Smart- store metadata file systems by using semantic r-tree

Miss. A. NIROSHA., (M.E),

Department of Computer Science and Engineering

Dhanalakshmi Srinivasan Engineering College, Perambalur.

niroanbu@gmail.com

Mr. R.GOPI., M.Tech., Assistant Professor,

Department of Computer Science and Engineering

Dhanalakshmi Srinivasan Engineering College, Perambalur.

gopircse@gmail.com

ABSTRACT - Existing search mechanisms of DHT-based P2P systems allow every individual keyword to be mapped to a set of documents/nodes across the network that contains the keyword. Lookup time and Data traffic is increased. In this design proposes a decentralized semantic-aware metadata organization, called Smart Store, which exploits semantics of files metadata to judiciously aggregate correlated files into semantic-aware groups by using information retrieval tools. The key idea of Smart Store 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 Smart Store can improve system scalability and reduce query latency for Complex queries such as range and top-k queries. Bloom Filter to decrease space overhead and provide fast identification of stale versions by using Hashing queried items. A query in Smart-Store 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. Specifically, for a point query, the home unit servers. After obtaining query results, the home unit returns them to the user. To construct a semantic R-tree by leveraging three attributes, i.e., file size, creation time, and last modification time.

Keywords: Smart-Store, Semantic R-Tree, Metadata Management Server, Bloom Filter.

1. INTRODUCTION:

FAST and flexible metadata retrieving is critical requirements bin the next-generation data storage systems serving High-end computing [3]. As the storage capacity is approaching Exa-bytes and the number of files stored is reaching billions, Directory-tree based metadata management widely deployed in conventional file systems 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 in efficient in very large-scale file systems due to four main reasons. [1] 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 exa-byte-data-volume (EB). Unfortunately, existing distributed databases fail to achieve efficient management of peta-

bytes of data and thousands of concurrent requests. Second, for heterogeneous execution environments, devices of file systems are heterogeneous, such as supercomputers, clusters of PCs via Ethernet, Infinite Band and Fibers, and cloud storage via Internet.

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. This issue has also been observed by file system researchers. Third, for heterogeneous data types, their metadata in file systems are also heterogeneous. The metadata may be structured, semi-structured, 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, [10] 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.

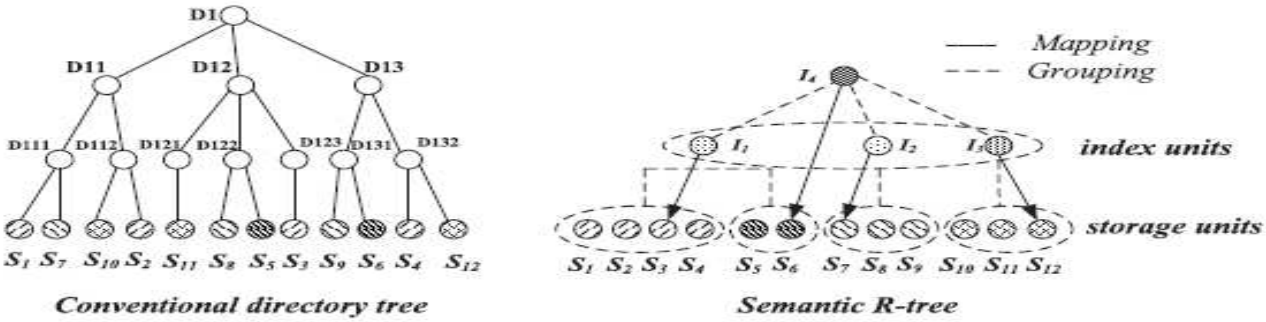

Fig 1: Comparison with Conventional File System.

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 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 Smart-Store, to effectively exploit semantic correlation to enable efficient complex queries for users and to improve system performance in real-world applications.

Additionally and importantly, Smart-Store 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 Smart-Store 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 Multi-query 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 Smart- Store can support three query interfaces for point, range, and top-k queries.

2. SMARTSTORE SYSTEMS:

The basic idea behind Smart-Store is that files are grouped and stored according to their metadata semantics. 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.

