UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

CAPTURE AND ANALYSIS OF THE NFS WORKLOAD OF AN ISP EMAIL SERVICE

Nuno André Henriques Loureiro

MESTRADO EM SEGURANÇA INFORMÁTICA

Dezembro 2009

UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

CAPTURE AND ANALYSIS OF THE NFS WORKLOAD OF AN ISP EMAIL SERVICE

Nuno André Henriques Loureiro

Orientador
Greg Ganger
Co-Orientador
António Casimiro Ferreira da Costa

MESTRADO EM SEGURANÇA INFORMÁTICA

Dezembro 2009

Resumo

Os objectivos desta tese são capturar a carga de comandos NFS de um serviço de email de um provedor de Internet, converter a captura para um formato mais flexível, e analisar as características do mesmo.

Até ao momento, nenhum outro trabalho publicado, analisou a carga de comandos de um serviço de email de um provedor de Internet. Um novo estudo, irá ajudar a compreender qual o impacto das diferenças na carga de comandos de um sistema de ficheiros de rede, e o que caracteriza a carga de comandos de um sistema de email real. A captura será analisada, de forma a encontrar novas propriedades que futuros sistemas de ficheiros poderão suportar ou explorar.

Nesta tese, fazemos uma análise exaustiva de como capturar altos débitos de tráfego, que envolve vários desafios. Identificamos os problemas encontrados e explicamos como contornar esses problemas.

Devido ao elevado tamanho da captura e devido ao espaço limitado de armazenamento disponível, precisámos de converter a captura para um formato mais compacto e flexível, de forma a podermos fazer uma análise de forma eficiente. Descrevemos os desafios para analisar grandes volumes de dados e quais as técnicas utilizadas. Visto que a captura contém dados sensíveis das caixas de correio dos utilizadores, tivemos que anonimizar a captura. Descrevemos que dados têm de ser anonimizados de forma a disponibilizarmos a captura gratuitamente.

Também analisamos a captura e demonstramos as características únicas da captura estudada, tais como a natureza periódica da actividade do sistema de ficheiros, a distribuição de tamanhos de todos os ficheiros acedidos, a sequencialidade dos dados acedidos e os tipos de anexos mais comuns numa típica caixa de correio.

Palavras-chave: Captura Passiva de Pacotes, Filtragem de Pacotes, Computadores, Armazenamento

Abstract

The aims of this thesis are to capture a real-world NFS workload of an ISP email service, convert the traces to a more useful and flexible format and analyze the characteristics of the workload.

No published work has ever analyzed a large-scale, real-world ISP email workload. A new study will help to understand how these changes impact network file system workloads and what characterizes a real-world email workload. Storage traces are analyzed to find properties that future systems should support or exploit.

In this thesis, we provide an in-depth explanation of how we were able to capture high data rates, which involves several challenges. We identify the bottlenecks faced and explain how we circumvented them.

Due to the large size of the captured workload and limited available storage, we needed to convert the traces to a more compact and flexible format so we could further analyze the workload in an efficient manner. We describe the challenges of analyzing large datasets and the techniques that were used. Since the workload contains sensitive information about the mailboxes, we had to anonymize the workload. We will describe what needed to be anonymized and how it was done. This was an important step to get permission from the ISP to publish the anonymized traces, which will be available for free download.

We also performed several analyses that demonstrate unique characteristics of the studied workload, such as the periodic nature of file system activity, the file size distribution for all accessed files, the sequentiality of accessed data, and the most common type of attachments found in a typical mailbox.

Keywords: Passive packet capture, Packet filtering, Industry NFS traces, storage, Computers

Acknowledgments

I would like to thank several people for their time, suggestions, and influence. Creating this thesis would not have been possible without them.

First, I would like to thank my advisor, Greg Ganger, for giving me the opportunity to create this thesis. His orientation and expertise are definitely valuable inputs that any student would like to have.

My sincere thanks to Matthew Wachs, who worked with me from the beginning and gave me the initial guidance. His guidance allowed me to have an overall overview of the work involved for this thesis and how to achieve what was being proposed.

I would also like to thank my co-advisor, António Casimiro, for his valuable input and guidance throughout the project, particularly for the final part of the project.

My sincere thanks also goes to the ISP email and network teams, specially José Celestino, Carlos Pires and Pedro Mitra, who helped me with everything I needed on the ISP side. Their prompt responses and fast deployment times were extremely helpful.

