國 立 交 通 大 學

資訊科學與工程研究所

碩 士 論 文

藉由減少 I/O 以及磁碟資料區塊不連續現象以增進 POP3 伺服器效能

Improving Performance of POP3 Server by Reducing I/O and Disk Data Block Uncontinuity

研 究 生：吳旻儒

指導教授：張瑞川　教授

中 華 民 國　九 十 六　年 六 月

藉由減少 I/O 以及磁碟資料區塊不連續現象以增進 POP3 伺服器效能

學生：吳旻儒　　指導教授：張瑞川教授

國立交通大學資訊科學與工程研究所

論 文 摘 要

隨著近年來網路的發達程度，越來越多人使用電子郵件，而因為網路速度越來越快，所以收取信件的速度取決於 POP3 伺服器的效能，所以改善 POP3 伺服器效能可望有效增進收取郵件速度。

根據我們觀察實際郵件伺服器得知，當信件檔案變大的時候在磁碟上的分布往往變的破碎，同時根據我們的先期實驗發現，同樣大小的信件檔案根據不同的連續程度存取的速度也有相當大的差距。所以我們提出一個機制，此機制可以做到在刪除信件的過程中，即時的進行減少碎裂部份的功能，同時又可以減輕原本伺服器在刪除信件時所需要耗費的大量資源。由實驗結果可顯示我們所提出之機制可確實達到所以我們所提出之改善。

Improving Performance of POP3 Server by Reducing I/O and Disk Data Block Uncontinuity

Student: Min-Ju Wu **Advisor: Prof. Ruei-Chuan Chang**

Computer Science and Engineering College of Computer Science

National Chiao Tung University

Abstract

Email is one of the most widely-used network services in our daily life. Because the network speed is more and more fast, clients are more sensitive about the performance of POP3 server when retrieving mails. So we can fasten the duration of retrieving mails by improving the performance of POP3 servers.

According to our prior observation on a real POP3 server, we find that when the mail folders grow larger, they are more fragmented on disk. And we find that it takes more time to read a more fragmented mail folder. In this thesis, we proposed a mechanism to do defragmentation during mail deletions, and we can mitigate the large amount of disk I/O and memory access caused by mail deletion. As shown in experimental result our mechanism can do defragmentation more effectively with lower overhead than original mechanism.

致謝

首先要感謝我最尊敬的指導老師，張瑞川教授。在老師的細心指導下，這兩年來讓我獲益良多，瞭解到正確的做研究的態度及方法，也才得以完成此篇論文。而在撰寫論文的過程中，也十分感謝張大緯學長從頭到尾都給予我十分寶貴的建議。

同時要感謝實驗室的博士班黃亭彰學長，在產生論文的過程與我討論給予我一些建議以及方向。以及其他兩位張明絜學長以及邱國政學長的支持與鼓勵。也感謝實驗室同學以及學弟們的陪伴以及共同努力。

也感謝交大籃球隊的士勛學長，學君學長，暉智學弟以及好友彥樵在這段時間支持與關心，陪我在用功之餘紓解壓力。

最後感謝我的父母，姐姐，以及女友宜臻在這段期間的陪伴鼓勵，支持著我，關心著我，給予我最大的動力來完成這篇論文。最後僅以此論文，獻給我最摯愛的你們。

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1 Introduction

1.1 Motivation

Email is one of the most widely-used network services in our daily life. Many people check their email several times a day by using email client applications, and most of the applications retrieve mail from the servers via the 3rd version of the Post Office Protocol (POP3)[14]. In POP3, a client retrieves mail by first starting a new session with the server, and then downloading the mail specified by the user. During the session, the client can also delete some user-specified mail from the server. Finally, the client terminates the session by sending a *quit* command to the server.

After clients retrieve mails by *retr* command, these mails are still not deleted unless clients issue *dele* command. But clients will not delete these mails which are useful, so they deleted useless mails and back up useful mails over server side. Then the mail folders of these clients grow large. However, current POP3 servers have two performance problems when handling large mail folders. First, starting a POP3 session that corresponds to a large mail folder usually requires a long time. This is because a POP3 server has to scan the whole mail folder corresponding to the client at the beginning of a POP3 session in order to obtain the amount of the mail and the size of each mail. In most mail servers, each mail folder is implemented as a single file, and therefore a POP3 server has to read the whole file at the start

of each POP3 session. Since mail files are usually fragmented due to the concurrent handling (i.e., reception and deletion) of multiple folders by the server, accessing these files requires a large amount of disk seek operations and thus results in a long client-perceived wait time. The second problem is that a POP3 server may cause a large number of disk IO operations when a user deletes his mail from a large mail folder, especially when the deleted mail resides close to the beginning of the mail file (i.e., older mail)., This is because a POP3 server has to move the data blocks behind the deleted part forward.

In order to mitigate the two problems, we have to reduce the number of fragments in mail folders and the amount of disk IO operations caused by mail deletion. Disk reorganization is the most straightforward method for file defragmentation. However, the large amount of disk IO operations caused by disk reorganization may have a noticeable performance impact to the POP3 server. Moreover, disk reorganization cannot reduce the amount of disk IO operations caused by mail deletion.

In this thesis, we proposed a new mail folder update mechanism for POP3 servers to mitigate the aforementioned performance problems. The basic idea of the proposed mechanism is to avoid data block movement due to mail deletion unless such movement can reduce the fragment number of the mail folder. Specifically, we move the fragments following the deleted mail forward when the data blocks containing the deleted mail together with the adjacent free blocks can hold at least two of the following fragments. Otherwise, we

only free the data blocks occupied by the deleted mail.

The benefits of the proposed mechanism are four fold. First, the proposed mechanism minimizes the amount of disk IO operations caused by mail deletion as well as reduces the fragment numbers of mail folders. Second, it incurs little overhead. It requires much fewer disk IO operations when compared to disk reorganization techniques. Third, the mechanism does not require any code modification or recompilation on POP3 servers. It is a file system level mechanism, and takes advantage of file system information to improve the performance of POP3 servers. Fourth, the mechanism is not tight to a specific POP3 server since it considers the general behavior of POP3 servers. Therefore, the mechanism can be applied on more than one POP3 servers.

We implemented the proposed mechanism in Linux ext2 file system. According to the performance results, our mechanism reduces the IO access time caused by mail deletion into a small and nearly-constant time and makes clients do not need to wait for logout. Moreover, our mechanism has shortened the AUTHORIZATION state about 23.7%.

1.2 Thesis Organization

The rest of the thesis is organized as follows. In Chapter 2, we introduce the related work about disk performance improvement. In Chapter 3, we describe the design and implementation of our mechanism, which is followed by the performance evaluation in Chapter 4. In the end, we give conclusions in Chapter 5.

Chapter 2 Related Work

Several techniques have been proposed to improve disk performance by rearranging data on the disk. In this section, we briefly describe these techniques, which can be classified into 2 categories: disk defragmentation, hot-data concentration.

