SEMANTIC FILE SYSTEMS

Edra Fresku[1]
Arinela Anamali[2]

[1]IT Department, Regional Hospital of Shkodra, Albania, e-mail: freskuedra@hotmail.com
[2]IT Department, AgnaGroup, Tirana, Albania, e-mail: arinela_anamali@yahoo.com

Abstract

In this paper we will present the study on the integration on a semantic approach in a peer-to-peer file system implementation. Since a few years, various studies have been conducted by scientists to propose either semantic or peer-to-peer file systems approaches in separate ways. Indeed, semantic file systems deal with the ability for a system to provide flexible associative access to its contents by extracting attributes from files whereas peer-to-peer approaches aim at using the aggregate storage capacity of numerous computers gathered in a decentralized peer-to-peer network. A semantic file system is an information storage system that provides flexible associative access to the system's contents by automatically extracting attributes from files with file type specific transducers. We have tried to understand Pastis, a Java peer-to-peer file system implementation based on an open source implementation of Pastry/PAST and study the integration of a semantic concept into it for further extensions. Bringing together a semantic and a peer-to-peer file system is a relevant topic. The resulting system will provide rapid attribute-based access to the system's contents and take advantage of the interesting storage capacity provided by an arbitrary large number of computers connected to the Internet. The deep study of Pastis allowed us to identify locations where intervention could be made in order to integrate semantics. It will be a relevant step to join these two concepts in order to build a completely decentralized system that offers users the possibility to search contents in an easier and faster way.

Keywords*: semantic file systems, P2P systems*

1. INTRODUCTION

A semantic file system is an information storage system that provides flexible associative access to the system's contents by automatically extracting attributes from files with file type specific transducers. Associative access is provided by a conservative extension to existing tree-structured file system protocols, and by protocols that are designed specifically for content based access. Automatic indexing is performed when files or directories are created or updated.

The automatic indexing of files and directories is called "semantic" because user programmable transducers use information about the semantics of updated file system objects to extract the properties for indexing. Through the use of specialized transducers, a semantic file system "understands" the documents, programs, object code, mail, images, name service databases, bibliographies, and other files contained by the system. For example, the transducer for a C program could extract the names of the procedures that the program exports or imports, procedure types, and the files included by the program. A semantic file system can be extended easily by users through the addition of specialized transducers. Associative access is designed to make it easier for users to share information by helping them discover and locate programs, documents, and other relevant objects. For example, files can be located based upon transducer generated attributes such as author, exported or imported procedures, words contained, type, and title. A semantic file system provides both a user interface and an application programming interface to its associative access facilities. User interfaces based upon browsers [Inf90, Ver90] have proven to be effective for query based access to information, and we expect browsers to be offered by most semantic file system implementations. Application programming interfaces that permit remote access include specialized protocols for information retrieval, and remote procedure call based interfaces. It is also possible to export the facilities of a semantic file system without introducing any new interfaces. This can be accomplished by extending the naming semantics of files and directories to support associative access. A benefit of this approach is that all existing applications, including user interfaces, immediately inherit the benefits of associative access.

Semantic file systems can provide associative access to a group of file servers in a distributed system. This distributed search capability provides a simplified mechanism for locating information in large nationwide file systems.

Semantic file systems should be of use to both individuals and groups. Individuals can use the query facility of a semantic file system to locate files and to provide alternative views of data. Groups of users should find semantic file systems an effective way to learn about shared files and to keep themselves up to date about the status of group projects. As workgroups increasingly use file servers as shared library resources we expect that semantic file system technology will become even more useful. Because semantic file systems are compatible with existing tree structured file systems, implementations of semantic file systems can be fully compatible with existing network file system protocols such as NFS [SGK+85, Sun88] and AFS [Kaz88]. NFS compatibility permits existing client machines to use the indexing and associative access features of a semantic file system without modification. Files stored in a semantic file system via NFS

will be automatically indexed, and query result sets will appear as virtual directories in the NFS name space. This approach directly addresses the "dusty data" problem of existing UNIX file systems by allowing existing UNIX file servers to be converted transparently to semantic file systems.

Peer-to-peer (P2P) systems are distributed systems without any central authority and with varying computational power at each machine. In these systems, distributed computing nodes of equal roles and capabilities exchange information and services directly with each other. The big advantage of P2P systems is that the resources of many users and computers can be brought together to yield large pools of information and significant computing power and storage space.