I would also like to thank my class colleagues for the sharing of experiences, discussion of problems and companionship.

Lisboa, 15th November 2009

I would like to dedicate this thesis to my wife who has been extremely supportive during these last 16 months and to my mother for making me the person I am today. Without the support of the two of you, I would never apply to this program in first place.

Contents

List of Figures

List of Tables

Abbreviations

BPF	Berkeley Packet Filter
CPU	Central processing unit
GiB	Gibibyte
IMAP	Internet Message Access Protocol
ISP	Internet Service Provider
MiB	Mebibytes. 1MiB = 1,048,576 bytes
MUA	Mail User Agent
NFS	Network File System
NIC	Network Interface Card
OS	Operating System
POP	Post Office Protocol
RAID	Redundant Array of Inexpensive Disks
RAM	Random-access memory
RDBM	Relational Database Management System
RPC	Remote Procedure Call
RPM	Revolutions per minute
SAS	Serial Attached SCSI
SMTP	Simple Mail Transfer Protocol

Chapter 1

Introduction

Two decades ago, most filesystem workload analysis was focused on kernel-based trace studies. Some examples include the 'BSD Study' [2], as it became known, or the VAX/VMS study [3], which revealed a number of important observations and trends that guided the design of file systems for over two decades.

However, the workloads of local file systems are very different than those of network file systems. The workloads of local file systems include accesses to many operating system files, whose access patterns are mostly read-only and sequential, and they are focused on the client's point of view. It is evident that having studies done on local file systems is useful and plenty of studies have been conducted. Network file systems are important as well, but only a few studies have explored network file systems in recent years [4][5][6].

If we narrow down the scope of research to NFS traces of email workloads, then we find that the last study is from 2001 [6] and was captured in an academic environment. Since then, there have been significant changes in network bandwidth, computer power, network file systems and storage needs. Furthermore, no published paper has ever analyzed a large-scale, real-world ISP email workload. A new study will help to understand how these changes impact network file system workloads and what characterizes a real-world ISP email workload. Storage traces are analyzed to find properties that future systems should support or exploit.

Furthermore, we know that I/O benchmarking is a widespread practice for comparing the performance of storage systems and serves as the basis for purchasing decisions within the enterprise world. The process of comparing I/O systems is to subject them to known workloads and measure their behavior. If the workload does not reflect the reality of the enterprise needs, then the results of the benchmark can be biased, thus leading to inappropriate

storage acquisitions. Most known workloads are captured in academic settings, therefore commercial workloads are under-represented. Real-world traces will provide ISPs and other companies with the ability to more effectively benchmark new storage solutions.

Email workloads are distinct. Some even propose that a specialized file system for mail service workloads should be designed [6]. If our email platform relies on a mailbox format that stores one message per file, like in the studied ISP, then the main characteristic of the storage is that it is composed of a very large number of small files. While discussing this project with the ISP email team, we were told that several storage vendors have approached them with new storage solutions, but most of those solutions aim at throughput instead of latency and are optimized to a small number of large files instead of a large number of small files. We can understand that those solutions can be suitable to videos or photos services, but definitely not for an email service.

The aim of this thesis was to capture NFS traces of email workloads from a mid-size ISP, convert those traces to a more useful and flexible format, and analyze the results. In doing so, we hope to aid future network file system designs that are suitable for email services with the mentioned characteristics.

We captured real-world NFS traces of an ISP's email workload for a period of 7 days. Passive tracing was used to capture the workload. Passive NFS traces provide a simple and unobtrusive way to measure an NFS workload.

To convert the captured traces, we chose Anderson's tools and the DataSeries format [7]. DataSeries is an on-disk data format, run-time library, and set of tools that is optimized for storing and analyzing structured serial data. Due to the high volume of captured data and storage limitations, we were forced to convert the traces in parallel with the capture.

For the analysis, we used Anderson's analysis tools [5] and we also wrote our own.

