SEARCH ON THE CLOUD FILE SYSTEM

Rodrigo Savage[1], Dulce Tania Nava[1], Norma Elva Chávez[1], Norma Saiph Savage[2]
Facultad de Ingeniería, Departamento de Computación, Universidad Nacional Autónoma de México (UNAM) , Mexico[1]
Computer Science Department, University of California, Santa Barbara,USA[2]
{rodrigosavage, opheliac}@comunidad.unam.mx, norma@fi-b.unam.mx, saiph@cs.ucsb.edu

ABSTRACT

Research in peer-to-peer file sharing systems has focused on tackling the design constraints encountered in distributed systems, while little attention has been devoted to the user experience: these systems always assume the user knows the public key of the file they are searching. Yet average users rarely even apprehend that file public keys exist. File sharing systems which do consider the user experience and allow users to search for files by their name, generally present centralized control and they show several severe vulnerabilities, that make the system unreliable and insecure. The purpose of this investigation is to design a more complete distributed file sharing system that is not only trustable, scalable and secure, but also leverages the user's cognitive workload. We present a novel algorithm that by mining a file's information designates relevant keywords for the file automatically. These keywords are later utilized for the file search and retrieval. We also designed a metric for assigning relevancy to the files retrieved in a search, bettering the search results. We also create a modern mechanism for enabling file searches based on categories. Search on the Cloud is built on Pastry. Our system integrates these components, as well as good design principals from previous distributed file sharing systems to offer a trustable, scalable, secure and novel distributed file sharing system that an average user could utilize for file search. Our system is named "Search on the Cloud". The novelty of our approach is that our system provides an intuitive search modality, while still preserving an entirely distributed approach.

KEY WORDS
DISTRIBUTED COMPUTING, PEER-TO-PEER COMPUTING, CLOUD COMPUTING, HETEROGENEOUS COMPUTING, GRID COMPUTING, DISTRIBUTED SHARED MEMORY, NETWORK SECURITY, NETWORK PERFORMANCE, INTERCONNECTION NETWORKS, ALGORITHM DESIGN, ALGORITHMS FOR HETEROGENEOUS SYSTEMS, APPLICATIONS, COMMUNICATION ALGORITHMS, LOAD BALANCING, MESSAGE ROUTING, PROTOCOL DESIGN, SECURITY AND RELIABILITY, META-DATA CLUSTERING

1. Introduction

Distributed systems are a collection of autonomous computers connected through a network. A distributed system permits the computers to share resources and activities, allowing the end user to perceive the system as a powerful single computing machine. Peer-to-peer systems are a particular type of distributed systems, where all computers, also known as nodes, present identical responsibilities and capabilities. Peer-to-peer systems have many advantages over traditional centralized systems: they present better availability, scalability, fault tolerance, lower maintenance costs as well as lower operation and deployment costs. The drawback of these systems is that they encounter several design challenges. For example the system must remain functional, despite the varying number of uncontrolled participating nodes. Furthermore the system must be decentralized and symmetric; load should also be balanced among all nodes. Additionally, despite the system's size, data search on peer-to-peer systems must be fast and robust (scalable). A vast number of researchers have concentrated on solving the design challenges referred above. A problem that has been widely tackled is the lookup problem. The lookup problem assumes that a node A inserts a file x into the system and moments after, a node B seeks to retrieve the file x. Considering that the node A is no longer online, the lookup problem intends to find the location of a node that has a replica of the file x. Examples of novel architecture algorithms that were proposed to solve the lookup problem are CAN[16], Chord[15], Pastry[14], and Tapestry[17]. Systems that also solved the lookup problem, while presenting a more social design are Napster [18], Fast track[19], Gnutella[18]. Because of the characteristics of these systems, it is possible to utilize them as a base for developing more complex distributed systems such as PAST[14], Pond[20], CFS[21] and bittorrent[1].

PAST is a large scale internet based global storage utility that provided scalability, high availability and security. With PAST users were capable of inserting files into the system and later retrieving them, or retrieving files that other users shared. It is important to note, that to accomplish this operation, the user needed to know the file's public key. PAST looked up files by utilizing Pastry. PAST made several improvements to file sharing,

but because PAST's lookups were based on the file's public key, the system doubtlessly encountered many usability problems. In specific, new users that were unaware of the existence of the public keys, would be incapable of finding their file of interest. To overcome this problem, a centralized web server, that provided the public keys to the files the users were searching for, would be required. But adding a centralized web server to the system would increase the system's vulnerabilities to single points of failure. Additionally PAST did not handle all of the design issues encountered in peer-to-peer systems. Specifically it did not address load balance: PAST made no partition on the files that were inserted. Therefore if a large file was attempted to be added to the system, if it did not fit in one single node, the file would not be inserted, despite the fact that the system as a whole might present sufficient memory.

