexing, through checking multiple Bloom filters stored in index units as described in Section 3.3.3. Although Bloom

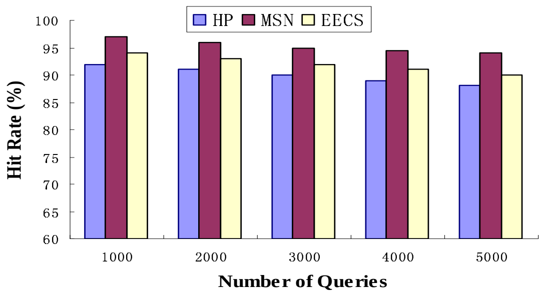


Figure 9. Average hit rate for point query.

Table 4. Recall of range and top-k queries using *MSN*.

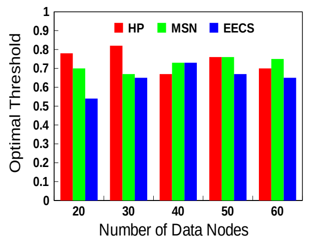
		1000	2000	3000	4000	5000
Uniform	Range Query	86.2	85.7	84.5	83.2	82.8
	K=8	90.5	89.7	87.4	86.2	85.8
Gauss	Range Query	90.5	89.3	88.6	87.7	86.4
	K=8	95.8	94.2	93.5	92.4	91.6
Zipf	Range Query	91.2	90.5	89.3	88.7	87.3
	K=8	96.5	95.1	94.3	93.6	92.6

Table 5. Recall of range and top-k queries using *EECS*.

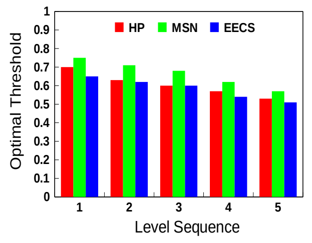
		1000	2000	3000	4000	5000
Uniform	Range Query	87.3	86.5	84.6	83.2	81.5
	K=8	91.5	90.2	89.8	87.4	85.6
Gauss	Range Query	89.7	88.2	87.5	85.5	83.1
	K=8	96.7	95.1	94.2	92.3	91.1
Zipf	Range Query	90.2	89.6	87.5	86.7	84.8
	K=8	97.3	96.2	94.8	93.5	92.7

filter-based searching may lead to false positives and false negatives due to hash collisions and information staleness, the false probability is very small. In addition, these false positive and false negative are identified when the target metadata is accessed. Figure 9 shows the hit rate for point query. It is observed that over 88.2% query requests can be served accurately by Bloom filters.

5.4 System Scalability



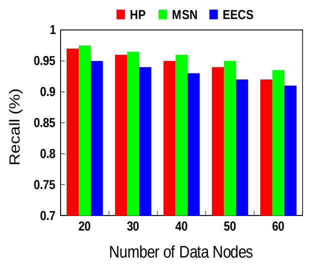
(a) Increasing system scale.



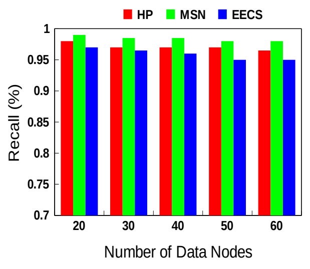
(b) Checking semantic R-tree levels for a total of 60 data nodes.

Figure 10. Optimal thresholds in system scale and semantic R-tree levels.

We study the impact of system size on the optimal threshold, as shown in Figure 10. Figure 10(a) shows the optimal thresholds when the total number of storage units increases from 20 to 60. Figure 10(b) shows the optimal thresholds at different levels of the semantic R-tree. We examine the query accuracy by measuring the recall when executing 2000 requests composed of 1000 range and 1000 top-k queries, as show in Figures 11. These the requests are respectively generated from Gauss and Zipf distributions. Experimental results show that SmartStore can consistently achieve a high query accuracy when the number of storage units increases, demonstrating the scalability of SmartStore.



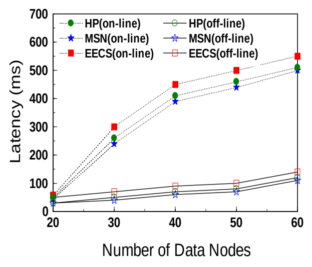
(a) Gauss distribution.