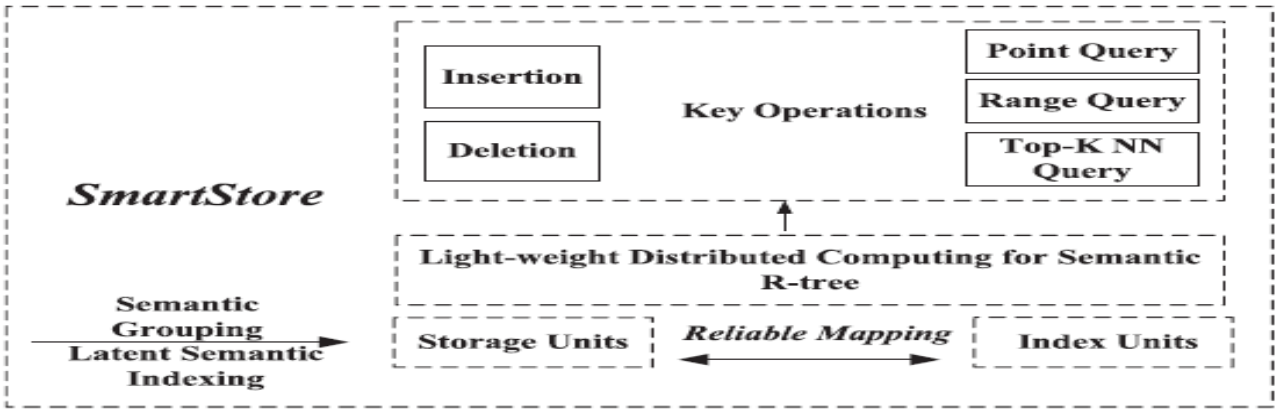

Fig 2: Smart-Stores

2.1 OVERVIEW:

A semantic R-tree is evolved from classical R-tree and consists of index units (i.e., Non-leaf 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. Smart-Store supports complex queries, including range and top-k queries, in addition to simple point query. Smart-Store that provides multi-query services for users while organizes metadata to Smart-Store 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 multi-query services.

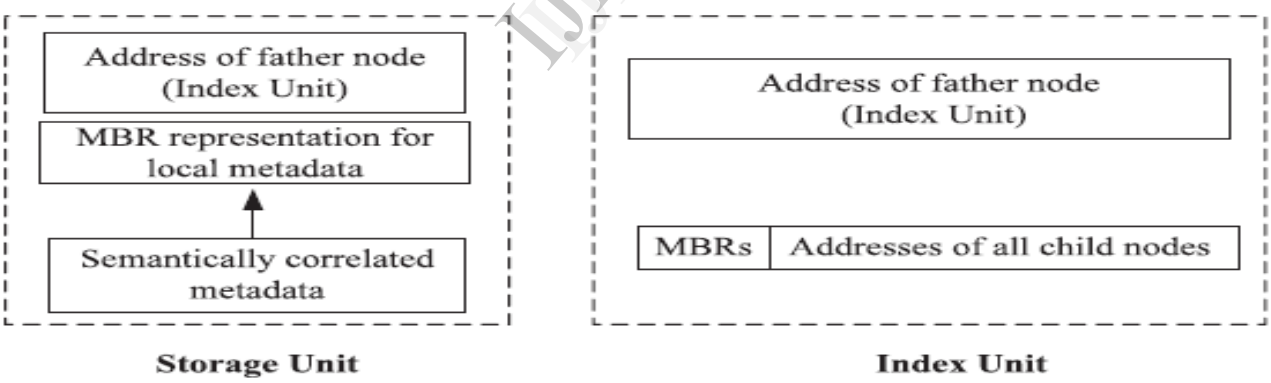

Fig 2.1: Storage Unit & Index Unit

2.2 USER VIEW:

A query in Smart-Store 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 stored locally in a way similar to the group-based hierarchical Bloom-filter array approach and, for a complex query, the home unit checks the Minimum Bounding Rectangles (MBR) 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. 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. Smart-Store 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. 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. In Section 3 & 6 specifies the User Process?