2.1 Hot-Data Concentration

Hsu *et al*.[10] proposed grouping hot data and sequentially accessed data into a specific area on the disk so that most disk accesses can be done in that area, reducing the disk seek time.

If we try to improve the performance of disk I/O by grouping hot data, there are some problems. As we mentioned above, POP3 server read the whole mail folder after clients logged in. Then, after mail deletion, POP3 servers sequentially move all the data blocks behind the deleted mails forward. So the latter mails are a little hotter than former mails. But, if we copy these mails into reorganization area, when the POP3 server reads the whole mail folder disk head stays over original place instead the reorganization area because the former mails are not in reorganization area. Then, if there are mail deletions, the POP3 server sequentially move the mails forward, but we cannot guarantee all these mails are in reorganization area, so disk head may need to start to read from the original place. So grouping hot data is not suitable for POP3 servers.

2.2 Disk Defragmentation

Chee *et al.*[5] proposed a disk block reorganization mechanism by exploiting the characteristic of disk writes in a log-structured filesystem. In a log-structured filesystem, write operations do not modify the original data blocks. Instead, write requests are batched and the corresponding blocks are written to the end of the log. By exploiting this characteristic, when the disk reorganizer is called up, it can place these data to be written by the access pattern.

The mechanism can only be used in a log-structured Filesystem, and it focuses on small write. If data size to be written is larger than cache size they will be written out to disk soon. Moreover, in POP3 server every mail deletions there is lots of data to be written, then according to this mechanism these data in one deletion will to be written together, but after several mail deletions in different sessions it may separate one mail folder into several place on disk. So this mechanism cannot solve our problem.

Wang and Hu[24] proposed a zone based reorganization mechanism for log-structured Filesystem. Based on the fact that outer zones of a hard disk drive has a higher data transfer rate, the mechanism moves frequently accessed data to the outer zones, during garbage collection time to reduce the overhead.

Similar to the previous mechanism, this mechanism can only be used in a log-structured filesystem. When using this concept over POP3 server on Ext2, how to move data online is another problem, if there are mails coming when moving this mail folder will be more

complicated. Besides, the mechanism does not help to mitigate the large amount of disk I/O and memory access caused by mail deletion.

Offline defragmenters such as e2defrag[9] can perform defragmentation over an unmounted file system. So we need an online defragmentation tools to solve the problems on POP3 servers.

2.3 Access Pattern Predictor

Optimizing disk layout usually requires the knowledge of the data access patterns. For example, hot blocks and sequentially accessed data have to be identified before they can be collocated. Long *et al.*[2][3][4][12][13][16][22][26][27][28][29][30] proposed several predictors by analyzing lots of statistics.

In POP3 sessions, servers read the whole mail folder after clients logged in, and the access pattern is known to be reading the whole mail folder. And after each deletion, data blocks after deleted part will be accessed. We know that when deletions happen, the access pattern will be reading every data blocks after deleted part. So we do not need to spend overhead of accumulating statistics to get information about access pattern.

From the above, we can know that the defragmentation and reorganization techniques nowadays are not suitable to solve the problems. In this thesis we proposed a new mechanism to do defragmentation online to solve the problems over POP3 servers.

Chapter 3 Design and Implementation

In this chapter, we introduce design and implementation of the proposed mail folder update mechanism for POP3 servers. In Section 3.1, we simply show the states of POP3 server, two formats of mail folder and the main problem to be dealt by the proposed mechanism. In Section 3.2, we introduce how to reduce overhead of memory access and disk I/O in UPDATE state. In Section 3.3, we introduce how to reduce the disk seek time of reading the whole mail folder. In Section 3.4, we introduce prototype of the proposed write mechanism.

3.1 The Problems

Figure 1 shows the state transition diagram of a POP3 session, which is defined in RFC 1939. As shown in the figure, a session starts by entering the AUTHORIZATION state, in which the server authorizes the client. After the client has been authorized, the session enters the TRANSACTION state and the client can then make requests to obtain mail folder information, retrieve mail, mark mail as deleted, and etc. When the client issues the *quit* request, the session enters the UPDATE state. In that state, the server removes the marked-as-deleted mail from the mail folder and then terminates the session.

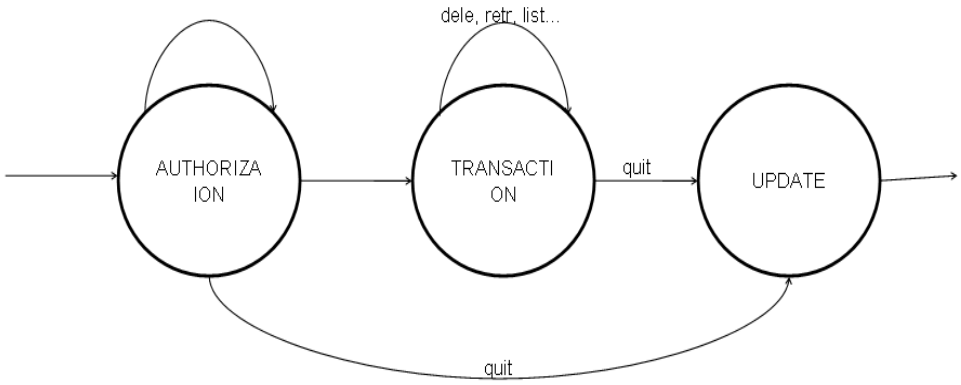

Figure 1: State Transition Diagram of a POP3 Session

Most email servers (e.g., the top three email servers on Linux, Sendmail[1][21], Qmail[19], and Postfix[18]) store mail messages corresponding to a specific user in a single file. There are two reasons for clients to left a copy of their mails on server side. First, clients back up their mails on server side, so they can get these mails back when their system crash. Second, clients with mobile devices may wants to retrieve mails on different devices, if there is no mail backed up on server side, clients will not be able to retrieve mails on different mobile device. As stated in the Introduction, current POP3 servers have two problems on handling large mail files efficiently. First, scanning a large and fragmented mail file at the beginning of a POP3 session causes a large amount of disk seek operations and thus results in a long client-perceived wait time. Second, in the UPDATE state, a POP3 server (e.g., Dovecot [8], UW-imap [23]) removes the mark-as-deleted messages from the mail file by moving all the messages following the deleted ones forward, causing a large number of disk IO operations. In this chapter, we describe a novel mail folder update mechanism that can

mitigate the above two problems. By performing defragmentation and reducing data block movement during mail deletion, the mechanism can reduce the client-perceived wait time in the AUTHORIZATION state and the IO load in the UPDATE state. In the following sections, we describe the design issues and implementation details of the proposed mechanism.

3.2 Reducing UPDATE State Overhead

In this section, we show how to reduce the overhead of updating mail folders. Because current file systems are not able to delete data from the middle of a file. So the mail deletion of POP3 server modifies every memory pages and disk blocks behind the deleted data as Figure 2 shown.