Furthermore, because computers communicate directly with their peers, network bandwidth is better utilized. A lot of applications are based on P2P systems. The most popular one is file sharing. But there are also storage, group communication and computing applications.

The goal was to study semantic file systems. We focused on Pastis, a peer-to-peer file system prototype based on an open-source implementation of Pastry/PAST P2P system. As P2P systems are becoming more and more popular, an analysis on how to bring together semantic and P2P file systems is a relevant topic.

The main contribution of this paper is to show that combining semantic approach and P2P file system is possible and it can be implemented reasonably in practical terms.

2. APPROACHES OF SEMANTIC FILE SYSTEM ARCHITECTURES

File semantics can be seen on various levels:

- *definitional* (e.g file extension)
- *associative* (e.g keywords in file's contents)
- s*tructural* (e.g. physical and logical organization of the data, including intra- and inter-file relationships)
- *behavioral* (e.g. viewing and modification semantics, change management)
- *environmental* (e.g creator, revision history) or other information related to the file

Current file systems(Microsoft File System, UNIX File System, NFS,etc.) provide some degree of organized semantic content. However, they are still far from semantic file systems (SFS) which intend to offer a more extensive and open data model approach with associative, structural, and behavioral information.

Approaches to SFS architectures can be broadly classified into **integrated** and **augmented** approaches. Integrated approaches incorporate extended semantic features directly within the file system. Augmented approaches provide these features via an evolutionary path that augments the traditional file system interface, thus allowing traditional file manipulation interfaces to remain unchanged.

- **Integrated Approach**

Integrated file systems present the user with a new and improved file system that is incorporated into existing file systems. These systems provide an integrated data model whereby file data and file metadata are represented within one data model, and the two are implicitly synchronized. These systems may provide a type definition language similar to OMG IDL, or some other means to define object structure and perhaps behavior as well. The IDL is used to define the file system's data model so that file content and metadata can be stored together (e.g. file structure, author, revision histories, content indexing, etc.).

The rich data models provide applications with an easier access to metadata and also help them store efficiently file-related data. However, integrated file systems may require the user to migrate to a brand new operating system, to a new version of an existing operating system and to acquire new versions of applications which will access data via the new data model.

We can classify the level of integration with the current file and operating system as either **loosely coupled** or **tightly-coupled**. Both types provide a file system with an integrated view of data and meta-data (using the defined data model); however, the loosely-coupled systems sit on top of current file systems providing a separate file store which is inaccessible via any lower layers of the file system, while tightly-coupled systems are implemented as one or more layers within the file system.

- **Augmented approach**

The **augmented semantic file system** approach provides an evolutionary path to SFS architectures. It leaves the traditional file system interfaces unchanged, while providing a parallel content abstraction view of the file's content. Tools layered on the content abstraction can be more intelligent about querying and manipulating file information(e.g. Microsoft Windows second search engine, Google Search Desktop). At present, augmented approaches are more prevalent than integrated approaches, as they place fewer demands on the end-user.

Augmented SFSes (ASFS) provide a content abstraction layer on top of traditional file systems to facilitate smarter querying and manipulation of files. Most advanced implementations of augmented SFSes use these content abstractions for either *query shipping* or *index shipping*. *Query shipping* directs a repository independent user query to the appropriate files (e.g. Microsoft search engine, Google Search Desktop). *Index shipping* extracts the contents of files and makes them available as meta-data (indices) to a user level query system.

Files are abstracted into logical collections, or *domains*, managed by a *domain manager*. Domain managers can be subdivided into a content summarization engine, and a query engine. The summarization engine executes *content summarizer scripts* on individual files to extract the values of type specific attributes for the particular file type. In an *index-shipping* ASFS architecture, indices and other meta-information are shipped from domain managers to higher levels of knowledge brokers using a specific exchange language .This allows the knowledge brokers to directly process application queries.

- **Augmented .vs. Integrated Approaches**

Integrated and augmented approaches differ not in their vision of greater access to file content, but in the manner in which they get there. Integrated approaches are the quicker way to get there, if end-users are willing to accept more drastic changes to their operating systems. Augmented approaches accept the OS as is (especially the file system) and provide *less perfect implementations* of the functionality that a user can expect from an integrated SFS, but with almost no negative perturbation to the user's current file access capabilities. At the same time, augmented SFSes can take advantage of progress in the OSes to get progressively closer in elegance and completeness to the capabilities of an integrated SFS.