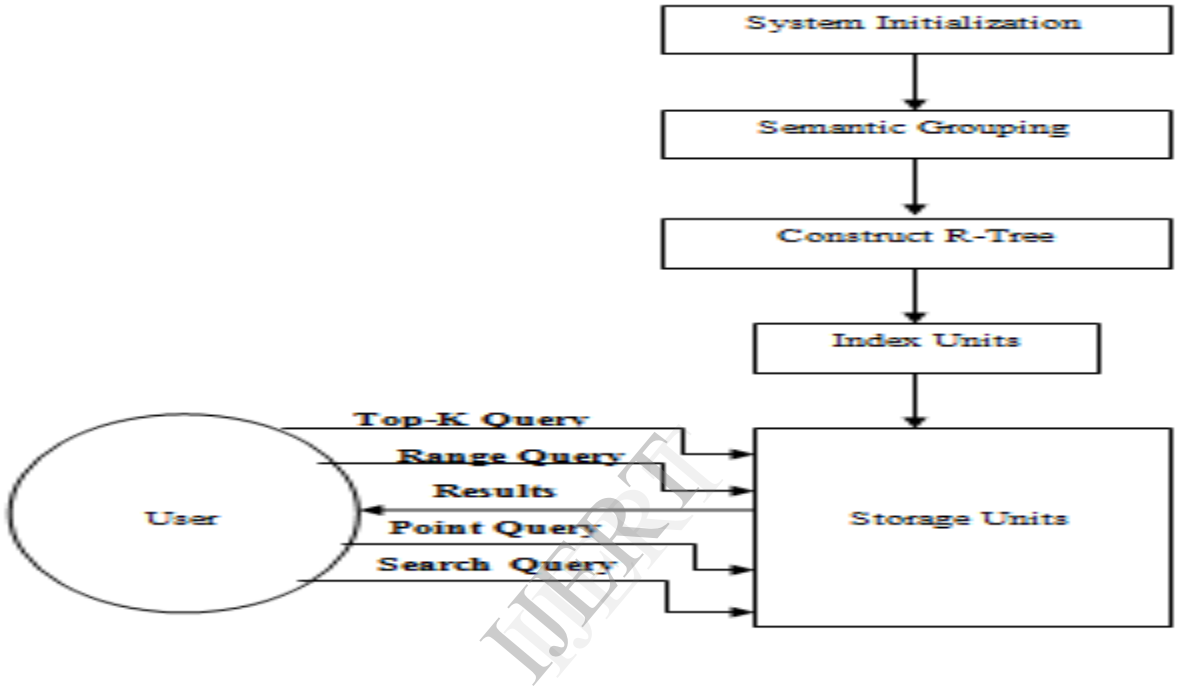

Fig 2.2: System Architecture

3. RELATED WORKS:

A query in Smart-Store 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. Specifically, for a point query, the home unit checks Bloom filters stored locally in a way similar to the group-based hierarchical Bloom-filter array approach and, for a complex query, the home unit checks the Minimum Bounding Rectangles (MBR) to determine the membership of queried file within checked servers.

After obtaining query results, the home unit returns them to the user. To 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 type and creation time, or other combinations of these three attributes.

3.1 HASH BASED ALGORITHM:-

3.1.1 BLOOM FILTER:-

Bloom filter are compact data structures for probabilistic representation of a set in order to support membership queries (i.e. queries that ask: "Is element X in set Y?"). This compact representation is the payoff for allowing a small rate of false positives in membership queries; that is, queries might incorrectly recognize an element as member of the set.

The base data structure of a Bloom filter is a **Bit Vector**. Each empty cell in that table represents a bit and the number below it its index. To add an

element to the Bloom filter, we simply hash it a few times and set the bits in the bit vector at the index of those hashes to 1.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Fig 3.1.1: Bloom Filter

3.1.2 DESCRIPTION OF ALGORITHM:-

Bloom filter, representing the set {x, y, z}. The colored arrows show the positions in the bit array that each set element is mapped to. The element w is not in the set {x, y, z}, because it hashes to one bit-array position containing 0. For this figure, m=18 and k=3. S is a set of n elements.

It's easier to see that that means than explain it, so enter some strings and see how the bit vector changes:

Enter a string: johnsmith:
Doe: You're set: 1

When you add a string, you can see that the bits at the index given by the hashes are set to 1. I've used the color green to show the newly added ones, but any colored cell is simply a 1.

Set of k hash functions with range {1 : : :m} (or {0 : : :m – 1}).

