UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

Computer Science Honours

Final Paper 2018

Title: A Portable Large Volume Email Retrieval System

Author: Shivaan Motilal

Project Abbreviation: FINDMAIL

Supervisor: Associate Professor Hussein Suleman

Category	**Min**	**Max**	**Chosen**
Requirement Analysis and Design	0	15	0
Theoretical Analysis	0	25	0
Experiment Design and Execution	0	20	20
System Development and Implementation	0	10	10
Results, Findings and Conclusion	0	20	20
Aim Formulation and Background Work	0	10	10
Quality of Paper Writing and Presentation	0	10	10
Adherence to Project Proposal and Quality of Deliverables	0	10	10
Overall General Project Evaluation [requires explicit motivation from the project supervisor]	0	0	0

A Portable Large Volume Email Retrieval System

Shivaan Motilal

University of Cape Town, South Africa

mtlshi005@myuct.ac.za

ABSTRACT

This paper presents the development of a portable system that allows you to search over mbox and maildir email archives. It specifically addresses the problem of locating and viewing past emails from archives, in the case of them significantly increasing in volume and being exported into large archives. The tool developed and discussed in this paper aims to help users to better organize and manage their email, giving them the convenience of locating past emails across multiple operating systems. The system parses and indexes the archives correctly but takes a long time to generate the inverted index for search. The index itself is created on the basis of simplicity as a means for portability, and thus plain XML files are used as indices.

CCS CONCEPTS

• **Information systems** → **Information retrieval** → Specialized information retrieval • **Software and its engineering** → Software libraries and repositories • **Software and its engineering** → Software Portability

KEYWORDS

Portable; Searchable; Email formats; Offline; Archives; Indexing; Parsing; User interface; Query System

1. INTRODUCTION

Email users who are busy with their day-to-day work often find themselves having to manage large volumes of incoming email as time goes by. Emails are either deleted or archived by the email user, depending on the importance of the email contents. If a user decides to archive their email, it usually contains information the user would like to access again in the future. Over time, these email archives can however become huge and what Whittaker et al. [17] termed as "email overload", can occur. This is a situation whereby manually retrieving emails from large archives becomes arduous and time-consuming. This can be caused by many factors, such as poor personal information management and having large volumes of high priority email.

Along with the problem of "email overload", there also exists the issue of archives becoming obsolete through *software aging* [9]. To address the issue of obsolescence, various preservation strategies have to be considered and then deemed appropriate to fit the context of the problem presented.

A portable offline searchable email archive that handles multiple email formats (such as mbox and maildir) was the proposed solution to the email overload and obsolescence issues mentioned. The search functionality addresses the problem of email overload by efficiently retrieving specific emails from a large email archive, while the portable and offline features allow for the archive to be less likely to become obsolete in the short-term. The parsing and indexing component of the proposed solution is detailed in this paper.

The whole project is divided into two logical sections, which, when used together, create the FINDMAIL system. The two sections are namely:

1. *Pre-Processing:*

   This is the process that includes parsing and indexing of the inputted archives of various email formats. Parsing consists of extracting and re-structuring relevant information from the inputted archive, while indexing is the process of creating indices from the parser output. This is the main focus of the rest of this paper.

2. *Email Processing:*

   This consists of the user interface and query system. The query system should ideally allow for fast and efficient retrieval of emails,

while the user interface should display emails clearly to the user and allow for ease-of-use. The query system should also be able to handle various queries, including single-word and phrase queries.

1.1. Project Significance

There are currently a few existing tools such as Windows Mbox Viewer [11], Mailpile [5] and Mairix [10] that allows users to view and search over their archive. These tools however are not platform independent and, if they are, they usually involve going online and having an Internet connection.

What we aim to provide is a tool that users can use offline to browse and search through their email archives, without having to worry about the operating system they are using. We would like this project to encourage and assist individuals in better managing of their emails.

We hope that the FINDMAIL system will allow users to better organize and handle their email, in the case of their email being stored in large archives. Ideally, we would like the user of the system to be able to parse and index archives of large sizes without having to be concerned about drastic decreases in performance. The main objectives of this project is to answer the following research questions:

- Can we create an indexing system that works for the popular email formats (mbox and maildir), as well as other relevant ones?
- Can both indexing and parsing work on multiple platforms (portable)?

