Keyword Search for Freenet

Likuo(Brian) Lin, David Wentzlaff, Alexander Yip
{brianlin, wentzlaf, yipal}@mit.edu

December 12, 2000

Abstract

While Freenet has laudable design goals of being an anonymous, distributed, file distribution network, these goals stand as a direct obstacle to the efficient searching of Freenet. Currently Freenet does not even have true search functionality let an alone an efficient mechanism to carry out searching. In this paper we document the obstacles to implementing a searching system for Freenet and propose several solutions to this problem. We implemented our solutions and used performance metrics to compare them.

1 Introduction

Freenet is a distributed, anonymous, information storage system[1]. It is designed such that no one can tell who inserted files or who is reading those files. In addition, it is completely distributed and decentralized; all nodes are completely equal. No one node has authority over another, and there is no centralized control[2]. These are all desirable features for a system made to combat information restriction, but unfortunately they also make building an indexed search system for Freenet rather difficult. There are existing solutions for searching Freenet but we find them inadequate, and conflict with the original spirit of Freenet[3].

We have developed several systems that enable keyword searching in Freenet, all of which are aligned with the original goals of Freenet. They are ananymous, decentralized, redundant and scalable, and require little or no change to the existing Freenet architecture.

Our first scheme which we call the Indirect File method was first described in Clarke's paper[1]. It maps a given keyword to a set of indirect files. Each of those indirect files point to the actual file that matches the keyword. The indirect files are named as a function of their keyword, so given a keyword, one would know the names of the indirect files.

Our second scheme which we call the Summary method builds on the first, but instead of using a single indirect file for each matching document, the indirect files could be aggregated together. These larger files contain many pointers to files matching a given keyword.

These schemes depend on the ability to insert multiple files under a single file name. This feature is not currently supported by Freenet, so we have implemented a method that circumvents this requirement which we will call the Base Enumeration method and designed another method but have not implemented which we call the Lightweight Indirect File (LIF) method.

In the rest of this paper we describe the problem in more detail, discuss past related work, and outline our goals. Afterwards, we propose our solutions, describe their benefits and drawbacks, and show our preliminary test results.

2 Background

The problem of searching through distributed information networks has been solved in the past, but the properties of Freenet make it both a new and interesting problem. Existing search mechanisms cannot function in Freenet because of its widespread use of anonymity and encryption.

The simplest approach to the search problem is to create a centralized clearing house for keyword listings. All files would be listed there, including their keywords and locations. This would allow queries to be answered directly by the central service. This architecture provides fast searches and accurate results. Unfortunately, there is little sense of centrality in Freenet; all nodes are essentially equal. In addition, it adds a single point of failure; it is possible to attack the central service and prevent queries from being answered. Since Freenet is designed to be decentralized, we would like the searching system to also be decentralized, ruling out the centralized

search archtitecture.

Another simple solution would be for each node to broadcast a query to neighboring nodes for the keyword being searched. In response, each node would either forward the request to its neighbors, reply with matching documents, or reply with a failure message. Unfortunately, Freenet cannot use this solution because each individual Freenet node has no idea what information it is storing as all the data is encrypted on disk. Only the requestor is able to decrypt the data that he is requesting. In addition, the requestor cannot find out which node a retrieved file was stored on. This means that the individual Freenet nodes cannot answer queries about what files they are sharing.

In addition, the names of the files are not the actual names of files. Instead Freenet references files through a one way hash of the plaintext names of files. This means that searches cannot be performed on the names of files.

Regardless of these constraints, it is useful to have a keyword searching system for Freenet. Information could be found readily by searching, rather than by exchanging keys, or through word of mouth. At the same time, we would like to uphold the ideals used when designing Freenet itself.

The current solutions to this seaching problem are inadequete, or rather are not real search mechanisms. They are describled later in the related work section.

3 Related Work

3.1 Searchability of Existing Peer to Peer Networks

Recently many attempts have been made to create distributed file systems. Among the more famous ones are Napster, Gnutella, and Freenet. But there are many more including CuteMX, File Rogue, Filetopia, Freebase, KaZaA, Mojo Nation, Ohaha!, Riffshare, Scour, SongSpy, and Swapoo just to name a few. Each one has a different flavor, with various strengths and weaknesses. Some are completely decentralized, some have different types of nodes, and some are more secure than others.