Another interesting large scale peer-to-peer storage system was Pond, an implementation of OceanStore [20]. Pond presented several improvements and differences over PAST, the only problem was thatPond presented the same usability issue PAST encountered: the system required the user to know the public key of the file they were searching.

A file sharing system, which did consider in more detail the user experience when sharing and seeking files is Bittorrent[1]. Bittorent is a file downloading protocol that together with sites, such as Piratebay.org, Lokotorrent.com and trackers servers provides probably the biggest distributed file-sharing system in the world Web pages supporting Bittorrent function by showing for each available file, its name, size, current numbers of downloaders and seeds, and the name of the person who uploaded the file. To download the file a user clicks on a link that points to a .torrent meta-data file. The .torrent metadata files are stored and distribute among .torrent file servers. This mechanism, permits users to search for files by simply inputting related keywords of the file name and querying a web server. Albeit Bittorrent presented a significant improvement on user experience in file sharing systems, Bittorrent is not a truly distributed system, the .torrent file servers have centralized the search. Additionally the web servers, such as Piratabay and Lokotorrents provide a user interface to locate the correct .torret file necessary for a search, but this creates a window of vulnerability and creates increasingly high maintenance costs. Furthermore, the architectures involving BitTorrent/WebServer/Tracker present several problems, which can be divided into four global types [14]: the first problem involves the web server and the fact that when it switches IP numbers it can be down for significant periods of time. The second issue concerns the mirrors, which rarely survive longer than a few days, due to the high demand of daily visitors. The third problem involves the .torrent file servers, which are occasionally unavailable, blocking all new downloads. The final vulnerability involves the trackers, which are a frequent target for denial-of-service attacks and are costly to operate due to GBytes of daily bandwidth consumption.

The majority of the research in file sharing systems has focused on bettering the design constraints encountered in distributed systems and little attention has been paid to the user experience: These systems require the user to know beforehand the public key of the file they are searching for. It is evident that novice users would have a very difficult time utilizing their services, because they are likely to be unaware of the existence of public keys. As these systems fail to acknowledge the novice user's needs, they are ignoring basic user interface design principles, which state that a variety of users with diverse backgrounds, should be able to interact with the system.

On the other hand, file sharing systems which do present more concern for the user experience, such as Bittorrent, have disregarded many of the principles of distributed systems, and present several severe vulnerabilities.

The aim of this study is to the design a more complete distributed file sharing system that while exhibiting ease of use and transparency for the user, still presents high availability, scalability and is fault tolerant. As well as keeping maintenance, operation and deployment costs to a minimal. Our system, named "Search on the Cloud", allows for a more user intuitive file search. Our system assumes that when an average user searches for a file distinct information about the file is known, such as certain words that appear in the file name or content. Search on the cloud utilizes the user's search query to automatically generate keywords. For each keyword, its public key is fetched and the keyword is then searched for in the network. Within the network, each keyword has an associated meta-data block that holds references to files for which the keyword in question is meaningful. For each file the keyword has an assigned relevancy score. The K files with the highest relevancy score are retrieved and presented to the user. The relevancy score each file has for a certain keyword is assigned upon upload by mining the data file.

Additionally, for usability purposes Search on the Cloud does not follow the conventional tree folder structure. Our system acknowledges that files can potentially belong to different categories (or folders). Therefore file search can be done by selecting multiple categories. For example a user may wish to list all his homework files from freshman year that are related with Artificial Intelligence. This type of query would be difficult to find on a conventional folder system, because the files could be in the Artificial Intelligence folder, or the freshman year folder, or any possible combination of these three folders. With Search on the Cloud's design, this type of query becomes easy and transparent for the user to do. The novelty of our approach is that in difference to previous file sharing systems, we present an intuitive search modality, while preserving the fully distributed characteristics of the system. In the following sections, we present background terminology related with our system,

and provide greater detail of the specifics of our system. Finally conclusions are presented.