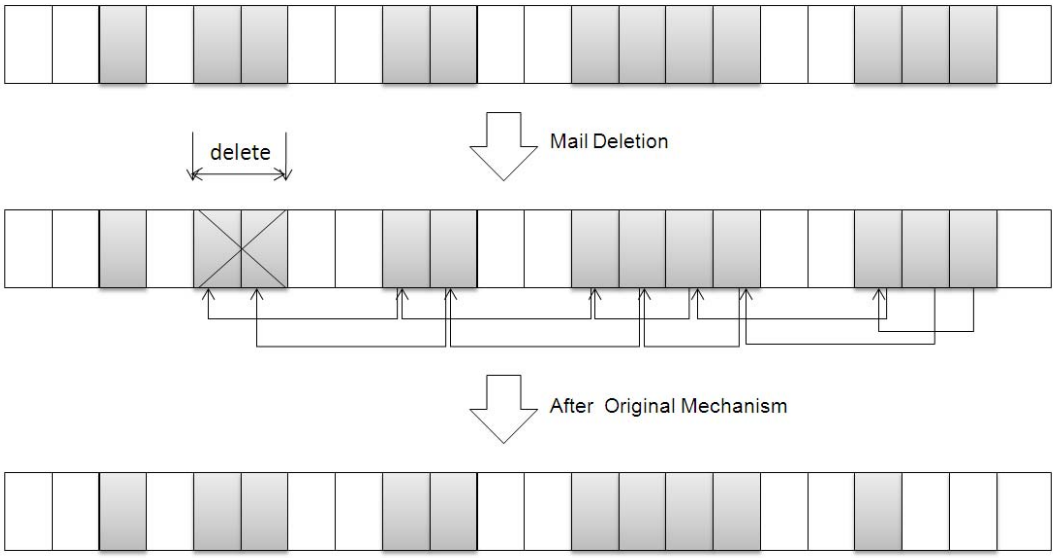

Figure 2: An Example of Mail Deletion of Original Mechanism

The main idea of the proposed mechanism is to free the memory pages and data blocks occupied by deleted part to reduce the overhead of mail deletions. In order to do so, we have to know the memory pages and disk blocks corresponding to the start and end of deleted part.

The way we take is as following:

First, we copy the return content and the file offset of every read system calls to an array. Then, every time the POP3 server calls a write system call, we compare the content of the every entry of the array and the data to be written by strcmp(). Because every mail has different a message identifier even if the content of these mails is the same. The file offset of mapped read system call is the end of deletion, and the file offset of write system call is the start of deletion. In order to avoid wasting memory space, we record the results of 30 read system calls.

After we get the information about the start and the end of deleted part, we start to free memory pages and disk blocks occupied by the deleted part. We free memory pages by calling truncate_inode_pages_range() to remove these pages, and free disk blocks by calling ext2_free_blocks(). We also modify the related file system information(ex. ctime and i_blocks in inode, data block bitmap) to avoid file system inconsistency. When the deleted part is less than one memory page, we just call memset() to clear the deleted part.

In the original mechanism, POP3 server will keep reading the data behind the deleted part by read system calls and write them forward by write system calls. In our mechanism, we have to filter out such read and write system calls. We use the following method to filter out such read and write system calls.

First, if the file offset of read is equal to file offset of previous read plus the return

value of previous read, we consider the read as an unnecessary read, because and do not read actually. Otherwise, there is a deletion, and we will handle this situation during next write. Finally, if POP3 server issues a write after several unnecessary reads, we consider the write as an unnecessary write and do not write actually because it is trying to move data forward. Otherwise, there is a deletion, and we start the procedure of finding the start and the end of the deleted part.

As shown in Figure 3, we know there is a read starting from file pointer position A and read data size of B. If next read starts from A+B, we consider it as an unnecessary read. Otherwise, if next read start from A+C (C>B), we consider that there is a deletion with the start is A+B and the end is A+C.

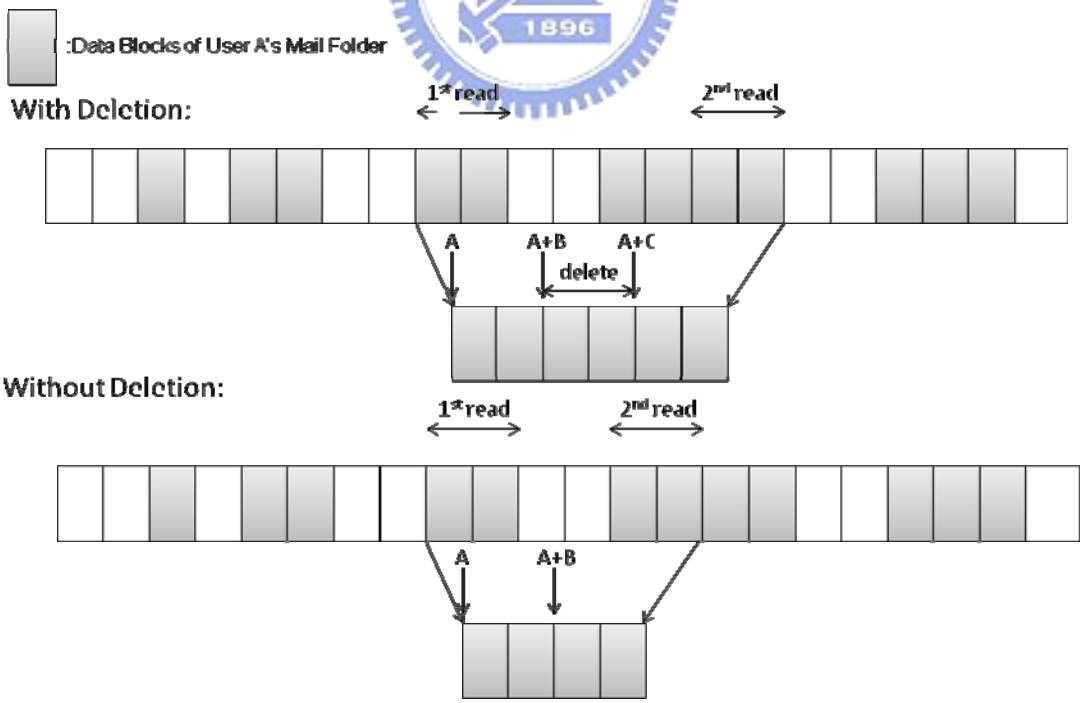

Figure 3: An Example of Block Deletion

According to the above description, the proposed mechanism can reduce the load of disk

IO and memory access effectively, and thus shortening the time spent on the UPDATE state. of client waiting after quit from POP3 session.

3.3 Speed up AUTHORIZATION State

In this section, we introduce that how to reduce the disk seek time of reading the whole mail folder by reducing fragments of mail folders on disk. In the progress of reading the whole mail folder in POP3 session, we cannot reduce the data transfer time, so we want to reduce disk seek time to speed up the AUTHORIZATION state. Lu Jun[11] proposed that most disk seek time gap happens when disk seek distance increase a little from zero as Figure 4 shows[11]. Steven's observation[18] has a similar result. So we try to reduce number of seeks to reduce disk seek time.