Also, recently there have been a crop of large distributed filesystems and persistent data store technologies such as OceanStore[4] and Freehaven[5] which use many of the same techniques as Freenet to provide thier service. OceanStore, for example uses a system where each file can be replicated throughout the network allowing each server to have a copy of a document. Ocean store also mentions the ability to search through an encrypted document for an encrypted string without ever decrypting the data[6].

Searching systems have been devised for distributed file storage mediums such as Napster and Gnutella. Napster uses the centralized indexing strategy. Search queries are sent to the indexing servers where keywords are matched against files shared by individual nodes. This architecture is very simple and performs well, but it isolates a single point of failure, namely the Napster[7] indexing servers. Their vulnerability has been shown by the recent legal action taken against Napster's network[8]. Gnutella[9] does without a central location to store index files; it uses the broadcast query technique. This system depends on each node knowing the contents stored on it.

3.2 Existing Solutions for Freenet

One proposed solution to Freenet searching is outlined in Ian Clarke's Freenet Paper[1]; it describes a system that uses Lightweight Indirect Files that are allowed to have key conflicts. These LIF files would be named according to keywords, and they would contain CHKs pointing to files relating to the keywords in the LIFs' tags. This system depends on Freenet support for LIFs, but those are not yet supported.

The existing solutions for searching Freenet are websites devoted to key listings[3]; these are websites that list Freenet keys. People who insert files into Freenet can add their keys to these lists for the public to see and search through. We find this solution inconsistent with the goals of Freenet because it relies on a central repository, which is vulnerable to attack, and is not anonymous.

4 Design Goals

Our goals in designing a search system are as follows:

- *Anonymity:* Since the designers of Freenet took so much care to enforce anonymity for both publishers and readers in Freenet, we would like to maintain this property in any searching system we develop for it.
- *Decentralization:* Another one of Freenet's goals was decentralization. We would like to avoid any kind of centralized structure in our searching systems.
- *Scalable:* We would like the searching system to scale with Freenet.
- *Efficient:* Obviously, we would like the search system to be efficient in terms of bandwidth

used, time required for insert and search operations, and the number of messages passed for each operation.

5 Solutions and Design

5.1 Name Collision Problem

5.1.1 Enumeration Method

The enumeration method is a simple way to simulate the ability to insert multiple files under the same name without making major changes to Freenet. Changes only have to be made to the Freenet Client. The main idea of this approach is to append numbers to the end of a *filename*. To insert a file under a certain *filename*, one would insert the file under a name with the *filename* and a number that has not been used yet for this *filename*. To request a file with this *filename*, one would request the file under a name that contained the *filename* with a number that exists on the file system.

To insert a file one could simply enumerate through all the numbers starting from 0 until one doesn't get a collision. For example, suppose we wanted to insert a file under the filename *freenet*. We would first try *freenet#0*, then, if there was a collision we would try *freenet#1* and so on until we get a miss. Upon a miss, we insert the file with the name we missed on.

To request a file or multiple files one could get the files either by starting from 0 and count up to the number of files desired or by enumerating from 0 until the first miss and then enumerating down from the miss. The first method gets the oldest files and is not as desirable. The second method gets the newest files and is more useful although more costly. From now on we will assume that we always want the newest files.

There are three major drawbacks to this method. One drawback is that to insert a file and to request the newest file, one would need to find the highest numbered name for that files. This makes getting a file very slow. This is essentially a linear search that takes $O(n)$ time where n is the highest number for that file. Another major problem is that if files are purged from the system or if one of the lower sequenced files becomes unavailible, one will not be able to correctly search for the highest numbered file. He end his search after querying for the unavailible file.