2. Background

2.1 Design of Pastry

A node is an active computer that is attached to a network, and is capable of sending, receiving, or forwarding information over a communications channel. Pastry[4] gives each node a randomly chosen key, which conceptually indicates its position on the pastry ring showed in figure 1. The digits in the key space are in base 2b, where b is a parameter typically set to 4 thus forming 128-bit keys. Assuming a network consisting of N nodes, Pastry can route to the numerically closest node to a given key in less than $\lceil log_{2^b}(n) \rceil$ steps under normal operation. Despite node failures, eventual delivery is guaranteed unless $\lfloor |L|/2 \rfloor$ nodes with adjacent node keys fail simultaneously (|L| is usually 16 or 32). Pastry routes messages to the node whose key is numerically closest to the message key. In each routing step, a node normally forwards the message to a node that's key shares with the message key a prefix that is at least one digit (or bits) longer than the prefix that the key shares with the present node's key. If no such node is known, the message is forwarded to a node whose key shares a prefix with the key as long as the current node, but is numerically closer to the key than the present node's key. A node's routing table, is organized into $\lceil log_{2^b}(n) \rceil$ rows with (2^b-1) entries each, thus implying a node state of $\lceil log_{2^b}(n) \rceil * (2b^2 - 1)$. Each entry in the routing table contains the IP address of the potentially nodes whose key have the appropriate prefix; The neighborhood set M contains the node keys and IP addresses of the |M| nodes that are closest (according the proximity metric) to the node. The neighborhood set is useful in maintaining locality properties. The leaf set L is the set of nodes with the |L|/2 numerically closest larger node keys, and the |L|/2 nodes with numerically closest smaller node keys, relative to the present node key. The leaf set is used during the message routing. Typical values for |L| and |M| are 2^B or $2x2^b$.

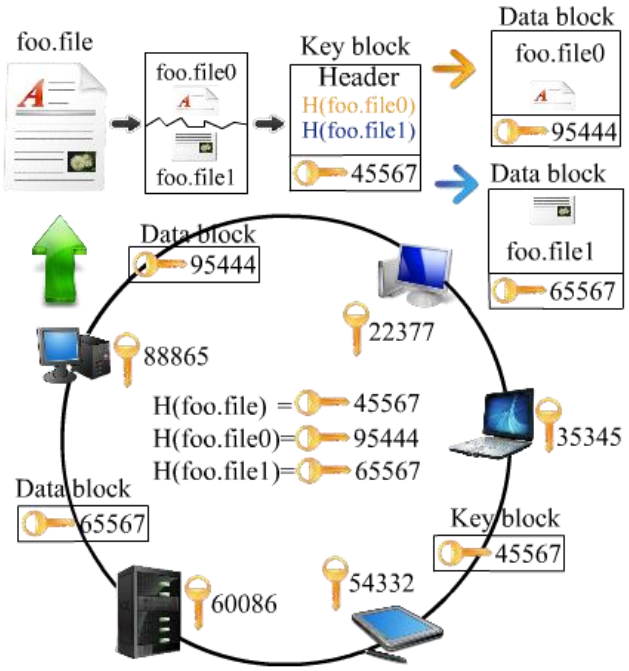

Figure 1 – Pastry Ring. Node with key 88865 inserts foo.file, foo.file is partitioned into data block that are referred by the keyblock. H(str) representes lookup(str). For example purposes 5 digit keys are used instead of 2^{128} Keys

2.2 Terminology and definitions:

Given a filename x, lookup(x) makes reference to the 128 bit unique number obtained by applying SHA-1[12] to x (also called key). A filename key is referred by lookup(filename). A sentence is a search query or a file name plus relevant file content information. The process for obtaining the keywords of a sentence is done by combining each word in the sentence, ordering the words in the combination alphabetically and upper-casing all the letters of each word as showed in table 3. For each keyword in a sentence a Meta-Data block is attached (showed in table 1). In specific the Meta-Data block (table 1) stores in entries the sentence that produced the keyword, the keys of the files associated with the keyword (this key is produced by lookup(sentence the relevancy of each of the file with respect to the keyword, number of downloads as well as other file related information. The operator name(Meta-Data) yields the keyword attached to the Meta-Data block.