1.2. Project Structure

In the sections to follow, this paper will present detailed information on the design and evaluation of the FINDMAIL system. Firstly, related work that influenced the design of the system is presented and thereafter the design and implementation of FINDMAIL's parsing and indexing components are shown. Secondly, the design of the experiments conducted, as well as the results obtained are illustrated. Finally, the ethical considerations, conclusions and future work are provided .

2. LITERATURE REVIEW

2.1. Digital Collections:

In South Africa and other developing countries, most preservation techniques cannot be implemented [15]. This is mainly due to insufficient resources or the high cost of Internet bandwidth. These countries often have to find alternative approaches that are more practical.

One such alternative approach of preserving digital collections (which are inclusive of email archives) is through applying the principle of simplicity when creating the system [10]. A specific way of doing this would be to use XML plain text documents to store information and metadata. This would make the information in the documents easier to retrieve after many years have passed. This approach of simplicity also enables seamless interconnection, extension and modification of the features of that specific system, thereby allowing for the system to operate on multiple operating systems. This can be seen to address the issue of technological obsolescence, which is relevant to email systems as email users often access their emails from archives that are in multiple formats and sometimes on different operating systems [1].

An example of a system that used XML plain text documents to store information is CALJAX. CALJAX was developed by Suleman et al. [16], to be a generic hybrid (online-offline) repository management and access system that utilized a strong AJAX foundation. The system allows for integration of content from a local source with content from a remote source, through the use of just a Web browser [16].

Expanding on the idea of hybrid offline and online systems, is the idea of having a hybrid online-offline digital collection (specifically an email archive) to address issues such as poor Internet bandwidth and digital preservation. Online and offline collections present both advantages and disadvantages, thus a hybrid digital collection(online-offline repository) could interleave advantages from both, and potentially aid in preservation [16]. The practicality of creating a hybrid

online-offline system is however an issue. Complications that can arise include:

1. *Inherited Security flaws*: ensuring that foreign-origin Web content included into the web app cannot gain access to local resources is important in hybrid systems. However, hybrid applications delegate security enforcement and this allows for flaws and vulnerabilities in security to be inherited by the application [6].
2. *Inconsistency across devices and operating systems*: A hybrid application should be able to run on multiple operating systems and therefore when interfacing with these different systems, lagging can occur. The appearance and functionality of the system can vary according to the type of operating system and device (platform used) [16].

2.2. Email Archives:

Windows Mbox Viewer(WMV) [14], Mairix [13] and Mailpile [5] are existing software projects that allow for display and/or searching of email archives. WMV allows for display of emails for archives in mbox format, but does not provide search functionality. It works offline and was designed specifically to work on Windows. Its benefits are thus that it works on the mbox email format and runs offline, and its drawbacks are that it does not accommodate email formats other than mbox and is not portable across multiple operating systems.

Mairix [13] and Mailpile [5] are quite similar, both include indexing and search functionality and both index mbox and maildir formats. However Mairix, additionally accepts the MH format email archive. Mairix [13] works offline and is mainly for Linux systems. It involves installation, which means it is not portable across non-Linux operating systems.

Mailpile [5] is an email client and also a personal Web mail server. It also has a much better user interface (in comparison to WMV) that is based on Gmail. It works on multiple browsers but does not work offline. It was coded using Python, JS and HTML5, and is the most relevant system to the one proposed in this paper.

3. DESIGN AND IMPLEMENTATION

3.1. Pre-processing

The design of the pre-processing components consisted of the following:

3.1.1. Parser

Before indexing can occur, email archives (sources/inputs) of different formats are streamed into the application. The parser extracts relevant data (constituents) from the email archives and passes this data on to the indexer. The parser was created in Python and made use of the Python mailbox module [11] as well as the Python multiprocessing module.