Binary Search Optimization In an attempt to speed up search, one could do an exponential search

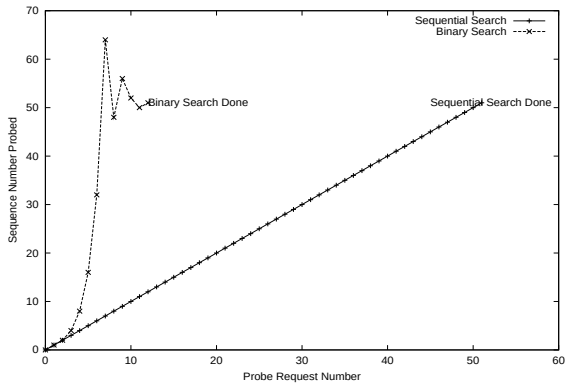

Figure 1: Comparison of Sequential and Binary Probing through 50 existing files

followed by a binary search. We first try numbers exponentially (i.e. $0, 1, 2, 4, 8, 16, \ldots$) until we get a miss. Then, supposing we miss on k, we then do a binary search from $\frac{k}{2}$ to k to find the highest number. A miss would denote that the highest number should be lower than the current number being tested, and a hit would denote the opposite.

5.1.2 LIF Method

Lightweight Indirect Files are another way to solve the name collision problem. This method is much more complicated and requires major changes to the underlying architecture for Freenet. Both the Server and the Client need to be modified. However, with this new power we are able to eliminate the inefficiencies of the enumeration method. Namely, we can simply change the system so that we can insert multiple files under the same name.

To insert a file, one would not check for any collisions. A server upon receiving this request would store multiple files that are referred to by the same filename and possibly pass on this request so that the file may be duplicated.

To request a file, one would send a request to a sever with a filename and the number of desired files. Server A upon receiving this request would get all the files that correspond to this filename up to the number requested. If the request can be satisfied, server A sends the files back to the node that made the request. If server A is unable to satisfy this request, it forwards the request along with a list of CHK's for the files that it has found so far. The next server B would try to fulfill this request while making sure that there are no duplicates. If the request can be satsfied, server B sends the files that it has found (without duplicates) to server A. If the request can

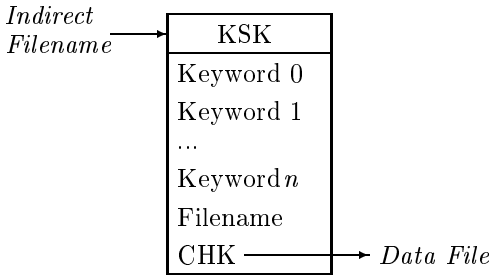

Figure 2: Indirect File Structure

not be satsfied, then server B also forwards the request with the list of CHK's for files that represents the combination of unique files that both server A and B collectively have. This goes one until the request is satsfied or when hops to live is zero. Eventually all the files found should return to the node that requested these files.

The main advantage of LIFs is that one does not need to search for the highest numbered file, thus inserts and requests can happen immediately without any kind of search. Of course one may have to go to many nodes to get multiple files, but this too will be much faster since one can essentially request multiple files at once instead of iterating through the numbers. An added bonus of this is that the time to complete a certain search is not adversely affected by the number of files on the system. The performance may actually improve if there are more files on the system since they are now easier to find. In the Enumeration case, the more files are on the sysetem, the longer it takes to search for the highest numbered file. One side effect of this though is that only the closest files are returned and not the newest keys.

5.2 Search Problem

5.2.1 Multiple Indirect Files

One way to implement search on Freenet is to insert indirect files under the keywords that one wants to be associated with the data file. This requires a user to supply a list of keywords for a file to be inserted under upon a file insert. Then, a series of keyword files will be inserted under those given keywords, each of which point to the file containing the actual data. Then a search would be done by getting for those keyword files for the keywords that one wants to search under, each of which will then point to a file that contains data related to those keywords.

For example, if we wanted *freenet_paper* to be inserted under *6.899* and *freenet*, we would insert two indirect files. One indirect file would have the name *6.899*, the other would have the name *freenet* . Both would contain the name of the data file, a pointer to the data file, and a list of the keywords to be associated with this file. The advantage of putting the other keywords in the indirect file is that to do an AND search one would only have to get files for one keyword, and then do the operation locally. By using these indirect files one would be able to find the Freenet paper only by knowing any of the keywords may be associated with it.

Ofcourse we would want to associate multiple files with the same keyword. That is where we use the ability to insert multiple files under the same name. Suppose that we also wanted to insert the Freenet presentation under the name *freenet_presentation*. We would then be able to insert more indirect files with the same names of *6.899* and *freenet* . Thus, when one seaches for files under the keyword *freenet* or *6.899*, one would find pointers to two data files.

