IJCSEC- International Journal of Computer Science and Engineering Communications.
Vol.3, Issue 2, 2015, Page.663-670, ISSN: 2347–8586

SEMANTIC BASED DATA STORAGE WITH NEXT GENERATION CATEGORIZER

Ragavan.A
Lecturer/IT Department, S.K.P Engineering College,
Thiruvannamalai,Tamil Nadu, India.
athiragav@gmail.com
Vijayakumar.K
P.hD scholar Computer Science and Engineering
SKP Engineering College, Tiruvannamalai, Tamil Nadu, India
vijayakumarktvm@gmail.com
Kumaresan.A
Phd scholar, Computer Science and Engineering, SKP Engineering College
,Thiruvannamalai, Tamil Nadu, India.

Abstract-The namespace management is based on hierarchical directory trees. This tree-based namespace scheme is prone to severe performance bottlenecks and often fails to provide real time response to complex data lookups. This paper proposes a semantic-aware namespace scheme, called sane, which provides dynamic and adaptive namespace management for ultra-large storage systems with billions of files. Associative access on the files is provided by an initial extension to existing tree structured file system protocols, and by the use of these protocols that are designed specifically for content based file system access. Access on the file details such as versions or any other concepts were interpreted as queries applied on our container engine, and thus provides flexible associative access to files. Indexing of key properties of file system objects and indexing/ caching on the file system is one of the fantastic features of our system. The automatic indexing of files and grouped based on relativity is called "semantic" because user programmable nature of the system uses information about the semantics of updated file system objects to extract the properties for indexing. The semantic correlations and file groups identified in sane can also be used to facilitate file perfecting and data de-duplication, among other system-level optimizations.

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

I INTRODUCTION

The main aim of this project is storing the data in a custom repository in the file system. The data will be compressed and it will be stored in the container and retrieving the data faster the Indexing of data inside the container. Restring data access on the container and providing security and access rights on the documents in the container and data stored in a grouped manner based on the keywords and metadata information about the document. The scope of this project is to indexing the data in the container and Version management of the data in the container. Easy access of data in container

SYNOPSIS

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 Exabyte 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. 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. 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 but also result in billions of pieces of metadata in directories. 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. 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 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. Here, we propose a novel decentralized semantic ware 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.

II **RELATED WORK**

A. Namespace Construction

SANE offers a scalable way to construct semantic-aware per-file namespaces for the file system. SANE can provide the constructed namespaces of files in any storage area, such as portions of the main memory, and/or portions of the secondary storage of SSD or HDD. Here, take the main memory for an example, SANE is initialized by first carry-ing out LSH-based hash computation to cluster into groups files. These files are semantically correlated with the files already in the main memory of the file system, in which SANE is installed. SANE then organizes these groups into an R-tree structure. A nearest-neighbor query over the R-tree can identify the member files of the namespace of an individual file. This process repeats when a new file is accessed and loaded into the main memory. When the main memory is full, the namespace of a file will be replaced and flushed to the secondary memory based on a proper replacement policy, say, LRU. In other words, the main memory stores the per-file namespaces of the "hot" or "popular" files at any given time while "cold" files' name-spaces are stored in the secondary memory.We use the LSH-based R-trees described above to build the semantic aware per-file namespace. Specifically, for each file, its namespace is derived from the results of a top-t query that identifies the t nearest neighbors in the attribute space.

B. Data Storage Structure

LSH works well in identifying correlated data but suffers from the space-overhead problem as the amount of data increases. Since LSH requires many hash tables to main-tain correlated data, the space overhead becomes a poten-tial performance bottleneck. When dealing with massive amounts of data, data structure can easily overflow the main memory, leading to slow hard disk accesses and severe performance degradations. Furthermore, while the form of hash tables works for point-based queries (e.g., point and top-k queries), it may not efficiently support range queries that must obtain queried results within given intervals when the hash table fails to maintain the interval information.We make use of the R-tree [31] structure to replace the original hash tables, store the correlated data, and represent their multi-dimensional attributes in the R-tree nodes. The root node (e.g., R1) represents domain ranges of all possible attributes. Let N be the maximum number of children of a node. Each internal node can contain r ($\frac{N}{2}$ _ r _ N) child nodes. We set a lower bound on r to prevent tree degenera-tion and to ensure an efficient storage utilization. Whenever the number of children drops below r, the node will be deleted and its children will be re-distributed among sibling nodes. The upper bound N can guarantee that each tree node in fact can be stored exactly on one disk page. Each internal node contains entries in the form of ðI; PointerÞ where I ¼ ðI$_0$; I$_1$; . . . ; I$_{p_1}$Þ is a p-dimensional bounding box, representing a minimumbounding rectangle (MBR). I$_i$ is a bounded interval, which can cover items in the ith dimensional space. Pointer is the address of a child node.