2.2.1 Blocks

Blocks are the primary form of storage, all blocks are size n kilobytes (n is usually 512). Blocks are stored in the nodes whose key is numerically closer to their own key as showed in figure 1. A Data Block as showed in figure 1 is chuck of data, portion of a file (useless if found alone).
A Key Block (Inode, key indirections) as showed in figure 1, is crucial for reconstructing the file. An similar idea to that of [7][8][9] was followed. The key block contains a header describing the file, with information such as size, name, IP of inserter, time-stamp, size of

header, number of Data blocks and other related file information. Following the header, the keys of each Data Block for reconstructing the file are stored in sequential order. A Meta-Data Block as showed in table 1 is specific only for one keyword. The Meta-Data Block contains information about the data source from which the keyword was generated from, as well as information about the most relevant files associated to the particular keyword. The file information that is stored is the filename key, popularity of the file (number of downloads), percentage of relevance of the file to the keyword and other file related information. The Meta-Data block is referred as lookup(name(*Meta-Data*)). The meta-data block has a priority queue [3] design structure, for storing the relevant files associated with a keyword. In consequence every time a new entry arrives, it is ordered according to the relevance of the search thus making the results pre-ordered when a search is executed. A User Root block is specific only for one user, it contains information about the user's files. Each data entry holds the file name, size, modification date, IP, time, and the Pastry key referred to the file. A User Category block as showed in table 2 allows users to add categories to their files. A User Category block as showed in table 2 allows users to add categories to their files. The User Category block is specific only for one category. The User category Block stores the information related to all the files of the user that belong to that particular category. Each entry contains the file name, usage, size, date, IP, time and key referred by the file.

2.2.2 Messages

Given a 128bit number *key* and a node *sender*, insertMessage(*key*,*sender*) is a pastry message that is routed to the established *key* and that has been sent by *sender*, this message contains all necessary information so that when the node numerically closer to *key* received the message it may form a point to point (p2p) connection with *sender* and request the block referred by *key*. getMessage(*key*,*sender*) is a pastry message that is routed to the establish *key* and that has been sent by *sender*, this message contains all necessary information so that when the node numerically closer to *key* received the message it may form a p2p connection to *sender* and send the block requested referred by *key*. insertMetaDataMessage(*key*, *meta-data*, *source*) is a pastry message routed to the establish *key* and that has been sent by *source*, this message contains one *meta-data* entry so that when the node numerically closer to *key* received the message it will add the entry to the meta-data block presented in Table 1. insertUserCategoryMessage(*key*,*uce*,*source*) is a pastry message routed to the establish *key* and that has been sent by *source*, this message contains one *user's category entry* so that when the node responsible of the received the message it will add the entry to the corresponding user category block as showed in Table 2.

Table 1**:** Meta-Data block referred by 22345 obtained by lookup(BARFOO). Keyword BARFOO

Relevance	Popularity	size	Date	IP	File name
0.66	54	43543	14-12-64	x.x.x.x	Juan Foo bar.mp3
0.66	34	76755	23-53-23	x.x.x.x	foo hello bar.mp3
0.5	55	32453	23-56-79	x.x.x.x	Bar foo eat code.mp3
2/5 = 0.4	0	64675	12-34-87	x.x.x.x	Bar at night kills foo.mp3

Table 2: Category block referred by key 76298 obtained by lookup(name(user)+category)

Filename	Usage	Key	Insertion Date	IP	Size (kb)
File.txt	32	43243	14-12-99	x.x.x.x	45
Foo.c	15	98755	23-11-99	x.x.x.x	87, 323
Bar.h	5	12453	23-09-79	x.x.x.x	2,143
Im.doc	3	75343	12-04-87	x.x.x.x	43,212

3. Search on the Cloud set of Operations

3.1 Community Side

1. Insert a public file
2. Get a public file
3. Search for a public file

3.2 User Side

1. Insert a user (private) file
2. Get a user file
3. List user files
4. Add categories to user file
5. List files filtered by categories from user And sort them by Usage

3.3 Community Side Operations Definitions

3.1.1 Insert a Public File

When a node *source* seeks to insert a file *x* to the system, a new keys block for the file *x* is created. This keys block will be referred by lookup(*x*); The file *x* is then split into data blocks, each data block is referred by lookup(*x*+*part*) where *'part'* corresponds to the index of each data block. Each of the resulting keys is then stored in the key block of file *x*. For each data block a insertMessage(lookup(*x*+*part*),*source*) operations are simultaneously performed. Each of the resulting messages is routed through the pastry ring in consequence the node that is numerically closer to lookup(*x*+*part*) received the corresponding data block by forming a p2p connection with *source*. The last operation, is the insertion of the key block, which is inserted through

insertMessage(lookup(*x*),*source*). Figure 1 presents an overview of the steps involved in the insertion of a file to the system. For each *keyword* extracted from the file a Meta-Data entries is created and a insertMetaDataMessage(lookup(*keyword*),*md*,*source*) is routed to *key* and *md* represents in this case an entry in table 3.