The main problem with this is that there are too many files to handle and thus the system does not scale very well. There is anothre major problem. Files stay in Freenet as long as they are searched for. And these indirect files are constantly being searched for, while the actual file containing the data may not, and may be eventually purged. This leads to a lot of broken links that may never go away.

5.2.2 Summary Method

To improve on having just indirect files, we propose the summary method. The basic idea behind this method is to have all the inserted indirect keyword files be dated and to summarize them by keyword and date into summary files unless it is today. The reason is that more indirect files can be added for today, so we don't want to create the summary for today until the day is over. The summary will contain *entries* which will be the contents of the indirect files that this summary is summarizing. The summary method attempts provide solutions to three major problems. The first is that it reduces the number of files that need to be retrieved, thus greatly reducing the search time. Second, it allows a notion of absolute time, and is especially useful for LIFs which have no ability to differentiate newer files from older ones. Lastly, it allows pointers to purged files or unused files to be eventually purged also.

To insert an indirect file, instead of just inserting on the keyword, one would also attach a date to it. For example, instead of *freenet* we would use *freenet_12/09/2000*. An indirect file will be inserted whenever someone succesfully inserts a new file or when a file from a previous day has been succesfully

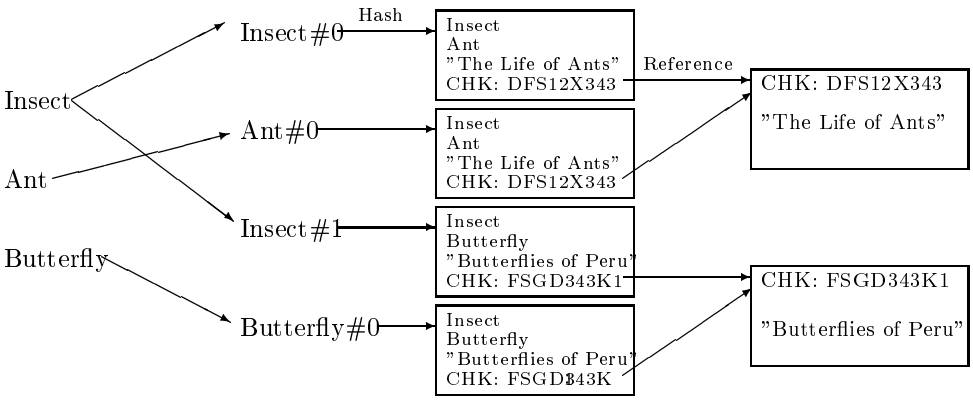

Figure 3: Keyword to Indirect Files

retrieved. The effect of this is that the indirect files for any day will only contain files that we know to be on the system on that day. This will be a good property if we want to make sure links to old files will be eventually purged.

To retreive a file is more complicated (See figure). The user must request a list of keywords, a date range, and the number of results desired. The keywords are assumed to be for an AND search. The algorithm runs in a loop until all the indirect file data for those keywords and that date range has been obtained, either by getting summary files or by getting the indirect files themselves. It starts from today and goes back in time. For today, it simply gets indirect files up to the number requested. As long as we have not gotten more results than desired or passed the date range we continue to run this loop. For each date, search for the summary file for that day. If it exists, get it and continue onto the next date. If it does not exist, pool all of the indirect files for that day and put it into a summary file and insert it into Freenet.

Despite it's obvious advantages, there are also some drawbacks. The most threatening is that people can make bogus summaries. Since there is no idea of one source being more trusted than another. Anyone can insert a summary for a keyword and date. One partial solution to this is to allow multiple summaries to be inserted under the same keyword and date. And one simply gets all of them and discard the ones that are garbage if they can tell them apart. If it is garbage they may want to pool all the keywords for that day and insert a correct summary. Although, this seems like a major problem, it is not entirely exclusive to the summary files. All indirect files can have wrong entries or garbage in them and that is just a property of Freenet and cannot really be solved. Another performance issue is that since these summaries may be big, it would be a waste of bandwidth if someone only wanted two entries for a certain day but was forced to get the whole summary file. However, since time for file retrival is small compared to that of searching for a file, the performance improvement of summaries is well worth than the occasionally wasted bandwidth.