3.1.2. Indexer

The indexer creates inverted XML plain text file indices for searching and browsing through the archive. The indices are accessible to the Web browser running FINDMAIL, and can be parsed using Javascript. This pre-indexing process is slow compared to the actual search, but is necessary to obtain fast search results. The indices for browsing are sorted according to sender, date and subject. The indexer itself was also created using Python.

3.2. Final System Design

The final design of the system incorporates one indexer and one parser file written in Python. The parser file accepts both maildir and mbox formats, branching to either of the subclasses (parseMBOX or parseMDIR) depending on the path entered in the terminal.

At the moment, both the parser and indexer classes have code necessary to port the classes from Python 2 (2.7) to Python 3. This involved the use of the Python Future module, and all coding being done to support at least Python 2.7. This means that the parser and indexer will be able to run on Python versions 2 and 3 (specifically Python 2.7 and above).

The parser implements both the Python mailbox and multiprocessing modules. The mailbox module is used

to extract and perform various operations on the emails within the archive. The multiprocessing module, on the other hand, is used to speed up the run time of the whole program. It trades threads for processes and if a single instance of the Python interpreter is constrained by the GIL (a mutable lock that protects access to Python objects), one can achieve gains in concurrent workloads by creating multiple interpreter processes instead of threads. Each process is given a subtask of the program parallelized. The indexer class also uses the multiprocessing module for the same purpose.

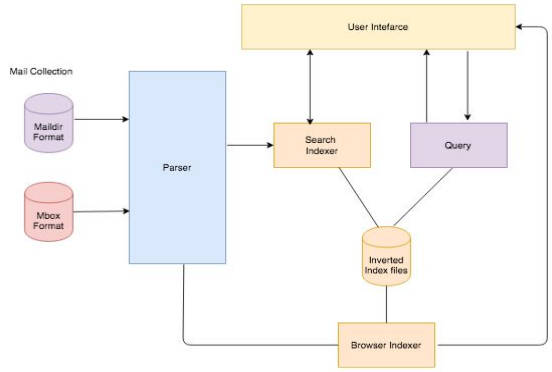

Fig. 1. Overview of the FINDMAIL system

In Fig.1 above, the overall design of the system is shown. The popular email formats, maildir and mbox, are inputted into the parser. The parser then sends its output to the two indexers (implemented between the two python classes). The browser indexer will create indices to facilitate browsing of the email, while the search indexer will create indices for the search functionality. Both of these indexers will interact with the user interface to provide the services of browsing and searching to the user.

The indices are in XML plain text format to aid portability, as explained in Section **3.1.2.** The particular algorithm used to index the documents is the inverted index information retrieval algorithm. The implementation of this algorithm is very similar to the one used to index the Bleek and Lloyd [3], with the only difference being that the weighting is calculated differently. Weights are taken to be the sum of all occurrences of the word within that particular email. The frequency of the occurrences is then later used to order the documents according to relevance.

4. FINAL EVALUATION AND RESULTS

4.1. Evaluation Metrics

4.1.1. Efficiency and Effectiveness

Efficiency refers to measuring the time taken (speed) to parse or index inputted archives, relative to the size of the archive (size referring to the number of emails in archive). The effectiveness, or accuracy, measures correctness of the parser and indexer output ie. whether the appropriate fields have been extracted fully from the email.

4.1.2. Portability

Portability is a measure of the parser's and indexer's ability to run on multiple operating systems, without terminating due to an error. For the FINDMAIL system, it also refers to the system being able to run on multiple versions of Python.

4.2. Experiment Design

Efficiency tests were conducted by running the parser and indexer on various datasets, and measuring the time they took to execute. The datasets were of different sizes (5000, 10000, 15000, 20000, 25000, 30000, 35000, 40000, 45000, 50000) and there were multiple tests done on each dataset (3 trials). All datasets were of maildir format and performance testing was done on a ASUS X555L laptop. These tests were done on a 16.04 Ubuntu Linux operating system.

The data collections used were synthetic email archives of maildir format. Synthetic meaning that the same emails were replicated and used in each data collection. This allowed for strict control over the number of files, as the exact number of files could be generated for each test.

The choice to use maildir instead of mbox, was so that testing could also be done to see the impact of the directory structure on the parsing and indexing time. In the case of an email archive containing folders with multiple levels of nesting, there could be errors or issues that arise when parsing and indexing that archive.