3.1.2 Get a public file

When a node *source* seeks to retrieve a file *x* from the system, a new getMessage(lookup(*x*),*source*) is created and routed through the pastry ring. Then when the node responsible for the key block receives the message it forms a p2p connection with the *source* and sends the key block file. Afterwards *source* extracts from the key block the necessary keys to reconstruct the file. For each *key* a getMessage(*key*,*source*) is routed, requesting the corresponding data block referred by the *key* to the node that is numerically closer to *key*. In consequence the responsible node forms a p2p connection with *source*, sending the data block. After all the data blocks are received from the corresponding nodes, the data blocks are joined together following the order indicated in the key file.

3.1.3 Search a public file

When a node *source* wants to find a file *x* on the system by typing a sentence, a meta-data listener is started and a meta-data master will manage all of the meta-data blocks generated from the keywords of the sentence. For each *keyword* formed by the combination of the words in sentence, as showed on table 3 , a getMessage(lookup(*keyword*),*source*) is routed thus the node responsible for lookup(*keyword*) will form a p2p connection to send the meta-data block to source. When the meta-data block is received, meta_data_master will merge sort it with other previous meta-data blocks and show the new results. The complexity of the search depends upon *m* keywords and *n* metadata block entries per keyword making it O(*m***n*).

3.2.1 Insert a User File

When a *user* wants to insert a file *x* with *c* categories to the system, a new keys block for the file *x* is created and is referred by lookup(*user*+*x*). The file *x* is subsequently split into data blocks, each data block is referred by lookup(*user*+*x*+*part*). Each key is then stored in the key block. For each data block insertMessage(lookup(*user*+*x*+*part*),*source*) are simultaneously routed through the pastry ring. Lastly the key block is inserted through insertMessage(lookup(*user*+*x*),*source*).
A user category entry (*uce*) is created for file *x* and an insertUserCategoryMessage(lookup(*user*+*root*),*uce*,*user*) is routed, where root is the root directory (category) to where all the uce are inserted to, making available a list of all the files the *user* has inserted. Afterwards for each category in *c*, a user category entry is created for file *x* and an insertUserCategoryMessage(lookup(user+category),uce,user) is routed adding the uce to the corresponding user category block.

3.2.2 Get a User File

When a *user* wants to retrieve a file *x* from the system, a getMessage(lookup(*user*+*x*),*source*) is routed through the pastry ring. When the node responsible of the key block referred by lookup(*user*+*x*) receives the message it forms a p2p connection with *user* and sends the key block file. Afterwards the *user* will extract the information from the key block acquiring the keys needed to reconstruct the file. For each *key* it will route a getMessage(*key*,*source*). When all the data blocks are received from the responsible nodes, the data blocks are joined together following the order indicated in the key file.

Table 3: shows the meta-data entries that are formed when a node publish the song: Juan Foo bar.mp3

keyword	Key	Relevance	File-name
JUAN	lookup(JUAN) 99483	1/3 = 0.33	Juan Foo bar.mp3
FOO	43212	1/3 = 0.33	Juan Foo bar.mp3
BAR	67277	1/3 = 0.33	Juan Foo bar.mp3
JUANFOO	90523	1/3 = 0.33	Juan Foo bar.mp3
BARJUAN	32233	2/3 = 0.66	Juan Foo bar.mp3
BARFOO	22345	2/3 = 0.66	Juan Foo bar.mp3
BARFOOJUAN	86423	3/3 = 1.0	Juan Foo bar.mp3

3.2.4 Add a Category to file

When a *user* desires to add a category *c* to file *x*. a user category entry is created for file *x* and an insertUserCategoryMessage(lookup(*user*+*c*),*uce*,*user*) is routed thus adding the entry to the category block.

