

# Semantic file Systems

Georges N'dou KOUAME, Kouassi N'GORAN  
February 2005

## **Abstract**

This paper presents our 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. In our work, 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.

## **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[4]. A transducer is a filter that takes as inputs the contents of a file and outputs the file's entities and their corresponding attributes. An attribute is a field-value pair, where a field describes a property of a file (such as its author, or the words in its text) and a value is a string or an integer. The automatic indexing of files and directories, performed at creation or later (at update for example), 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, one per file type, a semantic file system understands the documents, programs, object code, mail, images, name service databases, bibliographies and other files contained by the system.

Associative access is designed to make it easier for users to share information by helping them discover and locate programs, documents and other files using various attributes and a query facility. This tool-independent functionality offers a finer-grained and uniform view and access to file data. Thus, it is commonly admitted that semantic file systems present a more effective storage abstraction than do traditional tree-structured file systems for information sharing and command level programming.

Peer-to-peer (P2P) systems are distributed systems without any central authority and with varying computational power at each machine [12]. In these systems, distributed computing nodes of equal roles and capabilities exchange information and services directly with each other [13]. 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.

Our goal was to study semantic file systems. Since we could not study all semantic systems at once and because we needed to have access to an open-source prototype, 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 for us 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.

The remainder of the paper is organized as follows. In Section 2, we present the two common approaches in the building of semantic file systems and an example of a semantic file system implementation. Section 3 introduces Pastis file system and Section 4 presents the installation and running of the Pastis prototype. Finally, Section 5 gives an overview of the integration of a semantic concept within Pastis and Section 6 concludes.

## **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), *structural* (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. [3]

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.[3]

### **2.1 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 model provides 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 unaccessible via any lower layers of the file system, while tightly-coupled systems are implemented as one or more layers within the file system[3].

## **2.2 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.

## **2.3 An example of implementations of semantic file systems**

### **HAC**

HAC[14] stands for Hierarchy And Content. It is a file system proposed by Burra Gopal and Udi Mamber in 1999.

HAC can be classified as an augmented semantic file system approach. Indeed, HAC's objective is to provide a smooth transition to semantic file systems by building a file system that will have the benefits of both hierarchical and semantic file systems, and allow users to choose among their features at any time.

It allows the use of the file system as a regular traditional hierarchical file system with no need to change anything. The added features of a content-based access (CBA) are made optional under the control of the user. They can cover the whole file system, any part of it, or none at all. They can be discarded and added at any time. Consequently, the design is based on a hierarchical file system with an added content-based access, rather than a given content-based mechanism extension.

Burra Gopal and Udi Mamber started with a hierarchical naming system and extended it to support query (content) based naming. Their approach shows many advantages: it gives users a lot of flexibility and power, and at the same time it makes the system easy and intuitive to use.

## **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, let us 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.

### **3.1 Pastry**

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.[9].

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 neighbours 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. More details on Pastry can be found in [9].

### **3.2 PAST**

PAST[10] 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 filesystems 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 certificate 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. More details on PAST can be found in [10, 11].

### **3.3 Pastis ' design**

Pastis[6,7] 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 retrieve 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. Other details on Pastis file systems are available in [6,7].

#### **4. Installation and running of Pastis prototype**

This section gives a deeper description of the Pastis prototype (version 1.0) which is entirely written in Java 1.4 and can downloaded at <http://regal.lip6.fr/projects/pastis/> (as of today February 10<sup>th</sup>, 2005).

It also gives clues on how to install it in a NFS network. It did take us a lot of time to install and configure the prototype in order to make it run on your network, as Pastis is still under development and there is no documentation. We feel sure this section will help people interested in studying Pastis or continuing our work, particularly our teachers.

##### **4.1 Modifications made in Pastis scripts**

To install and run Pastis you have to configure some parameters. Please adapt the following lines to your computer environment.