For effectiveness, testing was conducted by looking at the original email message and then the parsed message (HTML and XML), and comparing whether the

extracted fields were correct. This was done for 20 randomly chosen emails. For the HTML files of these emails, all the necessary fields were fully extracted and displayed to the user (relatively 100% effectiveness). For the XML files however, there were issues translating certain special characters. This is explained in detail in Section **5.1**. Therefore, effectiveness can be measured at 98% (98/100 fields correctly displayed) for the XML files.

4.3. Portability Tests

The parser and indexer of the FINDMAIL system was tested on Windows 10, Mac OS and Linux (Ubuntu 16.04 and 17.10) operating systems. The test for scalability of the parser and indexer was performed on an ASUS X555L laptop and an Ubuntu 16.04 operating system. The results of the tests showed that the parser and indexer could run on all the tested operating systems. The parser and indexer were also tested with different Python versions and this test showed that it could run on Python versions 2.7 and above. The python Future module was used to assess this [12]. Error handling was also done for Python versions 2.6 and below.

4.4. Analysis of Results

4.4.1. Efficiency and Effectiveness

In Appendix A, all the test results are shown. A summary of the test results is shown in Fig. 2 and Fig. 3.

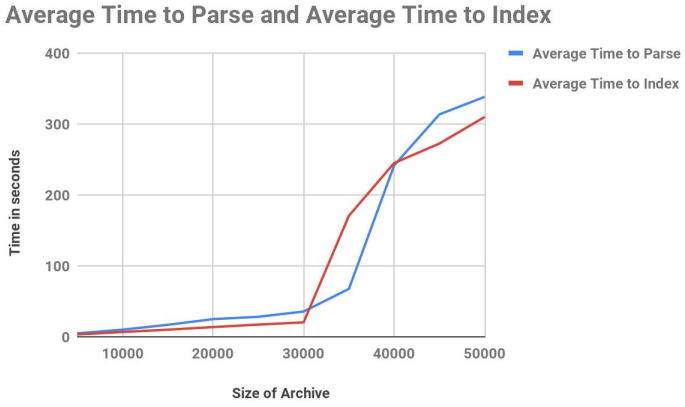

Fig. 2. Chart showing Time to Parse and Index various sized maildir archives

From the graph shown in Figure 2 above, we see that as the dataset size increases for the parser, there is relatively linear increase in time up until approximately size 30000. After size 30000, both the average parsing and indexing times drastically increase. The main reason for this phenomenon is unknown, but it could possibly be the nesting and structure of the maildir archives. The nesting within the sizes 40000- 50000 archives used for testing were up to six levels deep and this meant that more directories needed to be traversed and created for larger archives.

To test whether the nesting was causing the increase in parse and index times after size 30000, archives of size 30000-50000 archives were restructured to have just one level of nesting. They were then parsed and indexed and their results recorded. Fig. 3 shows the summarized results.

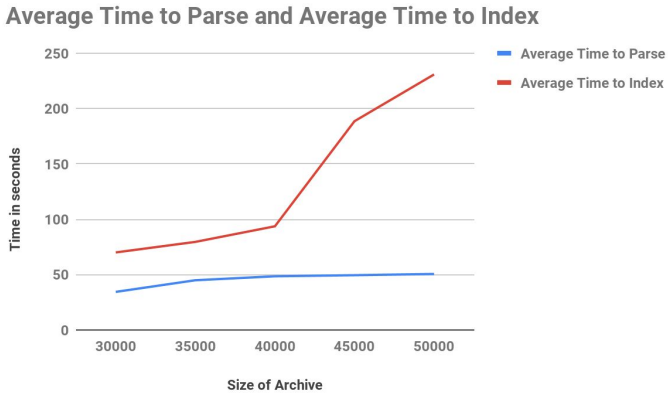

Fig. 3. Chart showing Time to Parse and Index various sized maildir archives with one level of nesting

From Fig. 3 above, we see that the nesting does have an effect on the time it takes to parse and index. The average time to parse now seems to increase linearly (with almost a gradient of 0) as the archive size increases. The average time to index however does not follow this trend; the gradient is much steeper and similar to that of Fig. **2**'s time to index. We can see however, that in general the average time to index the archive has decreased for all the archive sizes tested.