3.2.5 List files filtered by categories

When a *user* wants to filter his documents by categories, a master listener is created for the *user*. For each *c* in categories a getMessage(lookup(*user*+*c*),*user*) is routed thus the node responsible for lookup(*user*+*c*) will form a p2p connection to send the category block to *user*. When a category block is received, the master listener of the *user* will merge sort it with other previous category blocks and show the new results. The complexity of the list depends upon m categories and *n* user category entries per category block, thus O(*m***n*)

4. Replication

4.1 Inserting Replicas

When a block is inserted, Pastry routes the insertMessage to the node that is numerically the closest to the block key. This Node then sends a direct insertMessage to the k nodes from its leafset, informing the nodes to request the block that has been inserted. Each of these nodes then forms a p2p connection with the sender of the insertMessage and stores a copy of the block. The replication factor k depends on the availability and persistence requirements of the block and may vary between blocks. A lookup request for a block is routed towards the live node with a key that is numerically closest to the requested block key. This procedure ensures that a file remains available as long as one of the k nodes that stores the file is alive and reachable via the Internet; with high probability, the set of nodes that store the file are diverse in geographic location, administration, ownership, network connectivity, rule of law, etc.; and the number of blocks assigned to each node is roughly balanced.

4.2 Updating Replicas

All nodes have a log with the blocks that they manage, with information about where the block's physical location within a node is, the block time stamp (last time updated), and the key. When a node joins the ring, it requests the log from each of the nodes in its leafset. Afterwards it searches for any updates (comparing the timestamp from its log and the other logs from each node of its leafset), if any updates are present (a node from the leafset has a new version of a block), the node will request the newer version of the block thus updating its own log and blocks.

5. Updating files

Search on the Cloud is a read only file system for public files, this is due to the issue that anyone could update a public file and destroy information from that file. If there is a collision with the key (the file already existed), a new key will be calculated using information of the filename and the file size. If there is still a collision, the node will be informed that the file is already in the system and be requested to modify the filename. User files may be updated, only the latest version will be kept in the system. To update a block, an insertMessage is routed through the ring to the blocks key. When the node numerically closer to the key receives the message, it request the updated block to sender and updates the block and its log. Afterward the node sends a direct message to each node in its leafset informing about the newer version of the block thus each node in the leafset requests to the sender the updated block. In consequence each node in the leafset updates the block and changes are reflected in nodes log.

6. Encoding

A similar encoding to that found in PAST is used, using the same premise that storing *k* complete copies of a block is not the most storage efficient method to achieve high availability. They use ReedSolomon encoding. Search on the Cloud add *m* additional checksum blocks to *n* original data blocks allowing recovery from up to *m* losses of data or checksum blocks [24][25][26].
This reduces the storage overhead required to tolerate *m* failures from *m* to (*m + n)/n* times the block size (512 kb). The storage overhead for availability is very small thanks to multiple blocks that are created per file. Independent of the encoding, also improving bandwidth. However, these potential benefits must be weighed against the cost (in terms of latency, aggregate query and network load, and availability) of contacting several nodes to retrieve multiple blocks.

7. Public Key Generation and File search.

Search on the cloud considers that the average user of a file sharing system is not necessarily aware of the existence of a file's public key, but can provide relevant content information related with the file they are searching for. Our system assumes that when a user seeks to retrieve a file, he or she will provide words related with the name or content of the file they are searching for. Given these words, our algorithm searches the node space and retrieves the metadata entries that are the most relevant. The metadata entries retrieval is done in the following form: First for each new file that is added to the system, a series of keywords are generated, these keywords are formed from the file's name and from the file's content. If the file is a text file, the log frequency weight[28] of each word in the document is calculated, the k words with the largest log frequency weight are then selected as representatives of the file's content. The log frequency weight of a word w in a document d can be defined as a function f(w):

$$f(w) = \begin{cases} 1 + \log\ wf_{w,d} \\ 0 \end{cases}$$

Where $wf_{w,d}$ represents the number of times the word w occurs in the document. If the file is of a movie type, then the words taken as representative of the file are obtained from the information imdb[27] provides about the movie name. The data that is recollected are the name of the actors participating in the film, the director name, the movie genre, and the non-stop words in the film plot. The imdb API[27] is utilized for recollecting this data. If the system does not find any movie entry for the file name the user provided, the system asks the user if the file name they have selected is adequate for the film. Furthermore, if the system finds many different movies that match the file name the user provided, the system displays the