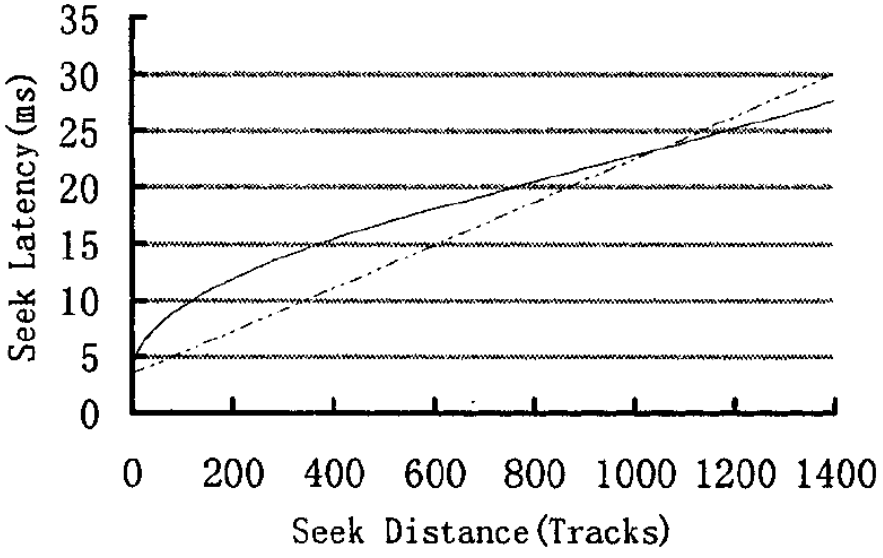

Figure 4: Relationship between seek distance and disk seek time

In reading the whole mail folder, we have to reduce number of fragments to reduce number of seeks. Because large overhead of disk reorganization, we do defragmentation by

another way. If there is enough continuous free blocks after mail deletion, we move data blocks to reduce number of fragments. After we free memory pages and disk blocks occupied by deleted part in every mail deletion, we do defragmentation as following:

First, if the former one and the latter one data block of deleted part on disk are occupied by same mail folder as the deleted part, we move the latter data blocks forward to compact this fragment. Otherwise we will handle in next stage.

As Figure 5 shows, the deleted part are logical block number 875 and 876, and we find that logical block number 874 and 877 are occupied by same mail folder as the deleted part. So we move the blocks to compact.

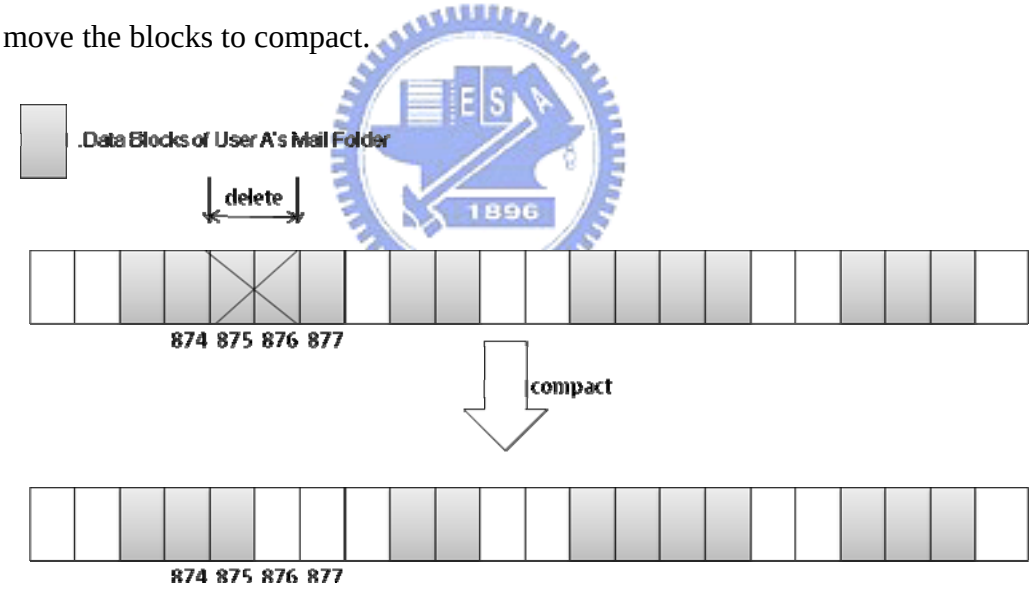

Figure 5: Compacting a Mail Folder in a Mail Deletion

After we consider about compacting in a fragment, we will consider about defragmentation in different fragments as following:

First we calculate the length of free blocks from the former one to the latter one data block in this mail folder by scanning the data block bitmap with data blocks which is mapped

by the former one to the latter one data blocks in this mail folder. Then we calculate the length of fragments behind the deleted part by scanning the i_data field in ext2 inode, and compare with the length of free blocks. If the length of free blocks is larger than the length of at least first two fragments, we move data blocks of these fragments to these free blocks. Finally, we update the related information of inode.

As Figure 6 shows, the deleted part are the fifth to eighth data blocks in this mail folder, and these blocks are mapped to logical block number 570 to 573. Then we check the fourth and ninth data blocks in this mail folder which are mapped to logical block number 568 and 575. We can find that the length of longest continuous free blocks is 5, the length of first fragment is 2, the length of second fragment is 3, and the length of third fragment is 2. So we can move the first and the second fragment into free blocks, and reduce one fragment.

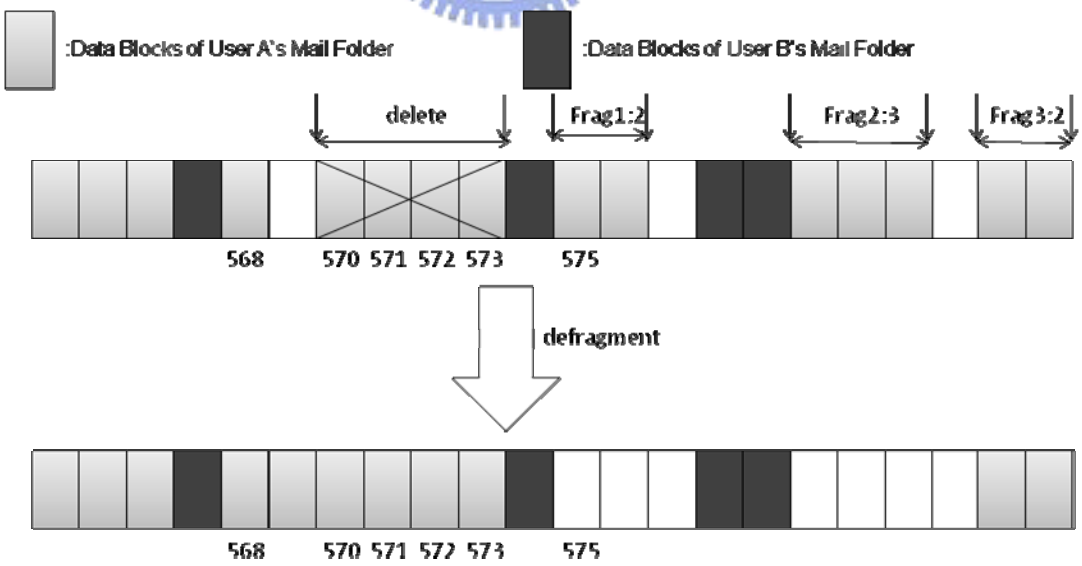

Figure 6: Example of Defragmentation

3.4 Write Mechanism Prototype

In this section, we propose a Write Mechanism Prototype which integrate two components mentioned in sections above. Our mechanism will only be motivated in UPDATE state when there is a mail deletion in TRANSACTION state. Figure 7 shows the integrated flow chart. When we find there is write request issued by POP3 server, we will do as following:

First, we record the information about read and write system call. Second, if there is a mail deletion, we simply free the memory pages and disk data blocks. Third, if the fragment with deleted part can be compact, we compact this fragment. Fourth, we calculate the length of free blocks. Fifth, we calculate the length of fragments. Finally, if the length of free blocks larger than the length of fragments, we do defragmentation.

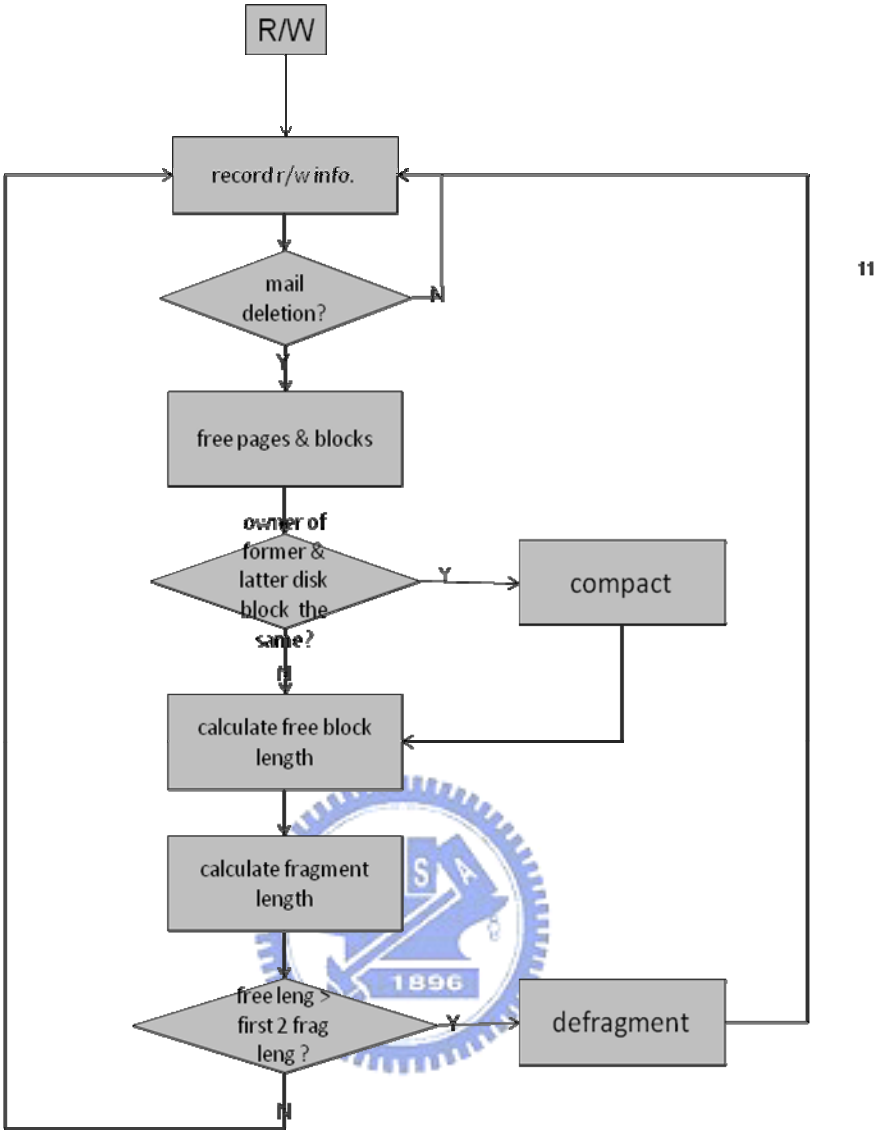

Figure 7: Flow Chart of Our Mechanism

We can divide implementation into two parts, first part is a monitor and second part is the proposed write mechanism. The monitor is implemented in Virtual File System(VFS), it modifies sys_read() and sys_write() to intercept the content of read and write, and file pointer position. The proposed write mechanism is implemented in the Second Extended Filesystem (Ext2 filesystem) as a kernel module, it modifies do_generic_mapping_read() and generic_file_buffered_write() to get information about memory pages and inode. There are two components in our kernel module, one is compactor to compact fragment and the other is

defragmentor to do defragmentation between fragments as mentioned above. Figure 8 shows the architecture of our mechanism.

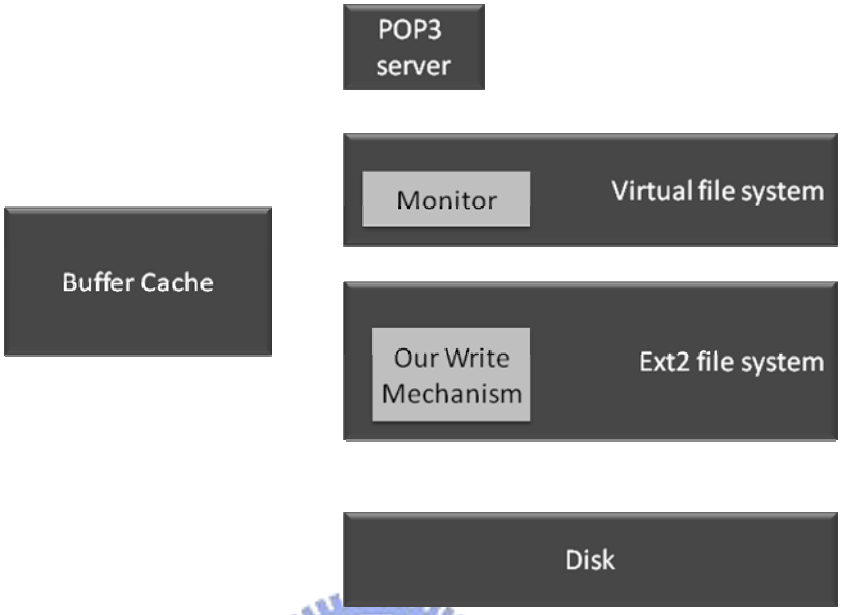

Figure 8: Architecture of Our Mechanism

Chapter 4 Performance Evaluation

In this chapter, we evaluate the performance improvements of the proposed mechanism in different states of a POP3 session. Section 4.1 introduces the experimental environment. Section 4.2 and 4.3 compare the performance of the proposed mechanism with the original one in the AUTHORIZATION and UPDATE states, respectively.