We can thus say that parsing and indexing is scalable in the instance where the maildir archive has one level of nesting and for small data sizes (around 30000 to 40000).

4.4.2. Portability

The parser and indexer, along with the entire FINDMAIL system were successfully run on Windows 10, Mac OS and Linux (Ubuntu 16.04 and 17.10)

operating systems. This indicates that the FINDMAIL system is portable across all the major operating systems tested.

5. DISCUSSION

5.1. Efficiency

The structure of the indexer is such that it has to go through every word in the original email and then write to an index file. This creates an IO bottleneck and results in the indexing process becoming slow for large email collections. Writing to the XML files cannot be avoided however, because the project is modeled with the guiding principle of simplicity. The purpose of the project was for the system to be portable, work offline and support digital preservation. As previously mentioned in Section **2.1**, developing countries can use simplicity to achieve all the previously mentioned points.

Another more important point to mention is scalability of the parser and indexer. It seems that the parser and indexer is scalable but only if the maildir archive has one level of nesting. It is not scalable for heavily nested maildir archives and shows drastic increases for datasets above 30000 in size. There could also be other reasons for time fluctuations observed in this situation, including an inefficient algorithm being used to parse and index the dataset, or the process of writing to files and folders causing the CPU speed to affect the total time of the process to run (CPU bottleneck).etc. This needs to be looked into and will be proposed as future work in Section **8**.

5.2. Effectiveness and Search

When measuring effectiveness, there were issues decoding special characters that were not recognized by the "ASCII" and "UTF-8" decoders. These special characters could not be written to XML files. Due to this issue, these characters had to be omitted from the final XML output.

With regards to search, some emails did not have certain fields such as "Subject", "Sender" or "Date". These emails could therefore not be sorted. To incorporate these emails to work with search and existing python sorting libraries, dummy fields such as "No Subject" were inserted into the XML files. Users could thus search for emails without a subject (or other relevant fields) and the sorting algorithm could group these emails for the user.

It was found that during parsing and indexing, Python's inbuilt multiprocessing module did not work as desired on Windows. The reason for this is due to the use of "fork()" in Python's multiprocessing module. On Linux and other Unix-like operating systems, Python's multiprocessing module uses fork() to create new child processes that inherit a copy of the parent process's memory state. This means the interpreter does not need to "pickle" the objects that are being passed as the process arguments, because the child process will already have them available in their normal form.

Windows does not have a fork() system call however, so the multiprocessing module needs to do more work when running the child-spawning process. This excess work is what leads to a "pickling error" when the multiprocessing module is run under the condition "if __name__ == '__main__' " , on Windows. In Python version 3.4, a new system was added to allow you to select the start method that you would prefer to use. For the current FINDMAIL system however, we do not implement multiprocessing on Windows as it would involve "pickling" all the objects passed as the process arguments. This would mean restructuring the code from scratch.

5.3. Portability

A number of different considerations were put in place to achieve the notion of portability. The index was composed of XML plain text documents to aid preservation and portability, the FINDMAIL system was coded in Python and using a browser to facilitate portability and the indexer and parser were ported to Python 3 for the same reasons. The FINDMAIL system now runs on multiple operating systems and Python versions, which means that portability has been achieved as defined in the scope of the project.

6. ETHICAL, PROFESSIONAL, AND LEGAL ISSUES

As this tool will be used to view and organize email archives, it was important to ensure that privacy of the users was maintained during testing. We thus used open access data, such as the Enron dataset [1], and our own personal email inboxes during testing. Efficiency and effectiveness tests were done without recruiting students. Thus, there were no ethical, professional and legal to consider when those particular tests were conducted.