To illustrate the point about less perfect implementations, augmented file systems may have to work with operating systems that do not notify them of changes to files. They may therefore have to build a custom file polling scheme that is less efficient than file notification. Even after file notification, it would be left to augment SFSes to detect incremental changes to file content. Here again, an integrated system may be able to modify the file system to support intermediate level content abstractions (e.g. model a file as a collection of text chunks).

Another example is providing transactional capabilities in the SFS. An augmented SFS can easily provide transactional capabilities on the extracted meta-data by storing it in a database. It cannot, however, maintain ACID properties across the meta-data and the file data, since it has no control over the ACID properties of file updates. However, if the OS were to support a transactional file system with event notification for file transactions, an augmented SFS could implement such ACID properties in a manner similar to multi-database transactions.

In summary, while augmented SFSes are tied down by the current capabilities of their OS substrate, they can take advantage of subsequent OS improvements (e.g. less latency in notification of data change, greater access to file content) in much the same way as integrated implementations. As the actual OS gets closer to the idealized SFS OS, an augmented SFS can come closer in functionality and implementation to an integrated one.

3. PASTIS FILE SYSTEM

Peer-to-peer systems can be characterized as distributed systems in which all nodes have identical capabilities and responsibilities and all communication are symmetric.

Pastis is a peer-to-peer file system application that runs on top of Pastry and Past layers. Before presenting Pastis, we will introduce these two layers.

One of the key problems in large-scale peer-to-peer applications is to provide efficient algorithms for object location and routing within the network. Pastry is a generic object location and routing scheme intended to solve that problem.

- **Pastry Layer**

Based on a self-organizing overlay network of nodes connected to the Internet, Pastry is intended as general substrate for construction of a variety of peer-to-peer Internet applications

like global file sharing, file storage, group communication and naming services. It is completely decentralized, fault-resilient, scalable and reliable. Moreover, it has good local route properties.

Any computer connected to the Internet and running a Pastry node software can act as a Pastry node, subject only to application-specific security policies. Each node in the Pastry network has a unique numeric identifier (nodeId) provided by the system when it joins the network. For routing purposes, nodeIds and keys (messages keys) are thought as a sequence of digits with base 2^b.

When presented with a message and a numeric key, a Pastry node efficiently routes the message to the node whose nodeId is numerically the closest to the numeric key among all live nodes in the Pastry network. This is accomplished as follows. In each routing step, a node normally forwards the message to a node whose nodeId shares with the key at least one digit(or b bits) longer than the prefix shared with the present node's id. If no such node is known, the message is forwarded to a node whose nodeId shares with the key a prefix as long as the present node but is numerically closer to the key than the current node's id. The expected number of routing steps is O(log N) where N is the number of Pastry nodes in the network. At each Pastry node along the route that a message takes, the application is notified and may perform application-specific computations to the message.

Pastry seeks to minimize the distance messages travel. Each pastry node keeps track of its immediate neighbors in the nodeId space with a neighborhood set as well as a routing table and a leaf set, and notifies application of new nodes arrival, node failures and recoveries. Because nodeId is randomly assigned, with high probability, the set of nodes with adjacent nodeIds is diverse in geography, ownership,etc. Each entry in the routing table contains the IP address of one of the many potential nodes whose nodeId has the appropriate prefix.If no such node is found, the entry is left empty. The neighborhood set M contains the nodeIds and IP addresses of the |M| nodes that are closest (according to the proximity metric) to the local node. The leaf set L is the set of nodes with the |L| / 2 closest larger nodeIds and the |L| / 2 nodes with the closest smaller nodeIds, relative to the present node's nodeId.

- **PAST Layer**

PAST is an Internet-based, peer-to-peer global storage utility built on Pastry, which aims to provide strong persistence, high availability, scalability and security. It is a self-organizing peer-to-peer Internet application.

The PAST system is composed of nodes connected to the Internet, where each node is capable of initiating and routing clients requests to insert or retrieve files. Inserted files are replicated on multiple nodes to ensure persistence and availability. Additional copies of popular files may be cached on any PAST node to balance load query.