4.1 Experimental Environment

The experimental environment consists of a server machine and a client machine, which are connected by a D-Link Gigabit Ethernet switch. Each of the machine is equipped with an Intel Pentium 4 3.2 GHz processor, 1GBytes DDR RAM, and a Seagate 40GBytes 7200rpm disk. We run Linux 2.6.17.13 and the UW-imap POP3 server on the server machine.

We use the Postal benchmark [17] to generate the email workload and to measure the performance.

4.2 Performance Improvement in the AUTHORIZATION State

In this section, we evaluate the effectiveness of our defragmentation mechanism. We created 20 mail folders in an empty partition and then backup the partition including the layout by *dd* command. The size of these mail folders is about 500Mbytes and each of them are composed of about 80000 mails and maximum mail size is 10Kbytes. We randomly

deleted a various number mails from a folder, and measured the number of fragments of this folder after the mail deletion had completed. Then we recovered the disk layout by *dd* command and measured the number of fragments after deleting the same mails with our mechanism.

Because the duration of reading the whole mail folder in AUTHORIZATION state is proportional to number of fragments of this mail folder. In this experiment, we measure the performance improvement, on the AUTHORIZATION state, of the proposed mechanism. We measured the duration from client login in to welcome message shown. We use a kernel module to get the experimental result from kernel message by printing out durations of every state. Figure 9 shows the results. The x-axis represents the number of deletions, and the y-axis represents the time for reading the whole mail folder. As shown in the figure, our defragmentation mechanism can reduce the time for reading the whole mail folder by about 23.7% after 40000 mail deletion operations have been performed.

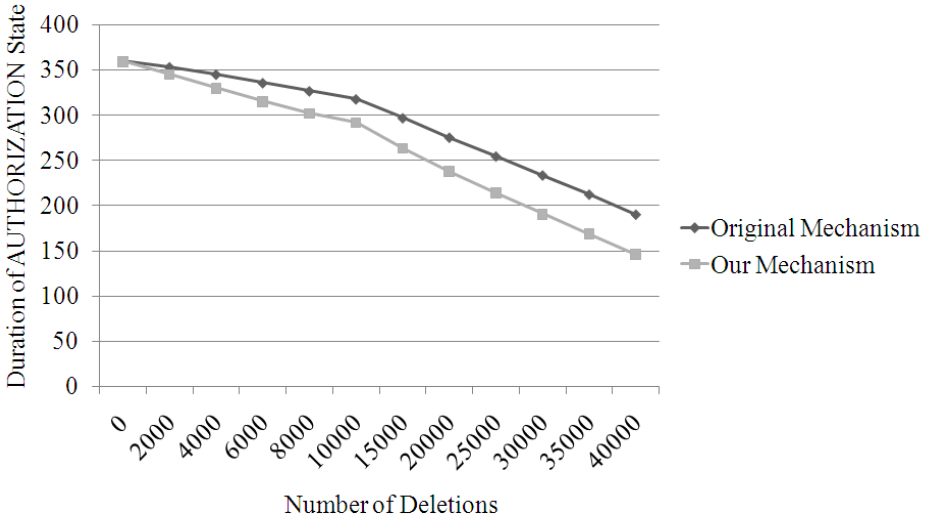

Figure 9: Comparison on AUTHORIZATION State Duration

Figure 10 shows the result of performance of defragmentation of our mechanism after mail deletions. The x-axis represents the number of deletions, and the y-axis represents the fragment numbers. As shown in the figure, 50% of fragments can be eliminated after 40000 mail deletion operations have been performed.

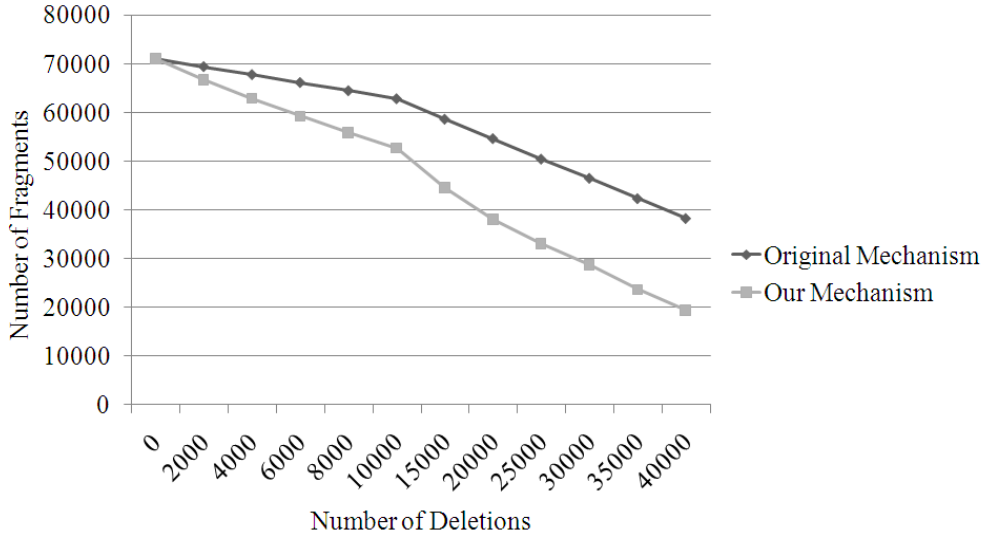

Figure 10: The Effectiveness of Defragmentation

4.3 Performance Improvement in the UPDATE State

In this section, we evaluate the performance in UPDATE state by measuring the UPDATE state duration. Before the experiment, we generated an initial set of mail folders on an empty disk partition by using the Postal benchmark. The initial set is composed of 35 mail folders ranging from 5M to 320M bytes, and the maximum mail size is 10K bytes.

During the experiment, we deleted one mail in each mail folder which ranges 5M to 320M bytes before the end of the mail file, and measured the duration of the UPDATE state. We get the duration by intercepting the *quit* command and the end of *read* system call issued

by POP3 server.

Figure 11 shows the update duration of the original and the proposed mechanism. The x-axis represents the *deletion distance*, which is the number of bytes from the last byte of the deleted mail to the end of the mail file. The y-axis represents the duration of the UPDATE state.