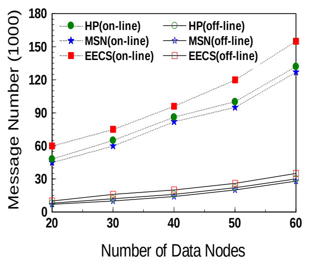
(b) Zipf distribution.

Figure 11. Recall with the increments of system scale within the requests following Gauss and Zipf distributions.

We compare on-line and off-line query performance in terms of query latency and message numbers with the system scale increments as shown in Figure 12. Figures 12(a) compares the query latency between two methods, as described in Section 3.4, under a Zipf distribution. The on-line method identifies the most correlated storage unit for the query request by multicasting messages; whereas, the off-line method stores semantic vectors of the first-level index units in advance to execute off-line LSI-based pre-processing to quickly locate the most correlated index unit. In addition, Figures 12(b) compares the number of internal network messages of the on-line and off-line approaches when performing complex queries. We observe that the off-line approach can significantly reduce the total number of network messages. This is mainly due to the fact that it uses LSI-based pre-processing to determine correlated storage unit for a query request.



(a) Average query latency.



(b) Message numbers.

Figure 12. Performance comparisons using on-line and off-line approaches.

5.5 Performance Comparisons between SmartStore and Baseline Systems

We compare the query latency between SmartStore and the two baseline systems described earlier in Section 5.1, labeled DBMS and R-tree respectively. Figure 13 and 14 show the latency comparisons of point, range and top-k queries using the *MSN* and *EECS* traces. It is observed that SmartStore incurs much lower latency. SmartStore effectively aggregates correlated file metadata together and significantly limits search scope. In contrast, DBMS has to check each B^+-tree index for each attribute, resulting in linear brute-force search costs. Although the non-semantic R-tree approach is able to improve query performance, to some extent, by using a multi-dimensional structure and thus allowing the parallel indexing on all attributes, its query latency is still much larger than SmartStore due to no considerations on semantic correlations.

We also examine the space overhead of SmartStore, R-tree and DBMS, as shown in Figure 15. SmartStore consumes much smaller space than the R-tree and DBMS approaches, due to its decentralized scheme and multi-dimensional representation. SmartStore stores the index structure, i.e., semantic R-tree, across multiple nodes, while R-tree is a centralized structure. Additionally, SmartStore utilizes the multi-dimensional attribute structure, i.e., semantic R-tree, while DBMS builds a B^+-tree for each attribute. As a result, DBMS has a large storage overhead. Since SmartStore has a small space overhead and can be stored in memory on most servers, it allows the query to be served at the speed of memory or network.

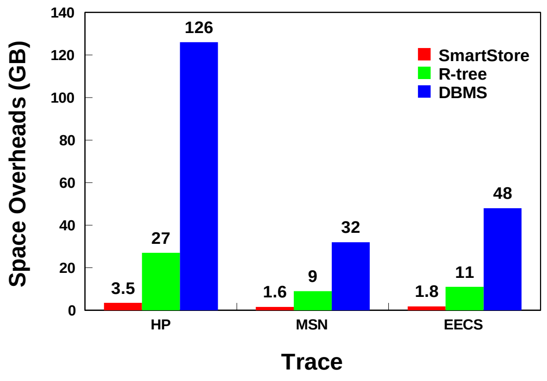


Figure 15. Comparisons of space overheads of SmartStore, R-tree and DBMS.

6. Related Work

Researchers in the database field aim to bring database capacity to Petabyte scales with billions of records. For example, the design based on the notion of data space [11] is proposed as a data co-existence approach to serving the collections of heterogeneous and partially unstructured data. However, the data space approach is potentially inefficient for mass data in file systems because it lacks the optimization for system workloads by exploiting access patterns and locality of reference and skewed distribution of metadata, and requires a prohibitively large and complex table lists for maintaining relationships of all participants.