Our main contributions include:

1. The first NFS traces of email workloads captured in an ISP environment
2. A comparison with past network file system studies
3. An in-depth explanation on one approach to capturing large workloads and circumventing experimental challenges
4. A new study on file access patterns
5. An analysis of file type patterns found in an ISP email platform

The rest of this thesis is structured as follows. Chapter 2 examines and discusses related work. Chapter 3 provides some background information in order to better understand our work and the traces. Chapter 4 explains the traced systems. Chapter 5 discusses the capture architecture and makes a comparison of the most common capturing tools. Chapter 6 describes the conversion options and what we have used. Chapter 7 gives an in-depth discussion of the capturing and conversion process and describes the challenges that were faced and how to circumvent them. Chapter 8 provides an analysis of the captured workloads. Chapter 9 concludes.

Chapter 2

Related Work

There have been a number of studies done on file system traces, but only a few of them have explored network file system workloads [8][3][9][6][10][4][5], despite their differences from local file systems. Leung et al. summarizes all of these studies [4].

We will focus on four papers of related work, which we consider to be the most relevant ones. [6][10][4][5].

Ellard et al. presented the first and only in-depth analysis of NFS traces of email workloads [6], which, as they state, are quite different from previously studied workloads. However, their captured NFS traces were from a number of servers on the Harvard Campus, i.e. an academic environment. Ellard et al. highlight that they contacted several commercial ISPs in an effort to gather traces from their sites. This proved to be a frustrating task, because the administrators shared interest but declined to provide permission due to privacy concerns. They hoped to convince ISPs to permit tracing, allowing those traces to be shared among the research community, and thereby invigorate contemporary file system design. The NFS trace of email workloads was captured from a collection of machines serving approximately 10,000 active user accounts, each with a default quota of 50Mb for their home directories. Their storage solution was distributed over three NFS servers hosting a total of fourteen 53Gb disk arrays. During periods of heavy activity, they estimate that they experienced 10% packet loss, and, for short bursts, the percentage could have been even higher.

Our NFS traces of email workloads were captured from an ISP's storage solution, hosting 1.2 million active email accounts, each with a default quota of 5Gb. The total storage space used for the email service is over 120Tb. We experienced less than 0.001% packet loss during our tracing period. With our contribution, we hope to help the research community to revive contemporary file system design.

In Ellard's study, each user's mailbox was stored in a single file, thus the workload is dominated by reading and writing of large files, whereas in our study, each email is stored in a separate file, thus our workload is dominated by a large number of small files. As we will show, these results in very different access patterns.

Ellard et al. published another paper [10], where they captured again NFS traces of email workloads. Because the system architecture had been changed, the workloads are very different from the ones they presented in their previous paper. In this later study, the email workload is utterly dominated by read operations (93.39%), which is related to the single file mailboxes and NFS client caching.

Leung et al, traced two large-scale enterprise network file system workloads. They collected CIFS network traces for over three months from two network file servers deployed in NetApp's data center. They used tcpdump [11] to capture the traffic and they collected approximately 2.25Tb of raw-file traces. This is in contrast with the 30Tb of raw-file traces that we collected over a period of seven days. They found increased read-write file access patterns when compared to those previously studied. They also found that read-write ratios had decreased and random file access and file lifetimes had increased.

Anderson's study [5] is probably the most relevant recent study on NFS workloads, and one of the very few that represent commercial workloads. They analyzed a commercial feature animation rendering workload using new techniques and discussed the characteristics of the workload. Their 2007 traces saw about 2.4 billion operations/day, which is of comparable intensity to our traces. They had to develop and adopt new techniques to capture, convert and analyze the traces.

Because of the underlying data rate, they developed and analyzed three techniques for packet capture: lindump, driverdump and endacedump. Lindump is based off of the Linux kernel example of a memory-mapped, shared ring buffer for packet capture. It was able to capture 2x the packets per second as tcpdump. Driverdump was the result of a modification in the network driver so that instead of passing packets up the network stack, it would just copy the packets in pcap format to a file, and immediately return the packet buffer to the NIC. The sustained packets per second was increased over lindump by 2.25x. Endacedump was the solution that they developed in order to dump to disk the captured 8Gbps using an Endace DAG 8.2X capture card. For our traces we evaluated lindump, but we decided to use gulp[12], as explained in chapter 5.