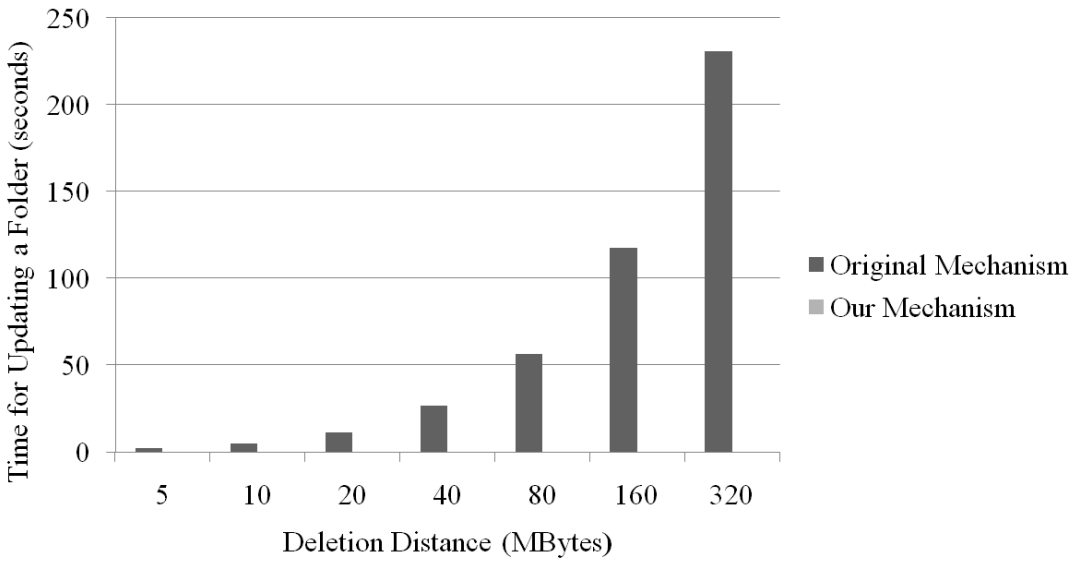

Figure 11: Comparison on UPDATE State Duration

As shown in the figure, the update state duration increases with the growth of the deletion distance under the original mechanism. The increment is due to the increase of the disk I/O for writing dirty pages to disk. On the contrary, our mechanism just frees the memory pages and the disk blocks occupied by the deleted mail, and hence the time for updating a mail folder is almost constant.

Figure 12 shows the number of disk IO during the update process of the original and the proposed mechanism. The x-axis represents the *deletion distance*, and the y-axis represents

the number of disk IO.

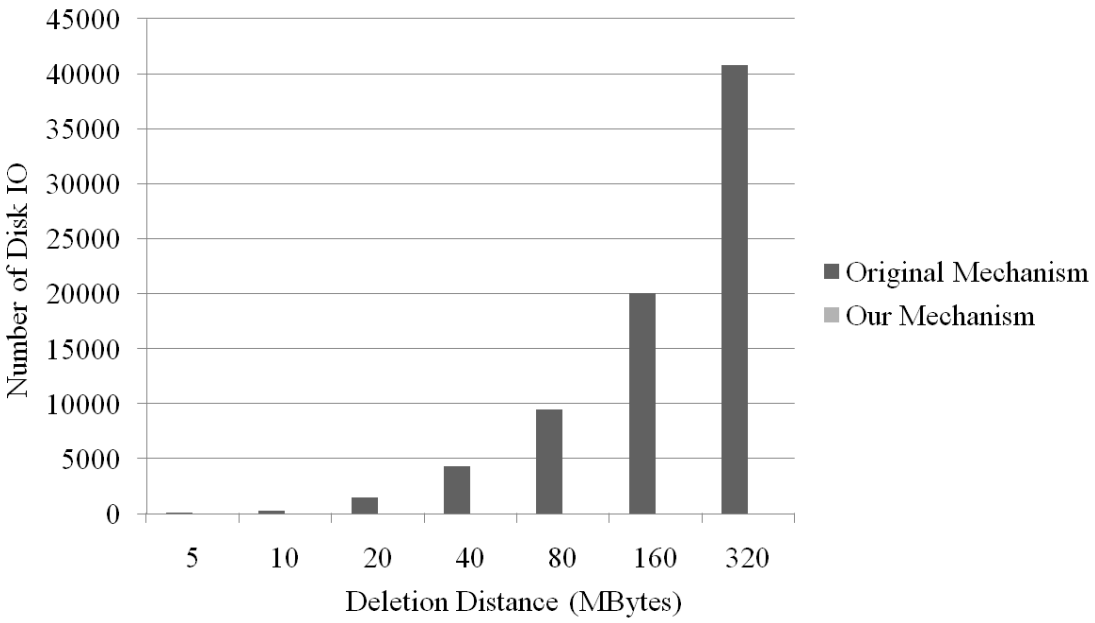

Figure 12: Comparison of Number of Disk IO

As the figure shown, the number of disk IO increases with the growth of the deletion distance under original mechanism. But under our mechanism, the number of disk IO is almost constant.

Figure 13 shows the average CPU utilization during the update process of the original and the proposed mechanism. The x-axis represents the *deletion distance*, and the y-axis represents the CPU utilizations that were obtained by using the *top* command.

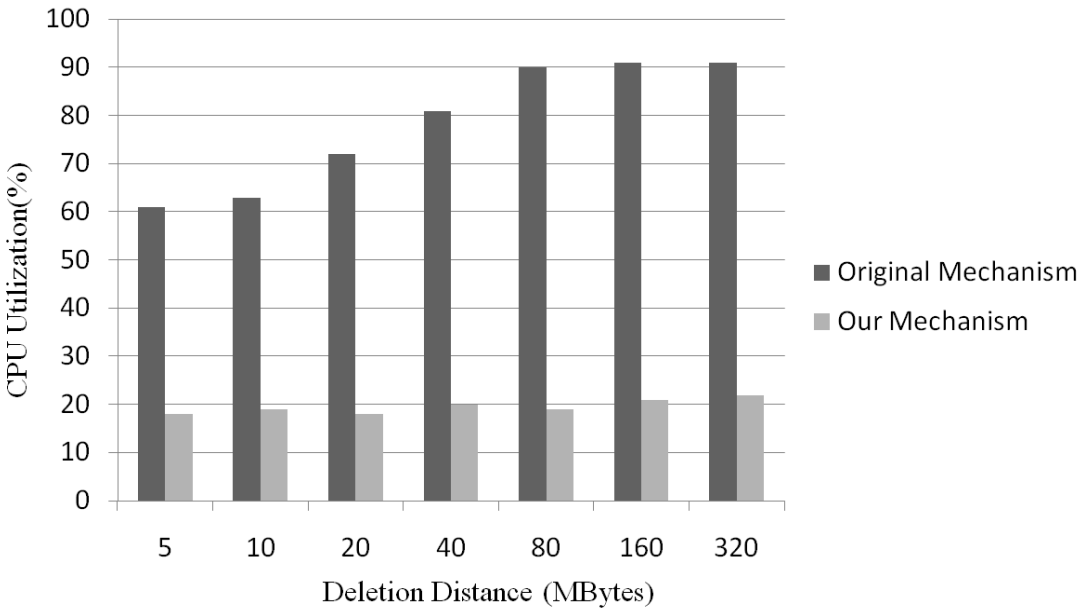

Figure 13: Comparison on CPU Utilization during the UPDATE state

Similar to the previous figure, the CPU utilization increases with the growth of the deletion distance due to the increment of the disk traffic. By contrast, our mechanism leads to a constant CPU utilization since we just frees the memory pages and the disk blocks occupied by the deleted mail, and the amount of memory access and disk I/O is almost constant. From the above two figures we can see that, our mechanism can effectively reduce the load, and hence improve the performance, of updating mail folders.