Diff files Since we reinsert indirect files when they are succesfully retrieved, it is very likely that many of the entries in the summary for one day will be the some as the entries for the next day. One solution to this is to have two type of summary files for each day, *Base* files and *Diff* files. The *Diff* file would contain the entries that are in today's indirect files but not in yesterday's indirect files. The *Base* file would then be the entries that are not in the diff file for that day. Then to get all the entries for a certain date range, only the *Base* file of the first day and the diff files of the other days are needed. If the files used from day to day are mostly the same, then we will be able to save a lot of space. Of course the combined *Diff* could also have duplicates. If this become a serious problem, one could make *Diff* files that span several days.

6 Results

To study the performance characteristics of the above described searching methods, the base enumeration method with binary searhing optimizations and the summary searching method were implemented. These were both implemented in Java and extend the

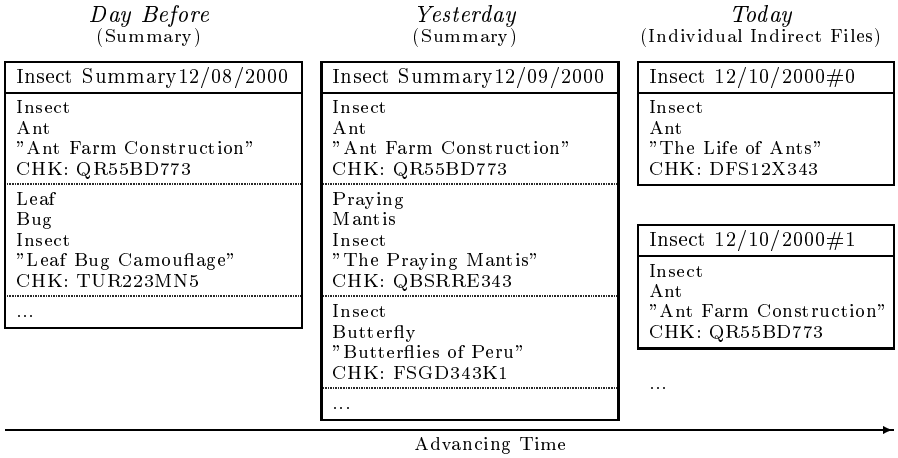

Figure 4: Structure of Summary Files

main Freenet implementation. The LIF method of overcomming the name collision problem was not implemented due to time constraints and difficulty since it required massive changes to the Freenet server architecture. It is hoped that the implimented searching system will be integrated into the main Freenet source and become widely used as the prefered searching method of Freenet.

6.1 Test Methodology

All of the tests we ran were done with real Freenet implimentations. We hope that becasue of this, our results are more valid than if we simply used a simmulator. Our testing infrastructure was a combination of instumentation of Freenet servers and Perl scripts that automated the tests. The instumentation that was added to the server was the logging to files of number and size of each message passed between servers and servers and clients in the Freenet network. Also the times used for testing are real wall clock times captured by live runs of the tests and thus can have transient inconsistencies due to operating system overhead and Java overhead such as Java's built in garbage collection. We feel that these inconsistancies are minimal becasue the machines that these tests were being run on were only being used for the purpose of these tests.

Our tests were run on several dual-processor x86 computers with a gigabyte of RAM unsing Sun's Java JDK 1.3 for Linux. Because of the use of wall clock time, comparative trials were run on the same computers under the same load. For more information on usage of the designed tools and a pointer to the actual code see Appendix A.

In this paper we only display graphs for time, which we feel to be the most useful metric because that is what people who are searching really care about. In addition the other metrics that we logged all followed the same trends as the time graphs do.

6.2 Enumeration and Enumeration with Binary Search Comparison

To prove that the Enumeration method works as a useful searching tool, we ran simulations of searches on our Freenet test network. We were able to use the Enumeration method to reliably insert multiple files with the same keywords into the testbed and retrieve the results of querys for keywords.

Next we compared the basic Enumeration method to the Enumeration method with the binary search optimization. To do this, for each run, we used 50 virgin Freenet servers with randomly generated connections between the nodes. All of the servers and the test clients were running on the same computer with the different runs not being run at the same time so that they all would have a control environment. The 50 servers were initially connected together with 40 random connections per node to approximate a well connected Freenet topology.