different film titles along with their plot information and requests the user to select the film that matches the one they are currently uploading, in this form ambiguity problems are avoided. If the file is of any other type, then the words taken as representative of the file content, are the words in the file's meta-data. With the filename and the words representative of the file content, a series of keywords are generated. Our system integrates the file content information in the creation of keywords, because the more file information that is provided, the less ambiguity that exist when the file search is performed and the better the retrieved results are. The keywords are created through combinations of the words in the file name and relevant file content data. With lookup(keyword) the public key of each keyword is computed. As mentioned previously, each keyword has an associated Meta-Data block, that holds information about the files whose name and content generated the keyword. Therefore each time a file generates a certain keyword, an entry to the keyword's Meta-Data block is added with relevant file information. The keyword's public key allows the retrieval of the file's meta-data block to which the file information is added. The data stored in the Meta-data entry is the file name and file data content that produced the keyword, the file's public key, the relevancy of the file with respect to the keyword, as well as other file related information. The relevancy metric of a file with respect to a keyword, is based on the number of downloads the file has, as well as the frequency each keyword terms presents in the file's content and name. When a User inputs a query to search for a file, from the query, keywords are automatically created. The system then searches for the keywords in the network. For each keyword, the system retrieves from their meta-data block the top K highest ranked files. (For visualization purposes in our study K=three). The top K files from all the analyzed keywords are then ordered with respect to their number of downloads, and that is what is finally returned and presented to the user. In essence our relevant metric benefits files that many users have considered useful and have downloaded, and files that present content relevant to the user's search intent.

8. Usability Inspection

In this section, we inspect the usability of Search on the Cloud by utilizing a cognitive walkthrough methodology. The cognitive walkthrough is a practical evaluation method, in which the user examines the interface and with the system intends to complete a series of assigned tasks. The cognitive walkthrough helps identify the ease of learning, use and usability of an application.

8.1 Users

The usability inspection of Search on the Cloud, was done by 15 different users. Only two of the participants had never used an online file sharing system, ten of the users had utilized file sharing systems similar to PirateBay, Only one of the participants had utilized a DHT distributed file sharing system before, the system used was POND.

8.2 Tasks

The tasks assigned to the users were: search for 6 different files on the system and incorporate 6 files to the system. In the searching for files task, a description about the content of each file was provided. The users had to create a query for finding the file in the system. The following is an example of a file description provided to the users: "*Comedy film where Linsay Lohan and Tina Fey appeared. Lohan played new girl, Cady Heron, that tries to click with various high school groups".* In the file incorporation task the user was also provided a description of each file. The user had to manually name the file they were adding to the system. If the system detected that the file name did not necessarily match the file content or if the file was a movie file and the system could not find the movie name in the IMDB database, the system suggested a naming. Additionally the system presented to the user the keywords that were generated from the file's content, these keywords could be modified by the user.

8.3 Results

From the list of six files the users were asked to find on the system, fourteen of the user were able to find all the specified files, at times the file they were searching for did not appear as the top result, but was within the list of files returned by the system. Only one of the users had trouble finding a Mexican film, the reason was that they provided the video's original Spanish name, yet in the system, only the English version with its associated English tags had been uploaded. Therefore the system could not retrieve any results. To overcome this problem, we have thought of integrating the movie's original title name, which is a field in IMDB. Other interesting things observed, was that users tended to use proper names in their search query. This made us believe that it might be best to generate keywords from proper names and not from verbs. Additionally we found that users tended to incorporate to the query cultural facts about the actors in the film. We therefore believe that integrating human knowledge to our search system could provide better results. All of the users expressed satisfaction with the system's speed and robustness in retrieving the files they searched for. They also enjoyed the display of the top most relevant files from the search.

In the file uploading tasks, in the case of the word files, ten of the users in two of the word files decided to change the keywords and input their own. These persons expressed, that although the automatically keywords were related to the file in question; they did not necessarily believe they were the most relevant. This gave us insight that a more thorough analysis of finding relevant keywords in word files is needed.

Overall the users expressed comments of satisfaction about the system.