For the conversion, and due to the high volume of captured data, they used their custom binary format, DataSeries [7]. DataSeries is a data format that enables the efficient and flexible storage and analysis of structured serial data, which was developed by HP Labs to pro-

vide six key properties: Storage efficiency; Access efficiency; Flexibility; Self-description; Usability; and Integrity. They compared Ellard's traces in their original format and converted to DataSeries, and found that the analysis distributed with Ellard's traces used 25x less CPU time when the traces and analysis used DataSeries and ran 100x faster on a 4-core machine. For our analysis, we used the DataSeries format, mainly because of storage limitations. A captured raw-file of 400Mb becomes a 5 or 6Mb file when converted to DataSeries.

Analyzing the very large amount of collected data was a challenge that required new techniques. The most important property that they aimed for was bounded memory, i.e., streaming analysis. They also used approximate quantiles in bounded memory, since exact quantiles would be impractical for their data, and data cubes to calculate aggregate or roll-up statistics. Due to our volume of captured data and because we decided to use the DataSeries format, we also used Anderson's tools to analyze our data.

Chapter 3

Background

Before trying to understand the captured NFS workload for an email service, it is important to have some background information on email usage, so that the characteristics of the captured workload can be better understood. It is also important for the reader to have an understanding on how NFS works, since the captured traffic is a NFS workload.

3.1 Email Usage

The way users utilize email is highly related to two factors: their email account's quota and the type of Internet connection they have.

Ten years ago, most of the free Webmail services on the Internet offered an email quota of 5Mb or 10Mb. At that time, the majority of users had a dialup connection to the Internet. These two characteristics dictated how users were using their email accounts. Due to the limited quota, they had to constantly delete old email messages in order to receive new messages. Due to the nature of their Internet connection, they would mainly send small email messages, mostly plain text without any attachments or with small attachments.

When Gmail was first released, Google offered 1Gb of email quota, and the behavior of users began to change. Users were no longer worried about the email quota limitation, thus they stopped cleaning their mailboxes, since there now was plenty of room to store all of their emails. At that time, Gmail did not even provide the functionality for deleting email messages.

When DSL and Cable started to be the common type of Internet connection, users started to send larger files. HTML messages, images, powerpoints and videos became common in email messages, and all this changed how users were using their email accounts.

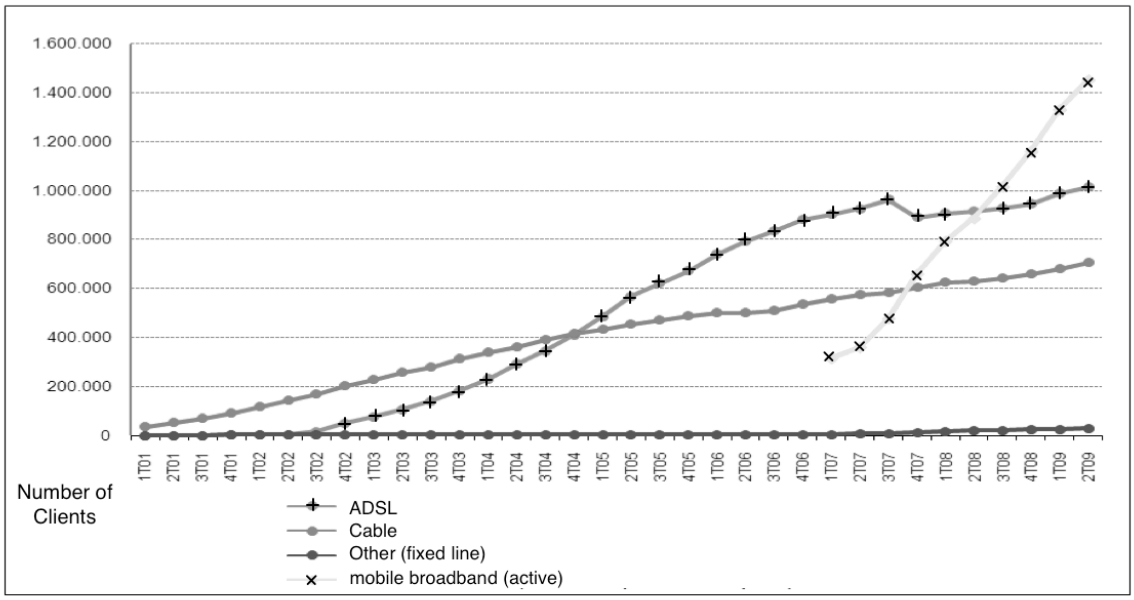


Figure 3.1: Internet Broadband Access in Portugal. Source: Anacom [1]

Today, the typical type of connection to the Internet varies from country to country, or whether we're referring to a developed country or a third world country. For instance, there are several countries in Africa where dialup connection is still the predominant type of connection to the Internet. Because we are tracing NFS traffic for the email service from a Portuguese ISP, it is useful to understand Internet usage in Portugal.