We varied the number of total indirect files matching a particular keyword in Freenet and logged the time that it took to do a request for one file matching a particular keyword. We did not actually retrive the file that a particular keyword pointed to but rather just the indirect file. This was done for three different modes. The Sequential mode, Binary Search Optimization mode, and the Baseline which is simply an oracle that knows what the highest numbered file to retrieve is and is just shown to profile the time that it takes to request an indirect file.

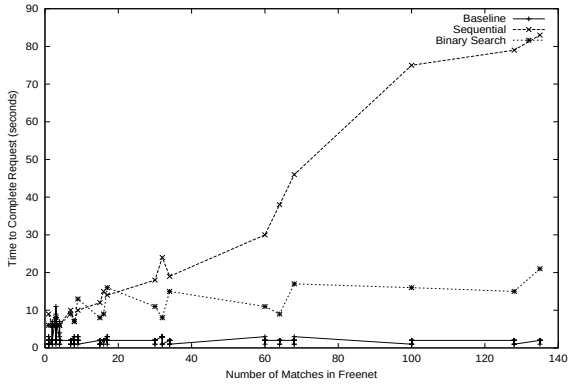

Figure 5: Number of Files with Same Keyword Comparison

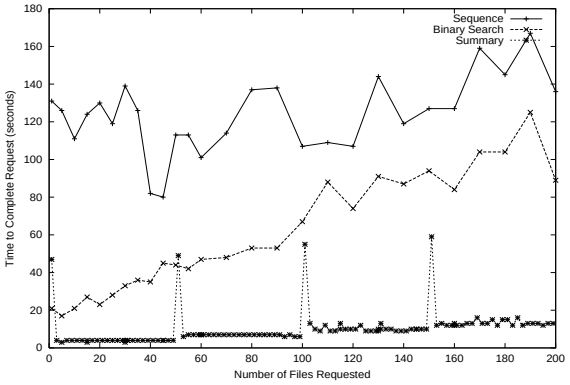

Figure 6: Number of Keyword Results Requested Comparison

As can be seen from Figure 5 our intuiton was correct. The time it takes to search for the Sequential Enumeration grows linerarly with the number of matching indirect files in Freenet. Also the Binary Search Optimization significantly reduces the required time to do a seach and grows roughly as the log_2 of the number of matching indirect files in Freenet. Lastly the Baseline plot shows that the average time it takes to request an indirect file with no need to find the highest file number is approximately two seconds.

6.3 Enumeration and Summary Search Comparison

To compare the Enumeration methods versus the Summary Search we tested them with a fixed number, 200, of matching indirect files in Freenet. Also we distributed the files uniformly across four days all of which were not "today" so that the Summary method would always summarize a day on the first request to the server. The tests used 20 servers with 15 initial random connections between servers. We would liked to have run more tests with a larger number of servers, but unfortunately, due to the large amount of memory that this test required, we were limited.

We did separate controlled requests against these servers for the Sequence seaching, the Binary Search and the Summary Search, varying the number of files requested from 1 to 200. Note that this is in contrast to our earlier results where we were always requesting only one file, but instead varied the number of files for a single keyword in Freenet. For the Summary trial, the algorithm works in such a way that it gathers the newest documents for a given keyword and date range. Thus as more files are requested, the algorithm starts to ask for older filesk and has to build summaries when the summary for that day has not been made yet. In this test the number of files for a given keyword was static at 200 across all trials.

The results from this test show some interesting results as can be seen in Figure 6. As expected the Binary Search beats the Sequence search due to less requests requests to find the top value of the inserted keyword. But surprisingly we still see that the Sequence and Binary Search now grow linearly with the number of files being retrieved. The reason for this is that even though there is a constant time across trials to find the highest numbered keyword, the searches still have to retrieve the number of requested files. This grows linearly with time.

The Summary method gives quite an improvment over the Enumeration based methods because once one person does the work of generating the summary file for a given day, subsequent requests are able to use that summary eliminating the need to get many files. As can be seen in Figure 6, the four peaks in for the Summary runs correspond to the four times, once for each day, that the user had to request all of the files for that day. Since the number of files to retrieve has a much larger impact on time than the size of the file, one can see that the summary method has superior performance.