Distributed directory service for Farsite [2] utilizes tree-structured file identifiers to support dynamically partitioning metadata at arbitrary granularities. Ceph [3] maximizes the separation between data and metadata management by using a pseudo-random data distribution function to support a scalable and decentralized placement of replicated data. G-HBA [6] supports a scalable and adaptive decentralized metadata lookup scheme that logically organizes metadata servers into a query hierarchy and exploits grouped Bloom filters to efficiently route metadata requests to desired servers through the hierarchy. Our SmartStore can efficiently support complex queries, which are rarely investigated in existing file systems. SmartStore clusters semantically associated metadata servers into groups and schedules a metadata query request to the group that can successfully serve this request with a very high probability.

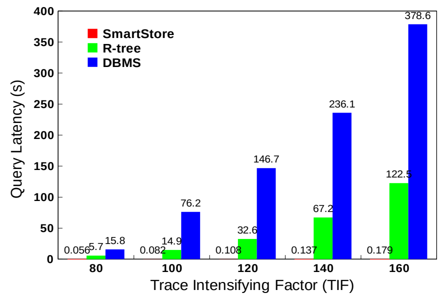
Our proposed architecture mainly exploits the advantages of R-tree for multi-dimensional range and top-k queries. An R-tree [7] is a tree-based data structure that, similar to B-tree [8], is often used to represent and index spatial multi-dimensional data by minimum bounding rectangles (MBR). An R-tree can split the data space into hierarchically nested bounding boxes, which can contain several data entities within certain ranges. In our design, we exploit its special capability of supporting range and top-k nearest neighbor (NN) queries by modifying its structure appropriately to serve our special purpose of semantic grouping. Although successful for databases, R-tree-based research has not been directly conducted in large-scale distributed file systems, especially for supporting complex queries. To the best of our knowledge, SmartStore is the first study that attempts to exploit the R-tree structure for semantic grouping of file metadata to significantly reduce the costs of key operations in complex queries.

7. Conclusion

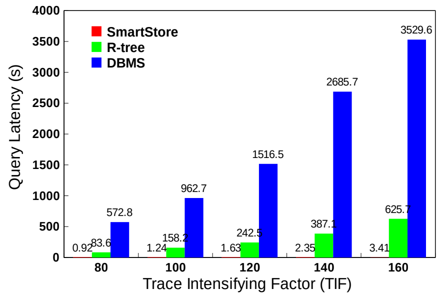
The paper presents a new paradigm for organizing file metadata for next-generation file systems, called SmartStore, by exploiting file semantic information to provide efficient and scalable complex queries while enhancing system scalability and functionality. The novelty of SmartStore lies in that it matches actual data distribution and physical layout with their logical semantic association so that a complex query can be successfully served within one or a small number of storage units. Specifically, this paper has made three main contributions. (1) A semantic grouping algorithm is proposed to effectively identify files that are correlated in their physical attributes or behavioral attributes. (2) SmartStore can very efficiently support complex queries, such as range and top-k, which will likely become increasingly important in the next-generation file systems. (3) Our prototype implementation proves that SmartStore is highly scalable, and can be deployed in a large-scale storage system with many storage units.

References

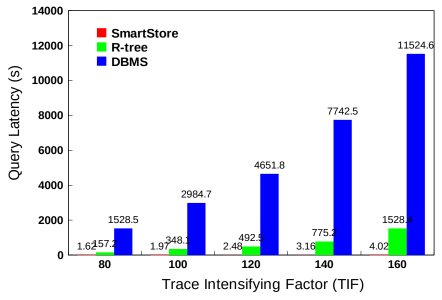
[1] Traeger A., Zadok E., Joukov N., Wright C.P. A nine year study of file system and storage benchmarking. *ACM Transactions on Storage*, 4(2): 1-56, 2008.



(a) Point query.

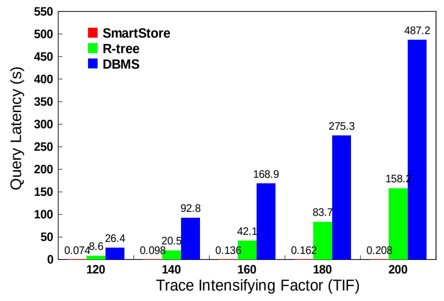


(b) Range query.

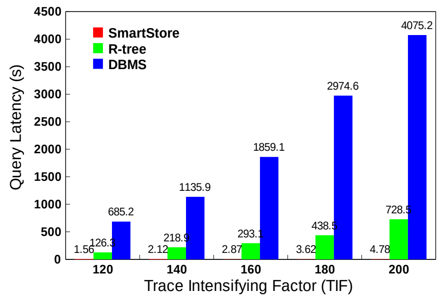


(c) Top-k query.