III SMARTSTORE SYSTEM WITH THE BASIC CRUD OPTIONS

The relationship among the files often evolves; Rapport can fast identify their changes to update the namespace by exploiting the particular file semantic of provenance. The basic objective of the system is to create a Smart Store system with the basic CRUD options. In turn, the basic query operation of the users were tracked, top N Queries and Range of the top queries were retrieved. Semantic grouping of file data with a proper indexing is the core concept and the files were stored securely with a clear segregation of Storage Units and Index Units. Systems Petabytes or even Exabyte's. Our scheme logically creates metadata servers (MDS) into a multi-layered query hierarchy and exploits grouped filters Bloom to efficiently route metadata requests to desired MDSs

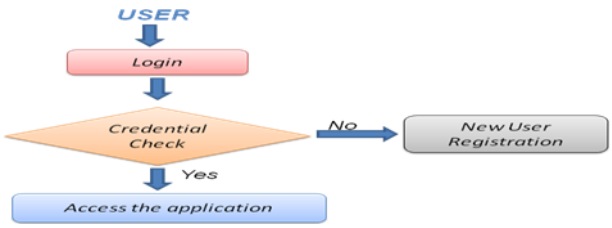
existing solutions that build a separate meta-data database outside of the file system face consistency and management challenges at large-scales. To address these is-sues, we developed Magellan, a new large-scale file system metadata architecture that enables the file system's metadata to be efficiently and directly searched.

IV SEMANTIC NAMESPACE IN LARGE SCALE FILE SYSTEM

The scale of these systems gives rise to many problems: they will be developed and used by many stakeholders across multiple organizations, often with conflicting purposes and needs; they will be constructed from heterogeneous parts with complex dependencies and emergent properties .Creating a customized container indicates the data will be compressed and it will be stored in the container. To enhance the process of retrieving the data faster the Indexing of data inside the container is achieved.Basic CRUD operations on the container cannot be logged in the existing system. This makes the admin to keep a track on the documents. Restring data access on the container and providing security and access rights on the documents in the container is not available in the existing system.the first attempt at providing semantic namespace in large-scale file systems, to the best of our knowledge. Different from existing namespace schemes. Indexing the container didn't specify the type of indexing which indicates it's a proposed one and not implemented. We are taking it up in our new system. However, providing effective search and indexing at the scale of billions of files is not a simple task. Current solutions rely on general-purpose index designs, such as relational databases, to provide search.

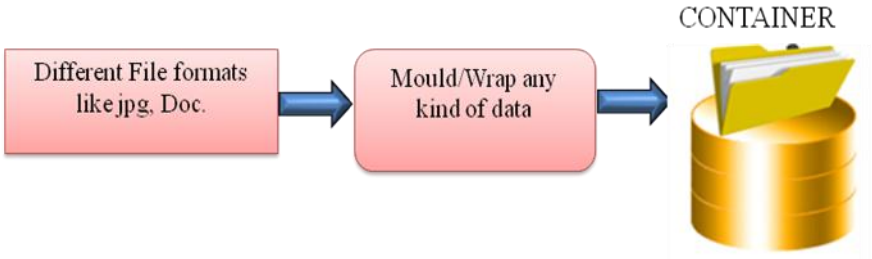
A. Authentication Module

Authentication Module describes the Co-Ordination between the users and the system admin. The user is allowed to create his credentials to login into the system. An admin needs to approve the users created and after the login approval the users will be allowed to access the application. During new user creation, the options of Security questions and answers were added in the system. This module provides an interface to the permission system inside the Application.



B. Container Creator Module

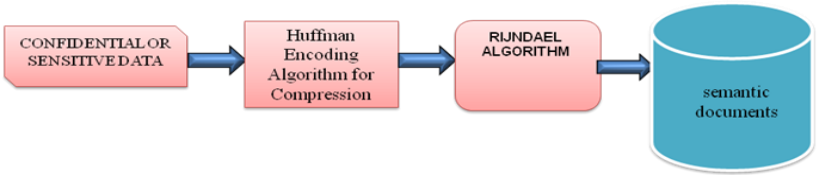
This module enables the user to create a container in the system. A container format could mould/wrap any kind of data. Though there are some examples of such file formats like jpg, Doc. Most container formats are specialized for specific data requirements. A popular family of containers is found for use with different multimedia file formats. Since video and audio streams can be coded and decoded with our specified algorithms, a container format may be used to provide a single file format to the user.