??

7 Future Work

In the future we hope to continue with this project and work with the main developers of Freenet to integrate our ideas and code into the main Freenet source tree.

While we have not had time to impliment LIFs for this paper, we believe that it would be a worth-

wile project to undertake as it would result in another choice in solving the name collision problem in Freenet. Also the performance gain by using LIFs would help in the adoption of Freenet as a useful information publishing system.

Other ideas that we didn't extensively explore but had merit were the caching of intersection searches and use of a centrally administered indexing system. The idea behind caching of intersection searches is that when one person does a search such as a search for foo AND bar, they insert some type of index into Freenet such that people in the future who want to search for the intersection of keywords just search for thier summary file so that subsequent searches don't have to do multiple searches with merging. This summary file would be inserted under some convention such as being indexed by the first keyword in lexicographic order. A centrally administered searching system is another way to speed up searching. The idea is to use some type of summarized system, such as the one described in this paper, or a crawler of Freenet webpages and have a trusted centrally administered entity like Yahoo every night rebuild a large index of keywords in Freenet. Unfortunately this has all of the problems with current indexing systems such as accountability but it may prevail becasue it would significantly reduce searching time.

8 Conclusion

In this paper, we have explored the problem of searching Freenet files by keyword. We analyzed the problem and designed several solutions for it. We went on to implement a number of them, and compared query performance experimentally. In the future, we plan to implement more of our proposed solutions and characterize their query performance.

9 Acknowledgements

We would like to thank Proffessor Hari Balakrishnan for showing much interest in this project and taking the time to have heated debates on the best way to implement portions of this project. We would also like to thank the MIT Computer Architecture Group for lending thier spare computer cycles on really fast computers such that we could quickly generate meaningful results for this project. Lastly we would like to acknowledge the Freenet project and all its contributers without which this project would not have existed.

References

[1] I. Clarke, O. Sanberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," in *Proceedings of The Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[2] I. Clarke, "A distributed decentralized information storage and retrieval system," Master's thesis, University of Edinburgh, 1999.

[3] B. Wiley, "Key index server listing," December 2000. http://uts.cc.utexas.edu/ blanu/keyindex.html.

[4] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: An architecture for global-scale persistent storage," in *Proceedings of ASPLOS-IX*, November 2000.

[5] R. R. Dingledine, "The free haven project: Design and deployment of an anonymous secure data haven," Master's thesis, Massachusetts Institute of Technology, 2000.

[6] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches encrypted data," in *Proceedings of Security and Privacy Symposium*, May 2000.

[7] "Napster," 2000. http://www.napster.com.

[8] J. Gallivan, "Napster shut down - federal judge rules against rouge music site," *The New York Post*, July 27, 2000.

[9] "Gnutella," 2000. http://gnutella.wego.com.

A User Interface and Usage

The source code along with extra documentation for this project can be found at `http://cag.lcs.mit.edu/~wentzlaf/classes/6.899/public/project/source`.

A.1 Enumeration Method and Binary Optimization

Currently to use these tools, there are simple command line interfaces that are derived from the default request and insertion clients included with Freenet. These all can take the default flags such that you

can use custom port numbers and change the logging verbocity, etc.

```
  Usage:  freenet_insert_keyword URL
[input-file] {[keyword_0] ...  [keyword_n]}
  Usage:  freenet_insert_keyword URL
[input-file] {[keyword_0] ...  [keyword_n]}
  Usage:  freenet_keyword_request KEYWORD
OUTPUT_FILE_PREFIX NUMBER_TO_RETURN
  Usage:  freenet_keyword_request_log
KEYWORD OUTPUT_FILE_PREFIX NUMBER_TO_RETURN
```

Summary Method `Currently this is how one inserts an indirect file for the summary method`

```
  Usage:  MainSearch -i [string of entry] =
[date]##[name]##[key]##[keyword]##[keyword]...
  Currently this is how one performs a
search
  Usage:  MainSearch -s [string
of query] = [number of files
wanted]##[startDayOfYear]_[startYear]##[endDayOfYear]_[endYear]##[keyword]##[keyword]...
```