M-long array of bits initialized to 0. To insert and query on a Bloom filter of size

m = 10 and number of hash functions k = 3.

Let H(x) denote the result of the three hash functions which we will write as a set of three values {h1(x); h2(x); h3(x)g}.Start with an empty 10-bit long array:

Insert x1: H(x1) = {4,5,8}

0	1	0	0	1	1	0	0	1	0
0	1	2	3	4	5	6	7	8	9

Delete x1: H(x1) = {4,5,8}

0	1	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9

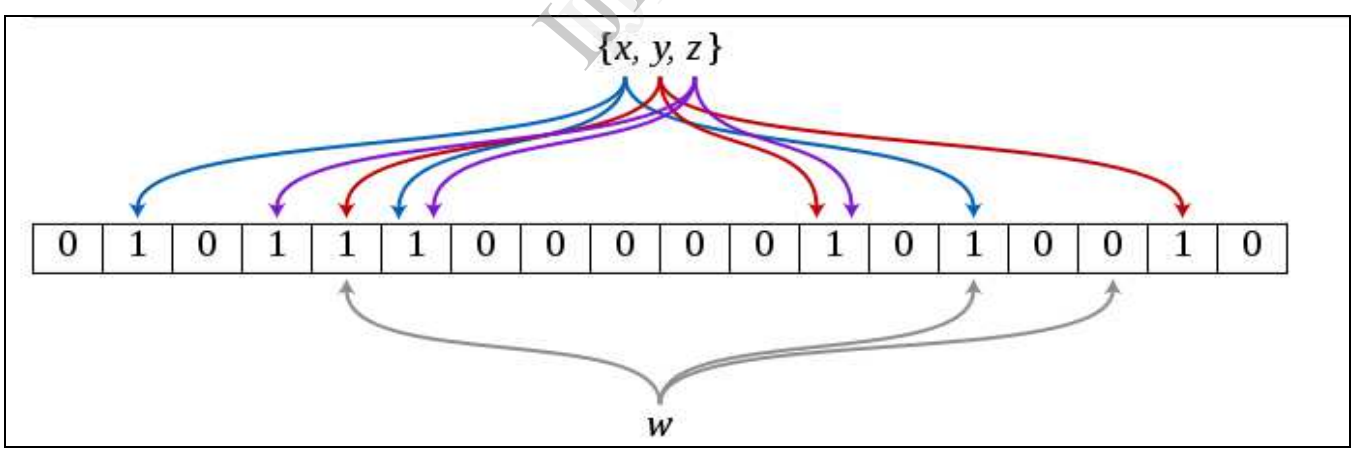

Fig 3.1.2: Bloom Filter Indexing Table

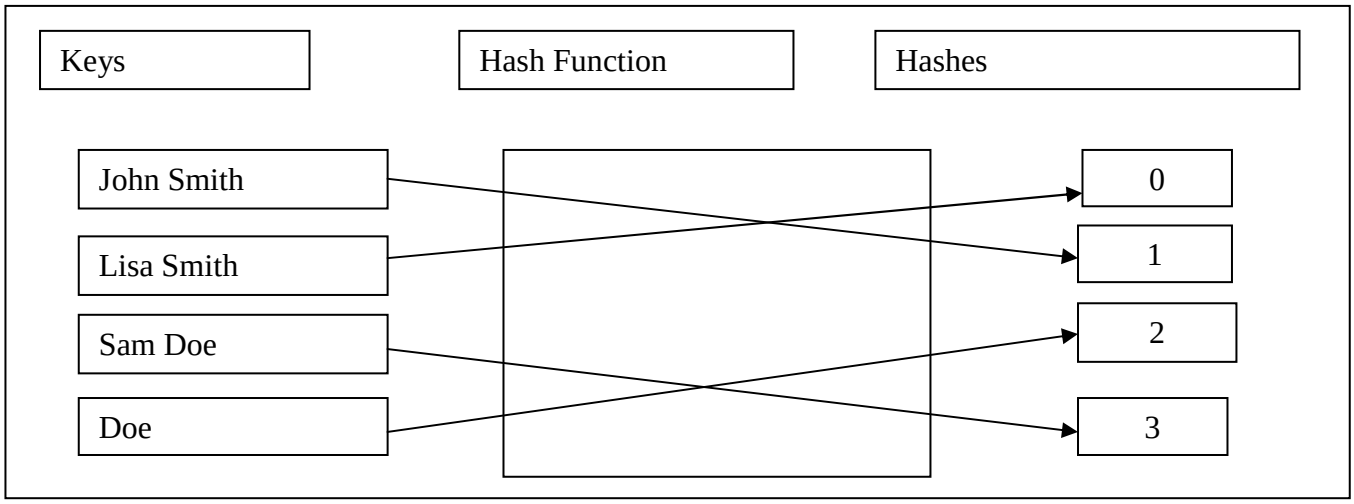

Fig 3.1.2.1: Hash Function

4. CREATION OF GROUP:

The system contains the collection of two different Units like Storage Unit, Indexing Unit. In this module first create the Groups by giving the detail information of group such as Group Name, Group IP Address and Group Port Number. Each Group having different number of MDSs. A group split operation is then triggered to divide this group into two approximately equal-sized groups, A and B. The split operation will be performed under two conditions: 1) Each group must still maintain a global mirror image of the file system, and 2) Workload must be balanced within each group.