C. Data Uploader Module

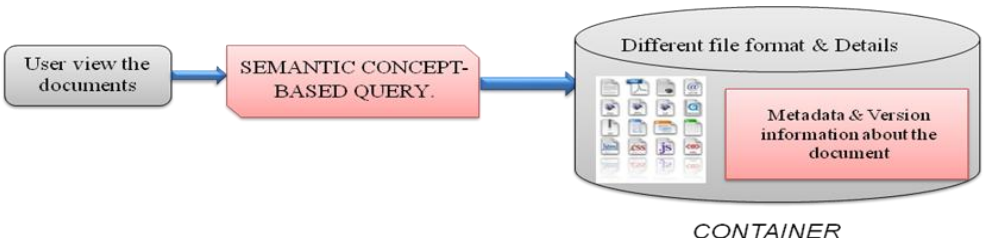
Users will have documents that contain confidential or sensitive data such that, the option of encrypt the data is one of an important factor in the field of network security/Data security.To avoid max storage space, our container provides an additional option of compressing the data.Semantic Doc user interfaces with a set of tools

that enable users to deal with semantic documents. In other words it provides the access to share their documents and to form a social container around shared documents.



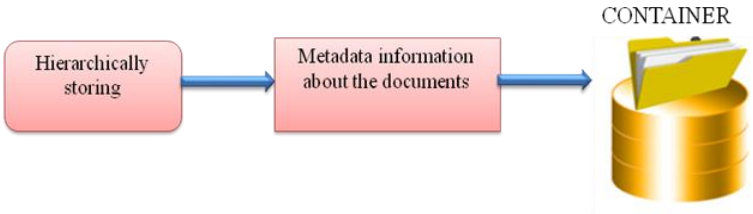
D. Container Visualize Module

This module provides an option for the user to view the documents stored in the semantic way. In turn, the documents will be stored in a grouped manner based on the keywords and metadata information about the document.The search starts by the user specifying keyword query which is then transformed into the corresponding semantic concept-based query.



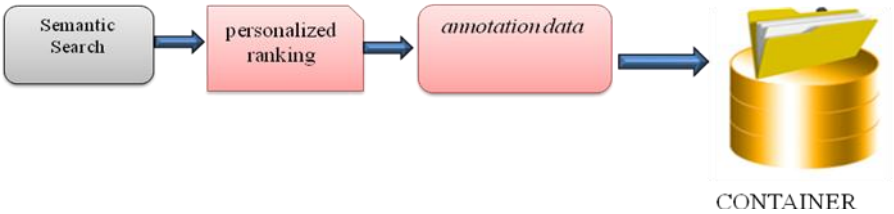
E. Index Tune Module

To enhance the performance of the document storage, a clear index were placed on hold on the documents.The documents will be hierarchically stored and indexed based on the logically metadata information about the documents.



F. Operations Details view

This module provides a detailed view about the operations happening on the system.The terms offered by the autocompletion are concept labels from domain ontologies which had been used for the semantic annotation of the semantic documents from the repository.After the semantic search and the personalized ranking, the recommender shows the list of found document units to the user. For each of the retrieved document units the user can also see additional information that come from its annotation data (e.g., the list of annotation concepts, the number of reuses, the number and list of users, the list of documents in which it appears and the number of versions)