9. Conclusions

In this work we presented the design of a more complete distributed file sharing system that offers the user an intuitive interaction. Our system, named Search on the Cloud, is fault tolerant and presents high availability, scalability, minimal costs of operation and deployment.
Our search procedure reads in a series of keywords automatically generated from the search sentence query. The keywords are then handled to retrieve a sorted set of files using a relevant metric. Our system allows the user to search for files based on relevant content and not on a public key, which is not representative of the file. Our system also leverages user input, by mining the file and generating the file's keywords automatically. Additionally, for usability purposes files in our system are categorized: Files can potentially belong to several categories/folders. These categories allow the user to query for files that could be associated with different categories, increasing therefore, the chances of producing a set with relevant matches in a reliable matter.
The novelty of our study is that we offer the user an intuitive search modality, while still presenting an entirely distributed approach.
Despite its preliminary character the research reported here indicates it is possible to design and construct a completely distributed file sharing system that does not present to novice user's foreign search interfaces.

References

[1] J Pouwelse, P Garbacki, D Epema, The bittorrent p2p file-sharing system: Measurements and analysis, Peer-to-Peer Systems IV, 2005 - Springer

[2] K Tutschku, A measurement-based traffic profile of the eDonkey filesharing service, Passive and Active Network Measurement, 2004 - Springer

[3] P van Emde Boas, R Kaas, Design and implementation of an efficient priority queue, Theory of Computing Systems, 1976 - Springer

[4] A Rowstron, Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems, Middleware 2001, 2001 - Springer

[5] M Castro, Practical Byzantine fault tolerance, Operating Systems Review, 1998 - usenix.org

[6] S Rhea, P Eaton, D Geels, Pond: the OceanStore prototype, Proceedings of the 2nd …, 2003 – usenix.org

[7] SJ Mullender, Immediate files, Software: Practice and 1984 - Wiley Online Library

[8] Sidebotham, Vohunes: The Andrew File System Data Structuring Primitive, Proceedings of EUGG Autumn, 1986 – reports-archive.adm.cs.cmu.edu

[9] MK McKusick, WN Joy, SJ Leffle, A fast file system for UNIX, ACM Transactions on 1984 – portal.acm.org

[10]Free Pastry Tutorials and documentation: https://trac.freepastry.org/wiki/, 5 September 2011

[13] E Adar, Free riding on gnutella, First Monday, 2000 Citeseer

[14] P Druschel, PAST: A large-scale, persistent peer-to-peer storage utility

[15] I Stoica, R Morris, D Karger, Chord: A scalable peer-to-peer lookup service for internet applications, ACM SIGCOMM,2001 - portal.acm.org

[16] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S. A scalable content-addressable network. In Proc. ACM SIGCOMM (San Diego, CA, August 2001),

[17] BY Zhao, J Kubiatowicz, Tapestry: An infrastructure for fault-tolerant wide-area location and routing Computer, 2001 – Citeseer

[18] S Saroiu, KP Gummadi, Measuring and analyzing the characteristics of Napster and Gnutella hosts, Multimedia systems, 2003

[19] Balakrishnan, MF Kaashoek, D Karger, Looking up data in P2P systems of the ACM, 2003 - portal.acm.org

[20] Sean Rhea, Patrick Eaton, Dennis Geels, Hakim Weatherspoon, Ben Zhao, and John Kubiatowicz, Pond: the OceanStore Prototype,. Appears in Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03), March 2003

[21] F Dabek, MF Kaashoek, D Karger, Wide-area cooperative storage with CFS, ACM SIGOPS, 2001 - portal.acm.org

[22] E Berlekamp, Bit-serial reed-solomon encoders, Information Theory, IEEE Transactions on, 1982 ieeexplore.ieee.org

[23] http://www.ipoque.com/sites/default/files/mediafiles/documents/internet-study-2008-2009.pdf, 5 September 2011

[24] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. Software — Practice and Experience, 27(9):995–1012, Sept. 1997

[25] J.S. Plank, Note: Correction to the 1997 tutorial on Reed–Solomon coding Plank - Software: Practice and Experience, 2005 - Wiley Online Library

[26] J.S. Plank, The RAID-6 liberation codes - Proceedings of the 6th USENIX Conference on File, 2008

[27] http://www.imdbapi.com/ 20 September 20, 2011

[28] Hanna M. Wallach, Topic modeling: beyond bag-of-words. In Proceedings of the 23rd international, 2006. conference on Machine learning (ICML '06). ACM, New York, NY, USA,