A storage utility like PAST is attractive for several reasons. First, it exploits the multitude in diversity (geography, ownership, administration, jurisdiction, etc.) of nodes of the Internet to achieve strong persistence and high availability. A global storage utility also facilitates the sharing of storage and bandwidth, thus permitting a group of nodes to jointly store or publish content that exceeds the capacity of any individual node.

PAST differs from conventional file systems in that the files it stores are all associated with a quasi unique field that is generated at the time of the file's insertion in the system. Therefore, files inserted into PAST are immutable since a file cannot be inserted several times with the

same fileId. Files can be shared at the owner's discretion by distributing the fileId (potentially anonymous) and, if necessary, a decryption key. PAST does not support a delete operation. Instead, the owner of a file may reclaim the storage associated with the file, which does not guarantee that the file will be unavailable. These weaker semantics avoid agreement protocols between the nodes storing the file.

An important issue in P2P systems is security. The inherently unsafe nature of the resources used in P2P systems, i.e the Internet, the computers running the P2P software arises a great number of security issues that must be taken into account. The minimum security guarantee is the integrity of the objects stored by the system. In PAST, security is achieved with the use of smartcards that every node and user holds and cryptographic techniques such as one-way hash functions and digital signatures. A private/public key is associated with each smartcard. The smartcards generate and verify various certificates used during request and reclaim operations and they maintain storage quotas allowed to each client.

Before a file is inserted in the PAST system using Pastry (presented earlier), a file certicate is generated which contains an assigned fileId, a replication factor k, a random salt, a cryptographic hash of the file's contents and the insertion date. The replication factor k depends on the availability and persistence requirements of the file and may vary between files. The fileId is computed as a secure hash (SHA-1) of the file name , the owner's public key and a randomly chosen salt.

When a file is inserted in PAST, Pastry routes the files to the k nodes whose nodeIds are numerically the closest to the 128 most significant bits of the file identifier (fileId). Each of these nodes then stores a copy of the file. A lookup request of a file is routed towards the live node whose nodeId is numerically the closest to the requested fileId.

- **Pastis' design**

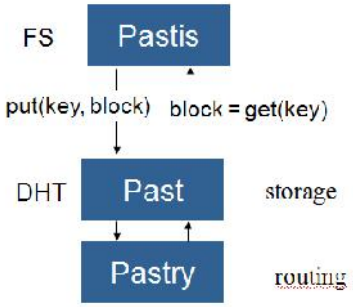
Pastis is a highly scalable, completely decentralized multi-writer peer-to-peer file system built on PAST and Pastry underlying layers.

As a peer-to-peer file system, it offers the opportunity to share a given file or portion of the file system between an arbitrary number of nodes and users connected on the Internet. Pastis differ from other peer-to-peer file systems proposed by different research groups in that it is designed to scale to hundreds of thousands of nodes and offer read-write access to a large community of users. Moreover, modifications in the original open-source implementation of Past/Pastry have been made to increase performance by reducing network communication.

For each file, the system stores an inode-like object which contains the file's metadata, much like the information found in a traditional inode. Each inode is stored in a Past Public-Key Block (PKB). The corresponding private-public key pair is generated when the file described by this inode is created, and is stored encrypted within the inode itself. File and directory contents are stored in fixed-size immutable blocks named Past Content- Hash Blocks (CHBs). The address of each CHB block is obtained from the hash of the block's contents and is stored within the file's inode block pointer table. Single, double and triple-indirect blocks are used to limit the size of

the inode's block pointer table. The contents of a directory are stored in the same way as those of a regular file. Each directory inode points to a set of CHBs containing the directory entries, which consist basically of a file name and the Past address of the corresponding inode. PKBs and CHBs are inserted into Past DHT(Distributed Hash Table) – abstraction offered by the Pastry network. The DHT abstraction provides the same functionality as a traditional hash table, by storing the mapping between a key and a value. This interface implements a simple store and retrieves functionality, where the value is always stored at the live overlay nodes/s to which the key is mapped by the KBR layer. Values can be objects of any type. In the case of Pastis, these objects are blocks (PKBs and CHBs).

A distributed hash table (DHT) provides two operations: put (key, value) and value = get (key). In Pastis, the key of a CHB is a hash of the block's contents and that of a PKB is the public key associated with the inode stored.



Security in the file system is achieved by signing a file inode before storing it in a PKB and by associating a CHB with a key obtained from hash functions. Modifying a file or directory in Pastis requires updating the PKB in which its inode is stored, but it is also involves the insertion of new CHBs reflecting the newly written data.