7. CONCLUSIONS

7.1. The Parser and Indexer are Effective but can be Improved

The design of the FINDMAIL system was based on the design of the Bleek and Lloyd collection [3] and therefore simplicity was prioritized to implement portability. The plain XML files used for simplicity to create the inverted index resulted in there being an IO bottleneck upon indexing. Indexing is thus slower in comparison to other indexing methods for large datasets due of this. Writing to the files was necessary however, to allow for the FINDMAIL system to run offline on multiple platforms. Although, there is still room for further speedup in the case of improving the algorithm to index and implementing multiprocessing successfully on Windows.

7.2. The Parser and Indexer are scalable, but only for maildir archives with one level of nesting and small data sizes

For heavily nested maildir archives, the parser and indexer shows a deterioration in performance. For large maildir archives, there are usually multiple folders within folders that need to be traversed and this increases the time to parse and index. The indexer still shows a large increase in average indexing time for large datasets above 30000, regardless of nesting. This therefore means that scalability can still be improved on, at least with regards to the indexer.

7.3. The FINDMAIL system is portable on all major operating systems

The FINDMAIL system can run on Windows 10, Mac OS and Ubuntu Linux. These are the top 3 main operating systems used [6]. Parsing and Indexing work when using Python 2.7 on all these systems.

7.4. Parsing and Indexing work on both Maildir and Mbox formats

The current parser and indexer accepts both mbox and maildir formats of email archives. Although the speeds to parse and index differ depending on the format, the indices are successfully and accurately created for search afterwards regardless of the format.

7.5. The Parser and Indexer can work on Python 2 and Python 3

After integrating the parser and indexer to work with the Python future module, both the parser and indexer can now run on Python versions 2.7 and above. This will be particularly useful as an aid to portability as the FINDMAIL system can run on multiple versions of Python in addition to the major operating systems.

8. FUTURE WORK

If someone were to improve on this project, they could make the parser and indexer scalable for heavily nested maildir archives (2 levels and above of nesting in the maildir) and look into the reason for the drastic increases in time observed with datasets above 30000 in size. A closer look at the multiprocessing module and the file writing process would be helpful in this regard. They could also test the scalability for mbox archives if feasible and necessary.

Another point of improvement would be to have the code (particularly the indexer) handle special characters in emails better .ie. such that they can be displayed to the user.

Finally, multiprocessing can be properly implemented on Windows after “pickling” the objects passed as arguments to the Python processes, during parsing and indexing.

9. ACKNOWLEDGMENTS

I would like to give thanks to my fellow project team member, Breyden Monyemoratho, for his contributions and Hussein Suleman and Joseph Telemala for their guidance throughout the project as project supervisors. And an additional thank you to Michelle Kuttel for her valuable input as second reader of the project. I would also like to thank Sonia Berman for contacting the second year students on behalf of us.

10. REFERENCES

[1] Ludäscher, Bertram, Richard Marciano, and Reagan Moore. "Preservation of digital data with self-validating, self-instantiating knowledge-based archives." ACM Sigmod Record 30.3 (2001): 54-63.

[2] CALO Project. Enron Email Dataset, 2015.DOI: https://www.cs.cmu.edu/~enron/

[3] Centre for Curating the Archive. The Digital Bleek and Lloyd, 2018. DOI: http://lloydbleekcollection.cs.uct.ac.za/

[4] DSA. Department of Student Affairs, 2018. DOI: http://www.dsa.uct.ac.za/

[5] Mailpile. An email client, 2018. DOI: https://www.mailpile.is/

[6] Georgiev, Martin, Suman Jana, and Vitaly Shmatikov. Breaking and fixing origin-based access control in hybrid web/mobile application frameworks. *NDSS symposium*. Vol. 2014. NIH Public Access, 2014. DOI: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4254737/

[7] Jakob Nielsen and Rolf Molich. 1990. Heuristic evaluation of user interfaces. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '90), Jane Carrasco Chew and John Whiteside (Eds.). ACM, New York, NY, USA, 249-256. DOI=http://dx.doi.org/10.1145/97243.97281

[8] Netmarketshare. Operating system market share, 2018. DOI: https://www.netmarketshare.com/operating-system-market-share.aspx?

[9] David L. Parnas. Software aging. In Software Engineering, 1994. Proceedings. ICSE-16., 16th International Conference on (pp. 279-287). IEEE. May, 1994.