Chapter 5 Conclusion

In this thesis, we proposed a mechanism to improve the performance of POP3 servers by performing defragmentation on Ext2 filesystem. Our mechanism can do defragmentation during mail deletions, and mitigate the large amount of disk I/O and memory access caused by mail deletion.

We implemented the proposed mechanism in Linux ext2 file system. According to the performance results, our mechanism reduces the IO access time caused by mail deletion into a small and nearly-constant time and makes clients do not need to wait for logout. Moreover, our mechanism has shortened the AUTHORIZATION state about 23.7%.

Reference

[1]. Eric Allman, “SENDMAIL-An Internetwork Mail Router”, Issued with the BSD UNIX documentation set

[2]. Ahmed Amer and Darrell D. E. Long, “Aggregating Caches: A Mechanism for Implicit File Prefetching”,*Proceedings of the Ninth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (MASCOTS), Cincinnati: IEEE, August 2001

[3]. Ahmed Amer, Darrell D. E. Long, Jehan- François Pâris and Randal C. Burns, “File Access Prediction with Adjustable Accuracy”, *Proceedings of the International Performance Conference on Computers and Communication* (IPCCC), Phoenix: IEEE, April 2002

[4]. Karl Brandt, Darrell D. E. Long and Ahmed Amer. “Predicting When Not To Predict”, *Proceedings of the Twelfth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (MASCOTS), Volendam, The Netherlands: IEEE, October 2004

[5]. C. L.Chee, H. Lu, H. Tang and C. V. Ramamoorthy, “Improving I/O Response Times Via Prefetching and Storage System Reorganization”, 21st International Computer Software and Applications Conference,1997

[6]. Wenjing Chen, Christoph F. Eick and Jehan-François Pâris, “A Two-Expert Approach to File Access Prediction”, Proceedings of the third International Information and Telecommunication Technologies Symposium (I2TS), Sao Carlos, Brazil, December 2004

[7]. Inchul Choi and Chanik Park “Enhancing Prediction Accuracy in PCM-Based File Prefetch by Constained Pattern Replacement Algorithm”, LECTURE NOTES IN COMPUTER SCIENCE, 2003

[8]. Dovecot- http://www.dovecot.org/

[9]. e2defrag- http://e2compr.sourceforge.net/attic/defrag.html

[10]. Windsor W. Hsu, Alan Jay Smith and Honesty C. Young, “The Automatic Improvement of Locality in Storage Systems”, ACM Transactions on Computer Systems (TOCS), 2005

[11]. Lu Jun, Lu Xianliang, Luo Guangchun, Han Hong and ZhouXu, “STFS: A Novel File System for Efficient Small Writes”, ACM SIGOPS Operating Systems Review, 2002

[12]. Thomas M. Kroeger and Darrell D. E. Long, “Design and Implementation of a Predictive File Prefetching Algorithm”, *Proceedings of Usenix Technical Conference*, Boston: Usenix Association, June 2001

[13]. Thomas M. Kroeger and Darrell D. E. Long, “The Case for Efficient File Access Pattern

Modeling", Proceedings of the Seventh Workshop on Hot Topics in Operating Systems, 1999.

[14]. J. Myers and M. Rose, "RFC1939-Post Office Protocol-Version 3", http://www.ietf.org/rfc/rfc1939.txt

[15]. Kyle J. Nesbit and James E. Smith, "Data Cache Prefetching Using a Global History Buffer", 10th International Symposium on High Performance Computer Architecture (HPCA'04)

[16]. Jehan-François Pâris, Ahmed Amer and Darrell D. E. Long, "A Stochastic Approach to File Access Prediction", *Proceedings of the International Workshop on Storage Network Architecture and Parallel I/O* (SNAPI), New Orleans: IEEE, September 2003

[17]. Postal Benchmark, http://www.coker.com.au/postal/

[18]. Postfix, http://www.postfix.org/

[19]. Qmail, http://infobase.ibase.com.hk/qmail/top.html#tips

[20]. Steven W. Schlosser, Jiri Schindler, Stratos Papadomanolakis, Minglong Shao, Anastassia Ailamaki, Christos Faloutsos and Gregory R. Ganger"On multidimensional data and modern disks", Proceedings of the 4th USENIX Conference on File and Storage Technology (FAST '05). San Francisco, CA. December 13-16, 2005

[21]. Sendmail- http://www.sendmail.org/

[22]. Purvi Shah, Jehan- François Pâris, Ahmed Amer and Darrell D. E. Long, "Identifying Stable File Access Patterns", *Proceedings of the Twenty-first Symposium on Mass Storage Systems* (MSS), Goddard, Maryland: NASA, April 2004

[23]. UW-imap- http://www.washington.edu/imap/

[24]. Jun Wang and Yiming Hu, "PROFS–Performance-Oriented Data Reorganization for Log-structured File System on Multi-Zone Disks", Ninth IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'01)

[25]. David A. Wheeler, "Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers!", http://www.dwheeler.com/contactme.html

[26]. Gary A. S. Whittle, Jehan- François Pâris, Ahmed Amer, Darrell D. E. Long and Randal Burns,"Using Multiple Predictors to Improve the Accuracy of File Access Predictions", *Proceedings of the Twentieth Symposium on Mass Storage Systems* (MSS), San Diego: IEEE, April 2003

[27]. Tsozen Yeh, Darrell D. E. Long and Scott A. Brandt, "Performing File Prediction with a Program-Based Successor Model", *Proceedings of the Ninth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (MASCOTS), Cincinnati: IEEE, August 2001

[28]. Tsozen Yeh, Darrell D. E. Long and Scott A. Brandt, "Using Program and User Information to Improve File Prediction Performance", *Proceedings of the International*

Symposium on Performance Analysis of Systems and Software (ISPASS), Tucson: IEEE, November 2001

[29]. Tsozen Yeh, Darrell D. E. Long and Scott A. Brandt,"Increasing Predictive Accuracy by Prefetching Multiple Program and User Specific Files", *Proceedings of the Sixteenth Annual International Symposium on High Performance Computing Systems and Applications* (HPCS), Moncton, New Brunswick, Canada: IEEE, June 2002

[30]. Tsozen Yeh, Darrell D. E. Long and Scott A. Brandt. "Caching Files with a Program-based Last *n* Successors Model", *Proceedings of the Workshop on Caching, Coherence and Consistency* (WC3), Sorrento, Italy: ACM, June 2001