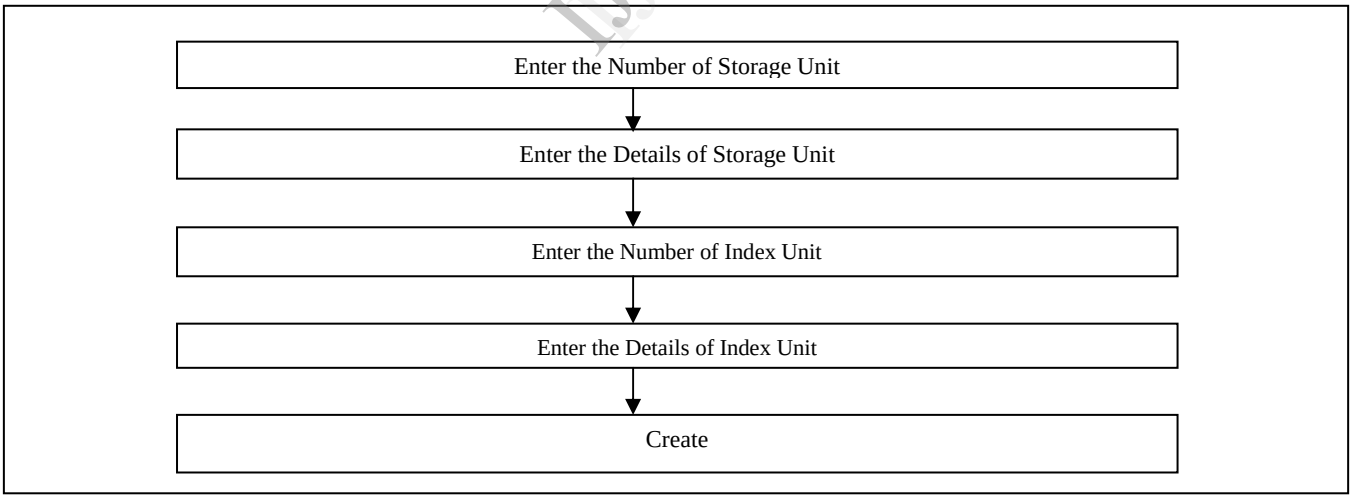

Fig 4: Creation of Group

5. CONSTRUCTION OF R-TREE:

A semantic R-Tree consists of index units (i.e., non-leaf 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.

Smart Store supports complex queries, including range and top-k queries, in addition to simple point query. Smart Store that provides multi-query services for users while organizes metadata to enhance system performance by using decentralized semantic R-tree structures.

Each metadata server is a leaf node in our semantic R-tree and can also potentially hold multiple non-leaf nodes of the R-tree. We refer to the semantic R-tree leaf nodes as storage units and the non-leaf nodes as index units.

Smart- Store can support three query interfaces for point, range, and top-k queries.

Smart-Store 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;
3) The Service Component that supports insertion, deletion in R-trees, and multi query services.