###### Script setjavapath (in the Pastis directory)

Set the correct JAVAPATH( your JAVAPATH) in this file

Make sure to use JDK1.5.x

You can now install Pastis with these comands:

- ". setjavapath"
- "make"

###### Script pastenv ( in the Pastis/bin directory)

This script makes an execution call to the script “javaenv”. Make sure to configure your environment correctly or include the correct path in the file in order to run the “javaenv” script. Adapt the variable \$MYHOME, relative to the Pastis directory, to your own environment.

###### Script distpast (in the Pastis/bin directory)

A number of execution calls are made to other scripts within the same directory. Make sure to configure your environment correctly or include the correct path in the file in order to run these scripts.

## **4.2 Evaluation of the prototype**

In order to evaluate the performance of the prototype, a Java program, “Andrew.java”, has been developed. This Java program generates a pattern of file accesses equivalent to that of an Andrew Benchmark[15]. This program allows to copy the contents of an entire directory in another one. For this, it creates directories, copy files, read file attributes, read file contents, and simulate a “make” command.

(Andrew.java can be found under <<Home\_dir>>/Pastis/src/pastis/src/lip6/pastis/test)

In order to test Andrew.java, we have written a script, “andrew” similar to the “aab” script in Pastis/bin. The only difference between the two scripts is that, in “andrew” script, we call the class “Andrew”. After this modification, it is possible to test the prototype with the command below:

```
andrew -source <source_dir> -dest <target_dir>
```

All the files of the source directory will then be copied in the destination directory. If there are any C files in the source directory, they will be compiled in the destination directory. The approximate copy time is also obtained.

Note: We did not manage to launch the Andrew benchmark script, “andrew”, simultaneously on several machines due to our network configurations.

## **4.3 Role of the main Pastis Java files**

Here under is a description of the functionalities of the main Java files of Pastis:

### **4.3.1 Description of Inode-like objects**

**File: Inode.java** (abstract class).

(FileInode.java, DirInode.java, SymlinkInode.java implement this abstract class)

Location: <Home\_dir>/Pastis/src/pastis/src/lip6/pastis/block

The inode object is made of 4 attributes:

- Inode type (File TYPE\_FILE=0, Directory TYPE\_DIR=1, Symbolic link TYPE\_SYMLINK =2)
- An attributes Object, that gathers all the attributes relative to the file, directory or symbolic link
- Security information: UNIX “uid”, “gid” and “mode”.
- an InodeStamp object. It represents either the last update time of the inode on the root node or the version number of that update (along with the owner's public key in order to sort the different updates of the inode)

Depending on the type associated with the inode, different tasks are performed. For more details, check FileInode.java, DirInode.java, SymlinkInode.java )

### **4.3.2 Description of data blocks**

**File: BlockId.java**

Location: <Home\_dir>/Pastis/src/pastis/src/lip6/pastis/block

This class gives methods on how to handle blocks ids. It also determines the id of the pastry node

on which the block id is stored.

**File: BlockIdList.java**

Location: <Home\_dir>/Pastis/src/pastis/src/lip6/pastis/block

This class gives the list of a file's or directory's blockIds.

**4.3.3Creation of Inodes**

FileInodeFactory.java, DirInodeFactory.java and SymlinkInodeFactory.java classes allow the creation of inodes for files, directories and links.They

- set the inode's type
- add attributes in inodes
- add the security information
- add the InodeStamp

**4.3.4Data Block Storage**

**File:** BlockStore.java

Data blocks are either PKBs (0) or CHBs(1).