[10] Lighton Phiri and Hussein Suleman. In search of simplicity: Redesigning the digital Bleek and Lloyd. DESIDOC Journal of Library & Information Technology, (pp 32-34), 2012.

[11] Python 3 Standard Library. Mailbox module, 2018. DOI: https://docs.python.org/3/library/mailbox.html/

[12] Python 2.7 Standard Library. Future module. DOI: https://docs.python.org/2/library/__future__.html

[13] SourceForge. Mairix. Programme for indexing and searching mail, 2009. DOI: https://github.com/rc0/mairix/

[14] SourceForge. Windows Mbox Viewer, 2015. DOI: https://sourceforge.net/projects/mbox-viewer/

[15] Hussein Suleman. An African Perspective on Digital Preservation. In *Multimedia Information Extraction And Digital Heritage Preservation* (pp. 295-306), 2008.

[16] Hussein Suleman, Marc Bowes, Matthew Hirst, and Suraj Subrun. Hybrid online-offline digital collections. In *Proceedings of the 2010 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on - SAICSIT '10*, ACM Press, 421-425, 2010.

[17] Steve Whittaker, and Candace L. Sidner. Email overload: exploring personal information management of email. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 276-283). ACM. April, 1996.

APPENDIX A					
Figure 2 Tests					
Archive Size	**Class**	**Trial 1 (s)**	**Trial 2 (s)**	**Trial 3 (s)**	**Average**
5000	Parser	4.81	5.12	4.96	4.963333333
	Indexer	3.38	3.34	3.33	3.35
10000	Parser	9.15	10.59	10.4	10.04666667
	Indexer	6.82	6.71	6.72	6.75
15000	Parser	18.4	16.16	16.26	16.94
	Indexer	10.13	9.8	10.26	10.06333333
20000	Parser	25.61	26.01	23.15	24.92333333
	Indexer	13.85	13.88	13.49	13.74
25000	Parser	28.53	28.46	27.72	28.23666667
	Indexer	16.63	17.85	17.28	17.25333333
30000	Parser	34.2	36.12	36.62	35.64666667
	Indexer	20.21	20.07	20.87	20.38333333
35000	Parser	67.91	64	70.69	67.53333333
	Indexer	177.59	182.62	151.41	170.54
40000	Parser	253.9	247.95	222.93	241.5933333
	Indexer	266.92	232.87	235.08	244.9566667
45000	Parser	315.71	316.55	308.47	313.5766667
	Indexer	275.95	273.12	268.13	272.4
50000	Parser	348.81	329.84	336.21	338.2866667
	Indexer	296.32	315.57	318.12	310.0033333
Size of Archive	**Average Time to Parse**	**Average Time to Index**			
5000	4.963333333	3.35			
10000	10.04666667	6.75			
15000	16.94	10.06333333			
20000	24.92333333	13.74			
25000	28.23666667	17.25333333			
30000	35.64666667	20.38333333			
35000	67.53333333	170.54			
40000	241.5933333	244.9566667			
45000	313.5766667	272.4			
50000	338.2866667	310.0033333			
Figure 3 Tests- One level of Nesting					
Archive Size	**Class**	**Trial 1 (s)**	**Trial 2 (s)**	**Trial 3 (s)**	**Average**
30000	Parser	38.3	33.48	31.44	34.40666667
	Indexer	86.12	66.04	57.97	70.04333333
35000	Parser	43.07	44.08	47.6	44.91666667
	Indexer	77.87	78.96	81.87	79.56666667
40000	Parser	50.26	44.09	51.16	48.50333333
	Indexer	97.38	86.21	97.39	93.66
45000	Parser	57.14	46.12	45.15	49.47
	Indexer	171.07	207.89	186.79	188.5833333
50000	Parser	48.7	48.42	54.64	50.58666667
	Indexer	231.73	230.87	229.87	230.8233333
Size of Archive	**Average Time to Parse**	**Average Time to Index**			
30000	34.40666667	70.04333333			
35000	44.91666667	79.56666667			
40000	48.50333333	93.66			
45000	49.47	188.5833333			
50000	50.58666667	230.8233333			