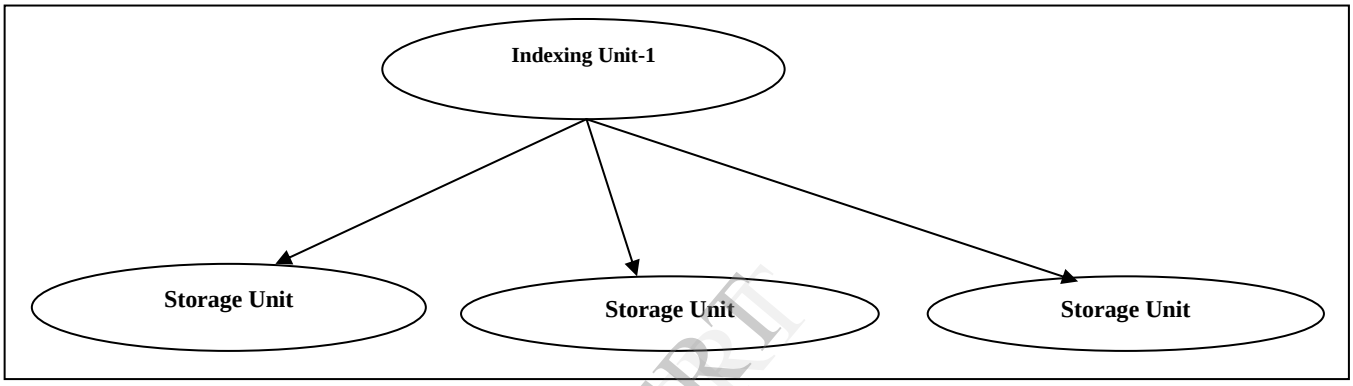

Fig 5: Construction of R-Tree

6. USER QUERY PROCESSING:

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.

To 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 type and creation time, or other combinations of these three attributes.

Smart Store is able to provide the existing services of conventional file systems while supporting new complex query services with high reliability and scalability. Bloom Filter to decrease space overhead and provide fast identification of stale versions by using Hashing queried items.