Methods implemented:

- Find a block
- Add a block
- Delete a block
- Check the existence of a block

**4.3.5Past interface**

**File:** BlockStorePast

**4.3.6 Key generation**

File: SecMan.java

Functionalities: Key generation, decryption, encryption

**4.3.7Pastis Interface**

Location: <HOME\_DIR>/Pastis/src/pastis/src/lip6/pastis

File: Pastis.java (Java interface)

Functionalities:

- file creation
- setting of attributes
- file deletion
- file re-naming

PastisImpl.java implements Pastis.java

**4.4 Other main scripts of Pastis file system**

**Past**

The “past start” command launches the Pastis/Past application on the local computer. Other options

are:

- stop (to stop the application) - “past stop”
- status (to check whether the past application is running or not) - “past status”

## **Distpast**

Command: distpast <Machinefile> <number><command\_option>

This script launches the application “past” on the nodes listed on <Machinefile> with the option <command\_option>. <number> is the number of nodes listed in <MachineFile>

Example: distpast MachineFile 3 start

## **5. Integration of semantics in Pastis**

Bringing together a semantic and a peer-to-peer file systems 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 have seen 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, we 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.

Our deep study of Pastis allowed us to identify locations where intervention could be made in order to integrate semantics. We propose the following methodology:

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.

## **6. Conclusion**

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

Pastis file system is a prototype still under development. However, we believe that our study will be a good help for people who intend to study it and add more functionalities. The future work that our work implies is a practical implementation of semantics in Pastis. Our orientations will surely contribute to that goal.

## References

[1] Google, <http://www.google.com>

[2] Pastis Homepage, [http://regal.lip6.fr/projects/pastis/pastis\\_en.html](http://regal.lip6.fr/projects/pastis/pastis_en.html)

[3] Semantic file systems, <http://www.objs.com/survey/OFSExt.htm>

[4] David K. Gifford, Pierre Jouvelot, Mark A. Sheldon, and James W. O'Toole, Jr. *Semantic file systems*. Programming Systems Research Group, MIT Laboratory for computer science.

[5] Craig A. N. Soules, Gregory R. Ganger. Toward Automatic Context-based Attribute Assignment for *Semantic File Systems*. Parallel data laboratory, Carnegie Mellon University, June 2004.

[6] J-M. Busca, F. Picconi, P. Sens. *Pastis: a highly-scalable multi-user peer-to-peer file system*. INRIA Technical Report.

[7] F. Picconi, J-M. Busca, P. Sens. *Exploiting network locality in a decentralized read-write peer-to-peer file system*. International Conference on Parallel and Distributed Systems 2004 (ICPADS 04), New Port Beach, California, USA.

[8] J-M. Busca, F. Picconi, P. Sens. *Pastis: un système de fichiers pair à pair multi-écrivain passant l'échelle*. In *DistRibUtIon de Données à grande Echelle 2004 (DRUIDE 04)*, Domaine du Port-aux-Rocs, Le Croisic, France.

[9] Anthony Rowstron, Peter Druschel. *Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems*. 18<sup>th</sup> IFIP/ACM International Conference on distributed systems platforms (Middleware 2001), Heidelberg, Germany, November 2001.

[10] Anthony Rowstron, Peter Druschel. *PAST: A large-scale, persistent peer-to-peer storage utility*. In *Proc. HostOS VIII*, Schloss Elmau, Germany, May 2001.

[11] Anthony Rowstron, Peter Druschel. *Storage management and caching in PAST, a large-scale, persistent, peer-to-peer storage utility*. In *Proc ACM SOSP'01*, Banff, Canada, October 2001.

[12] James Aspnes, Zoë Diamadi, Gauri Shah. *Fault-tolerant routing in peer-to-peer systems*. Twenty-first ACM Symposium on Principles of distributed computing, 2002. <http://cs-www.cs.yale.edu/homes/aspnes/podc02-full.pdf>

[13] Beverly Yang, Hector Garcia-Molina. *Comparing Hybrid peer-to-peer systems*. Computer Science department, Stanford University.

[14]Burra Gopal, Udi Mamber. *Integrating Content-Based Access Mechanisms with Hierarchical File Systems*. In Proc of the third symposium on operating systems and design, New Orleans, Louisiana, USA, February 1999.

[15] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R.Sidebotham, and M. West. *Scale and performance in a distributed file system*. In ACM Transactions on Computer systems, volume6, February 1988.0