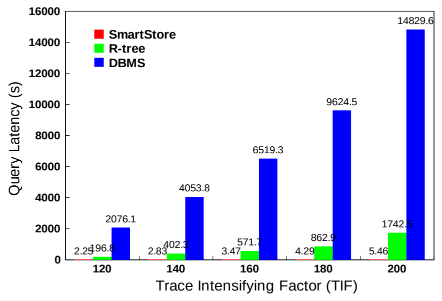
Figure 13. Query latency comparisons of SmartStore, R-tree and DBMS using *MSN* trace.



(a) Point query.



(b) Range query.



(c) Top-k query.

Figure 14. Query latency comparisons of SmartStore, R-tree and DBMS using *EECS* trace.

[2] John R. Douceur, Jon Howell. Distributed Directory Service in the Farsite File System. In *Proc. OSDI*, pages 321–334, 2006.

[3] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long and Carlos Maltzahn. Ceph: A Scalable, High-Performance Distributed File System. In *Proc. OSDI*, 2006.

[4] Floyd RA, Ellis CS. Directory reference patterns in hierarchical file systems. *IEEE Transactions on Knowledge and Data Engineering*, 1(2):238–247,1989.

[5] Yifeng Zhu, Hong Jiang, Jun Wang, Feng Xian. HBA: Distributed Metadata Management for Large Cluster-based Storage Systems. *IEEE Transactions on Parallel and Distributed Systems*, 19(4): 1-14, 2008.

[6] Yu Hua, Yifeng Zhu, Hong Jiang, Dan Feng, Lei Tian. Scalable and Adaptive Metadata Management in Ultra Large-scale File Systems. In *Proc. ICDCS*, 2008.

[7] Antonin Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proc. ACM SIGMOD*, pages 47-57, 1984.

[8] D Comer. The Ubiquitous B-tree. *Computing Survey*, 11(2): 121-138, 1979.

[9] Peng Xia, Dan Feng, Hong Jiang, Lei Tian, Fang Wang. FARMER: A Novel Approach to File Access coRrelation Mining and Evaluation Reference model for Optimizing Peta-Scale File Systems Performance. In *Proc. HPDC*, 2008.

[10] Roselli D, Lorch J.R., Anderson T.E. A comparison of file system workloads. In *Proc. USENIX Annual Technical Conference*, pages 41–54, 2000.

[11] Franklin M., Halevy A., Maier D. From databases to dataspaces: a new abstraction for information management. *ACM SIGMOD Record*, 34(4):27–33, 2005.

[12] Andrew W. Leung, Minglong Shao, Tim Bisson, Shankar Pasupathy, Ethan L. Miller. Spyglass: Fast, Scalable Metadata Search for Large-Scale Storage Systems. *Technical Report UCSC-SSRC-08-01*, 2008.

[13] Margo Seltzer. Beyond Relational Database. *Communications of the ACM*, 51(7): 52–58, 2008.

[14] Swaroop Kavalanekar, Bruce Worthington, Qi Zhang, Vishal Sharda. Characterization of storage workload traces from production Windows servers. In *Proc. IEEE International Symposium on Workload Characterization (IISWC)*, 2008.

[15] Daniel Ellard, Jonathan Ledlie, Pia Malkani, Margo Seltzer. Passive NFS Tracing of Email and Research Workloads. In *Proc. FAST*, pages 203–216, 2003.

[16] Alexander Szalay. New Challenges in Petascale Scientific Databases. In *Keynote Talk in Scientific and Statistical Database Management Conference (SSDBM)*, 2008.

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

[18] Golub G.H., Van Loan C.F. Matrix Computations. *Johns Hopkins University Press*, 1996.

[19] Papadimitriou C.H., Raghavan P., Tamaki H., Vempala S. Latent Semantic Indexing: A Probabilistic Analysis. *Journal of Computer and System Sciences*, 61(2):217–235, 2000.

[20] Burton Bloom. Space/time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 13(7): 422–426, 1970.

[21] Erik Riedel, Mahesh Kallahalla, Ram Swaminathan. A framework for evaluating storage system security. In *Proc. FAST*, 2002.