Pastis file system support two consistency models: close-to-open consistency and a variant of read your- writes models. The close-to-open consistency model consists in not propagating local write operations until the file is closed. Similarly, once a file has been opened, the local client need not check whether the file has been modified by other distant clients. A cached copy is used until the file is closed. This model requires that the latest version of a file be retrieved. The read-your-writes model ensures that read operations always reflect all previous local writes.

The main modification made by the Pastis file system to the PAST/Pastry implementation is the introduction of a constraint over the block's metadata when performing a Past lookup call. This constraint allows to choose among the potential replica encountered by the Past lookup message. In this way, if the first replica encountered does not meet the required criteria, the messages continues its path until another valid replica is found, or it returns an error message to the client. In this case (error message), all inode replicas are retrieved and the client will be able to keep the one with the most recent timestamp.

4. INTEGRATION OF SEMANTICS IN PASTIS

Bringing together a semantic and a peer-to-peer file system is a relevant topic. The resulting system will provide rapid attribute-based access to the system's contents and take advantage of

the interesting storage capacity provided by an arbitrary large number of computers connected to the Internet.

As we mentioned earlier, the whole Pastis system is made of three layers: the Pastry layer for the routing of messages, the Past layer for replication and storage management and the Pastis file system. As adding a semantic approach to Pastis file system will consist in managing files' metadata, may think that the best way to implement it would be to work in the Pastis layer. The advantage of this solution is that the other two underlying layers will remain unchanged, then avoiding complex changes to the current Pastis system. The top layer will be the only one carrying the semantic approach.

The deep study of Pastis allowed us to identify locations where intervention could be made in order to integrate semantics. Integration of semantics in Patris can be as follow:

1. First, implement simple changes in the current structure by adding Java semantic objects to carry some metadata and linking them to the current classes. Metadata will include simple attributes like the file's owner, creation date, last modification date. These attributes can be easily obtained
2. Second, increase the metadata size with a lot more attributes
3. Third, develop an interface between the file system and indexing tools (filters) that already exist. These filters will take charge of extracting files' attributes and the interface will generate the corresponding Java semantic objects to store these attributes.

Thus, in the first phase, the first intervention to be made would be to create a new Java object containing the semantic attributes. Then, include an class attribute in the “Inode” class whose type is the Java semantic object created earlier. Indeed, the Inode class is the system's key class as it is associated with all created files. After that, it is essential to link the new Java semantic object with the other Java files and interfaces as FileInodeFactory.java, DirInodeFactory.java and SymlinkInodeFactory.java. Finally, all the semantic attributes should be stored in a table in which links are made between an attribute and all the corresponding blockIds.

Once this work achieved, a great step towards semantic approach will be made. The second and third phases presented earlier will then be able to be implemented.

5. CONCLUSION

Semantic file systems and P2P file systems have a great future because of the large functionalities and improvements that they provide. Until now, researches on the systems have been conducted in separate ways. However, as this study of Pastis shows us, it will be a relevant step to join these two concepts in order to build a completely decentralized system that offers users the possibility to search contents in an easier and faster way. Pastis file system is a prototype still under development.

REFERENCES

F. Picconi, J.M. Busca, P. Sens, *"An experimental evaluation of the Pastis peer-to-peer file system under churn"*, February 2007

J.M. Busca, F. Picconi, P. Sens, *"Pastis: a Highly-Scalable Multi-User Peer-to-Peer File System"*

A.W. Leung, A.P.Wood, E.L. Miller, *"Copernicus: A Scalable, High-Performance Semantic File System"*,October 2009

P.Mohan, Raghuraman, V. S and Dr. Arul Siromoney, *"Semantic File Retrieval in File Systems using Virtual Directories"*

S. Faubel, C. Kuschel, *"Towards Semantic File System Interfaces"*

D. Margo,R. Smogor,*"Using Provenance to Extract Semantic File Attributes"*

M. Mahalingam, C. Tang, Z. Xu,*"Towards a Semantic, Deep Archival File System"*, 2002

D.K. Gifford, P. Jouvelot1, M. A. Sheldon, J. W. O'Toole, Jr**,** *"Semantic File Systems"*

http://regal.lip6.fr/spip.php?rubrique18

http://www.objs.com/survey/OFSExt.htm