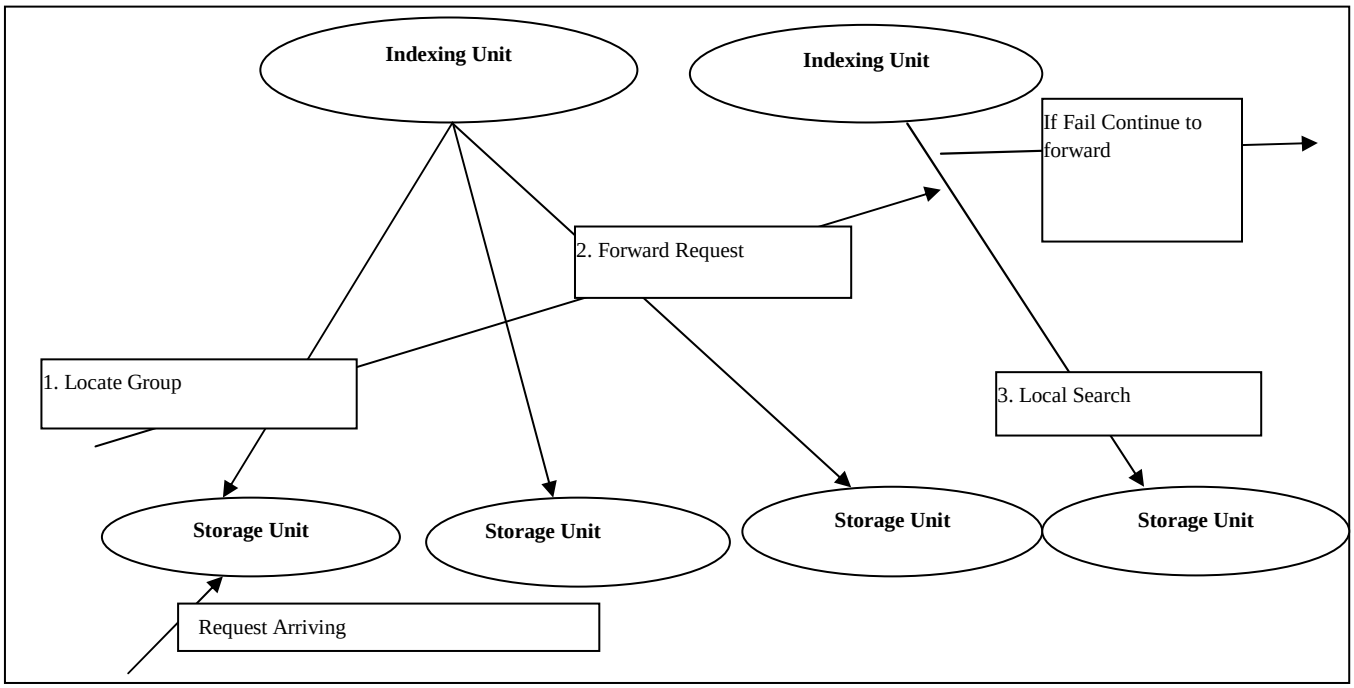

Fig 6: User Query Processing

7. MULTI QUERY SERVICE:

Smart Store 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.

Smart Store removes attached versions when reconfiguring index units. The frequency of reconfiguration depends on the user requirements and environment constraints. Removing versions entails two operations. 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.

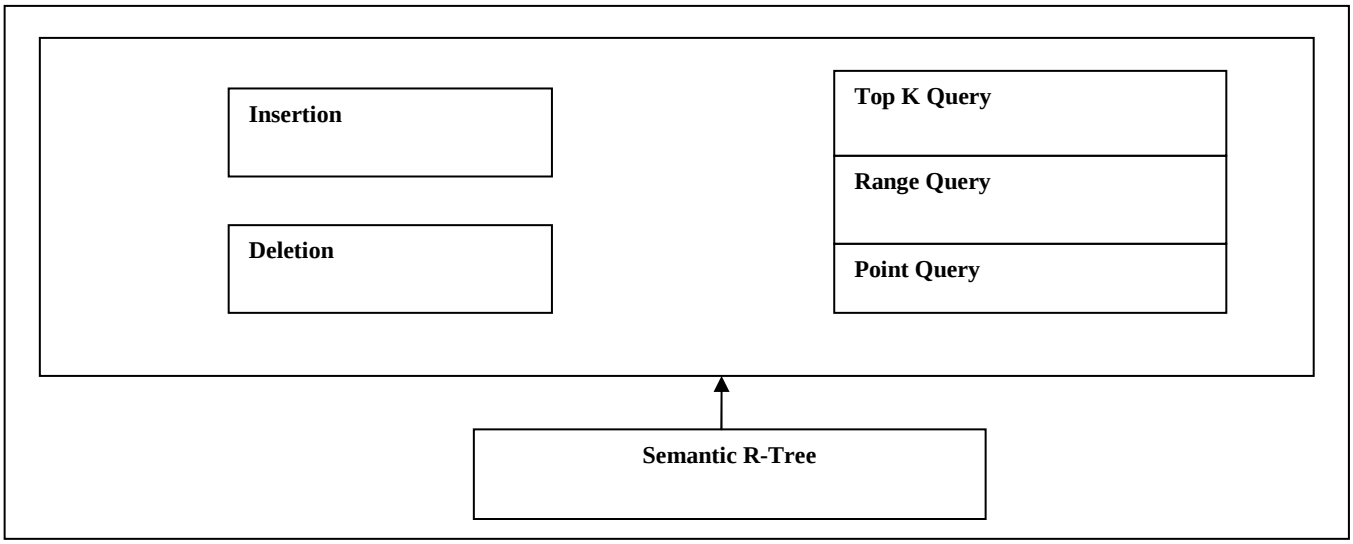

Fig 7: Multi Query Processing

8. MDSs OPERATION:

To create the Group of MDSs by giving the detailed information of MDSs such as Name of the MDSs, MDSs ID and MDSs Port Number. The maximum number of MDSs allowed in one group. The operations of adding and deleting MDSs are associated with group-based re-configuration (i.e., Group Splitting and Merging.).

To utilize an array of Bloom filters on each MDS to support distributed metadata lookup among multiple MDSs. An MDS, where a file's metadata reside, is called the home MDS of this file. Each metadata server constructs a Bloom filter to represent all files whose metadata are stored locally, and then replicates this filter to all other MDSs.