V PERFORMANCE EVALUATION

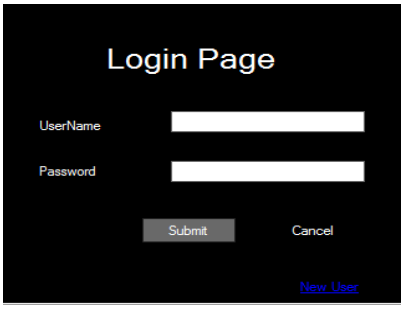
Platform. We have implemented a prototype of SANE in Linux kernel 2.6.28 and performed experiments on a cluster of 80 server nodes, each with Intel Core 2 Duo CPU and 2 GB memory. An RPC-based interface to WAFL (Write Anywhere File Layout) [32], [33] gathers the dynamic changes of file attributes, driven by our snapshot-based traces in the performance evaluation. SANE use three met-rics, namely, cost-effectiveness, searchability and scalability. All experiments have taken into account the dynamic evolution of file systems, such as file creations and deletions. In addi-tion, the user interfaces of SANE for namespace representa-tion, renaming and query service are also implemented in our prototype. We design and implement most modules at the user space level so that our prototype can run on many existing systems. Intensified Traces. We use four representative traces, of which three collected from the industry and one from the academia. These traces include HP file system trace [25], MSN trace [26], EECS NFS server (EECS) trace at Harvard [27] and Google clusters trace [28], which drive the SANE performance evaluation. Moreover, for the sizes of these traces, Google cluster contains 75 five-minute reporting intervals. There are a total of 3,535,029 observations, 9,218 unique jobs and 176,580 unique tasks. HP contains 94.7 million requests for a total of 4 million files from 32 users. MSN has 1.25 million files and records 4.47 million opera-tions, in which there are 3.3 million read and 1.17 million write operations. EECS contains 4.44 million operations. The number and size of read operations are respectively 0.46 mil-lion and 5.1 GB. Those of write operations are respectively 0.667 million and 9.1 GB. Due to their relatively small sizes, we use a scaled-up method to intensify them. In order to emulate the I/O behaviours of large-scale file systems for which no realistic traces are publicly available, we scaled up the existing I/O traces of current storage sys-tems both spatially and temporally. This method has been successfully used in Glance [12] and Smart Store [15]. Specifically, a trace is first decomposed into sub-traces. We then add a unique sub-trace ID to all files to intentionally increase the working set. The start times of all sub-traces are set to zero so that they are replayed concurrently. The chronological order among all requests within a sub-trace is faithfully preserved. In our performance evaluation, we intentionally choose specific comparison schemes in order to make the compari-son relevant and fair. We compare SANE with the target schemes only in the relevant aspects such as namespace construction, query performance for users, and file prefetch-ing and data reduplication for systems.

In the experiments, for a given file f, SANE first chooses its nearest neighbor file (t ¼ 1) as the member of its name-space. We then obtain the value of MSDðfÞ. If the value of MSDðfÞ increases after adding its next most closely corre-lated file to the namespace, the file is considered a member of file f's namespace and t is increased by 1, while MSDðfÞ is updated accordingly. Otherwise, the namespace construc-tion finishes. The bounds (minimum and maximum) and average values of t in four traces are respectively.In general, filename-based point query is very popular in most file system workloads. There are no file system I/O traces for both point and complex queries (range and top-k) requests. In order to address this issue, we leverage a syn-thetic approach to generating not only point query, but also complex queries within the multi-dimensional attribute space.

VI RESULTS

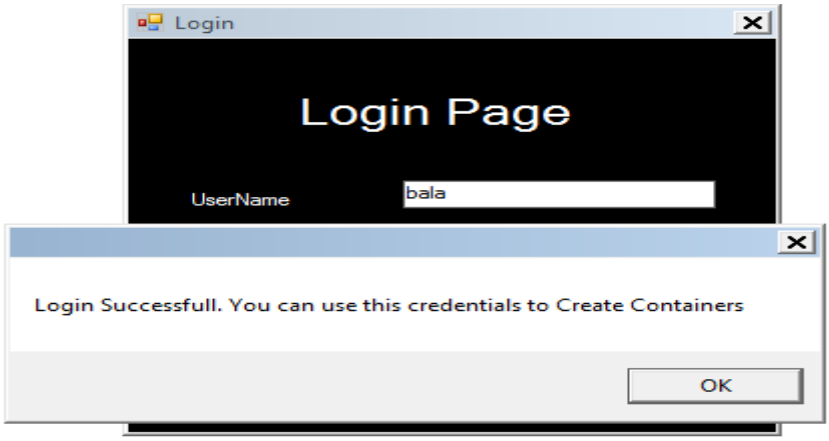
User Creation:

A. Login

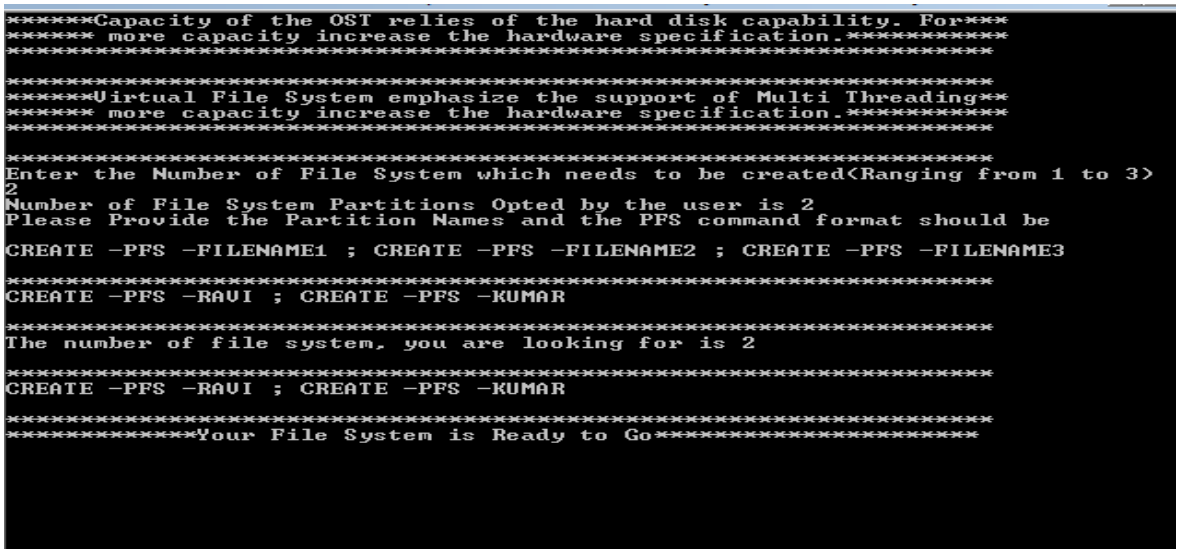


B. New user

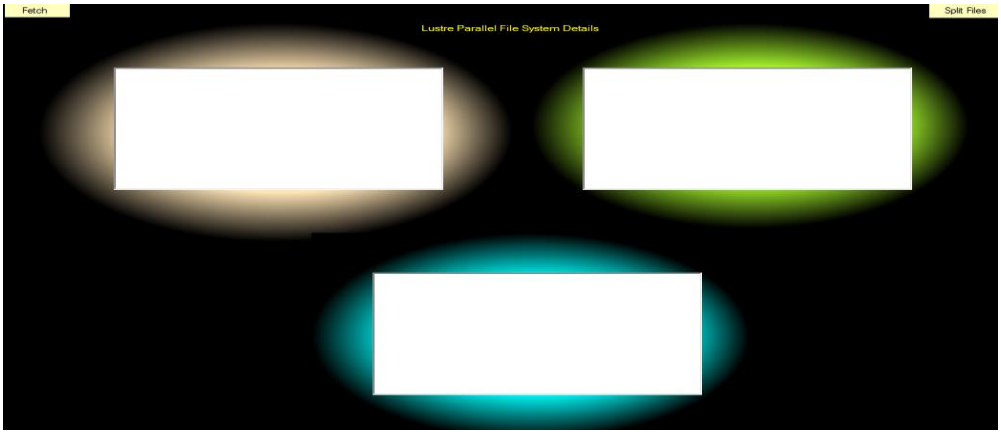
C. Logging in with valid username and password:



D. File Generator Generator



E. Parallel File System



VII CONCLUSION

We 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, 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.

REFERENCES

[1] R. N. Rodrigues, L. L. Ling, and V. Govindaraju, "Robustness of multimodal biometric fusion methods against spoof attacks," J.Vis. Lang. Comput., vol. 20, no. 3, pp. 169–179, 2009.

[2] P. Johnson, B. Tan, and S. Schuckers, "Multimodal fusion vulnerability to non-zero effort (spoof) imposters," in IEEE Int'l Workshop on Inf. Forensics and Security, 2010, pp. 1–5.

[3] P. Fogla, M. Sharif, R. Perdisci, O. Kolesnikov, and W. Lee, "Polymorphic blending attacks," in Proc. 15th Conf. on USENIXSecurity Symp. CA, USA: USENIX Association, 2006.

[4] G. L. Wittel and S. F. Wu, "On attacking statistical spam filters," in 1st Conf. on Email and Anti-Spam, CA, USA, 2004.

[5] D. Lowd and C. Meek, "Good word attacks on statistical spam filters," in 2nd Conf. on Email and Anti-Spam, CA, USA, 2005.

[6] C. Upendra ,G.Gopichand," Survivability and Protection of Nodes and Links from Failures in WDM Mesh Networks",vol: 2-1 (Pages 51-59),), *ISR Journal*, Available:http://isrjournals.org/archives_abstract.php?id=28&t_n=ijarcseit&d_id=66&dm

[7]K Munivara Prasad," Comparative Analysis of Performance of Ad-Hoc Wireless Routing Protocols Based On Topology Using Qualnet",vol: 1-1 (Pages 14-19),), *ISR Journal*,Available:http://isrjournals.org/ archives _ abstract.php?id=28&t_n=ijarcseit&d_id=66&dm