A metadata request from a client can randomly choose an MDS to perform membership query against its Bloom filter array that includes replicas of the Bloom filters of the other servers. The Bloom filter array returns a hit when exactly one filter gives a positive response. A miss takes place when zero hit or multiple hits are found in the array.

9. LRU-BF ARRAY:-

The query starts at the LRU BF array (L1), which aims to accurately capture the temporal access locality in metadata traffic streams. Each MDS maintains an LRU list that includes the most recently visited files whose metadata are maintained locally on that MDS. To use of the LRU BF to represent all the files cached in this LRU list. The LRU BF is then globally replicated to all MDSs of the entire system. The query starts at the LRU BF array (L1), which aims to accurately capture the temporal access locality in metadata traffic streams. Each MDS maintains an LRU list that includes the most recently visited files whose metadata are maintained locally on that MDS. We further make use of the LRU BF to represent all the files cached in this LRU list. The LRU BF is then globally replicated to all MDSs of the entire system. As a replacement occurs in the LRU list on an MDS, corresponding insertion and deletion operations are then performed by this MDS to update its LRU BF. The LRU BF is then replicated to all the other MDSs when the amount Segment BF If the query cannot be successfully served at L1, the query is then performed at L2.

10. SEGMENT-BF ARRAY

If the query cannot be successfully served at L1, the query is then performed at L2. L2 is a segment bloom filter array .It contains the replica copy of the other groups. Each MDS in the group contain n-m/m replicas. Each group maintain n-m replicas. The Segment BF array (L2) stored on an MDS i include only i BF replicas, with each replica representing all files whose metadata are stored on that corresponding MDS. Each MDS only maintains a subset of all replicas available in the systems. A lookup failure at L2 will lead to a query multicast among all MDSs within the current group (L3).

11. GLOBAL MULTICAST QUERY

At the last level of the query process, i.e., L4, each MDS directly performs a lookup by searching its local BF and disk drives. If the local BF responds negatively, the requested metadata are not stored locally on that MDS. since the local BF has no false negatives . However, if the local BF responds positively, a disk access is then required to verify the existence of the requested metadata, since the local BF can potentially generate false positives.

12. CONCLUSIONS

This paper presents a new paradigm for organizing file metadata for next-generation file systems, called Smart-Store, by exploiting file semantic information to provide efficient and scalable complex queries while enhancing system scalability and functionality. The novelty of Smart-Store 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. Smart-Store 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

Smart-Store is highly scalable, and can be deployed in a large-scale distributed storage system with a large number of storage units.

13. FUTURE ENHANCEMENT

To increase the hash function used in the Bloom Filter for reducing the false negatives. For any private data present in the data collections, we apply the effective security settings. To presents a new paradigm for organizing file metadata for next-generation file systems, called Smart-Store, by exploiting file semantic information to provide efficient and scalable complex queries while enhancing system scalability and functionality. Smart-Store 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.

REFERENCES

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

[2] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," Internet Math., vol. 1, no. 4, pp. 484-509, 2005.

[3] 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.

[4] T. Gross and V. Cate, "Combining the Concepts of Compression and Caching for a Two-Level Filesystem," ACM SIGARCH Computer Architecture News, vol. 19, no. 2, pp. 200-211, 1991.

[5] 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.

[6] E.L. Miller and R.H. Katz, "Rama: An Easy-to-Use, High- Performance Parallel File System," Parallel Computing, vol. 23, pp. 419-446, 1997.

[7] J.H. Morris, M. Satyanarayanan, M.H. Conner, J.H. Howard, D.S. Rosenthal, and F.D. Smith, "Andrew: A Distributed Personal Computing Environment," Comm. ACM, vol. 29, no. 3, pp. 184- 201, 1986.

[8] M.N. Nelson, B.B. Welch, and J.K. Ousterhout, "Caching in the Sprite Network File System," ACM Trans. Computer Systems, vol. 6, no. 1, pp. 134-154, 1988.

[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] H.T. Shen, Y.F. Shu, and B. Yu, "Efficient Semantic-Based Content Search in P2P Network," IEEE Trans. Knowledge and Data Eng., vol. 16, no. 7, pp. 813-826, July 2004.