According to Anacom [1], the Portuguese Communications Authority, in the second quarter of 2009, the broadband (Cable/ADSL) penetration in Portugal was 16.5%, the mobile broadband (3G) penetration was 27.7% and the dialup share was 0.4%. The evolution of Internet Broadband access in Portugal throughout the years is depicted in Figure 3.1. Independently of the Internet penetration in Portugal, we can say that broadband is the predominant type of Internet connection.

3.2 Network File System

Sun's NFS protocol [13] provides transparent remote access to shared file systems across networks. The NFS protocol is designed to be machine, operating system, network architecture, and transport protocol independent. This independence is achieved through the use of Remote Procedure Call (RPC) primitives built on top of an eXternal Data Representation (XDR).

Id	Operation	Description
0	NULL	Null
1	GETATTR	retrieves the attributes for a specified file system object
2	SETATTR	changes one or more of the attributes of a file system object on the server
3	LOOKUP	searches a directory for a specific name and returns the file handle for the corresponding fs obj
4	ACCESS	determines the access rights that a user has with respect to a fs obj
5	READLINK	reads the data associated with a symbolic link
6	READ	reads data from a file
7	WRITE	writes data to a file
8	CREATE	creates a regular file
9	MKDIR	creates a new subdirectory
10	SYMLINK	creates a new symbolic link
11	MKNOD	creates a new special file of the type, what.type
12	REMOVE	removes (deletes) an entry from a directory
13	RMDIR	removes (deletes) a subdirectory from a directory
14	RENAME	renames the file identified by from.name in the directory, from.dir, to to.name in the directory, to.dir
15	LINK	creates a hard link from file to link.name, in the directory, link.dir
16	READDIR	retrieves a variable number of entries, in sequence, from a directory and returns the name and file identifier for each
17	READDIRPLUS	retrieves a variable number of entries from a file system directory and returns complete information about each
18	FSSTAT	retrieves volatile file system state information
19	FSINFO	retrieves nonvolatile file system state information
20	PATHCONF	retrieves the pathconf information for a file or directory

Table 3.1: NFS version 3 Primitives

The supporting MOUNT protocol performs the operating system-specific functions that allow clients to attach remote directory trees to a point within the local file system. The mount process also allows the server to grant remote access privileges to a restricted set of clients via export control.

The RPC primitives available for NFS version 3 are depicted in table 3.1.

Chapter 4

Traced Systems

For this thesis, we traced the NFS Workload for the Email service of a Portuguese ISP for a period of seven days. Table 4.1 represents which email services were traced and what percentage of the total they represent. For example, the POP3 farm is composed of 5 servers and we traced the NFS traffic of one of them.

Service	Percentage of Total
SMTPi (incoming)	9%
SMTPo (outgoing)	25%
POP3	20%
IMAP	100%

Table 4.1: Traced Systems

The studied ISP relies on Netapp filers to store over a hundred terabytes of information from its various services and on NFSv3 over TCP as the underlying communication protocol to access the storage. Email, Videos and Photos are the predominant services that use this storage solution. The business models for the specified services can be compared to the ones of Yahoo! for email service, YouTube for videos service or Flickr for photos service. Note that email service is responsible for more than 90% of the used storage in the studied ISP.

4.1 Storage Network Appliances

The storage solution is divided into four nodes: Fasnode1, Fasnode2, Fasnode3 and Fasnode4. Fasnode1 and Fasnode2 are Netapp FAS980 Network appliances and Fasnode3 and Fasnode4

are Netapp FAS3040. More recently, they borrowed two additional Netapp filers from another department.

It is important to understand that the capture was not performed in a testbed environment. It was from real-world services in a production environment. We only captured NFS traffic for email service, but the storage nodes also serve other services. If, for example, one storage node has a higher load because of videos service, it will have an impact on the quality of the service provided to the email services, such as the number of operations served within a period of time, or latency, or response time.

The email service uses all of the fasnodes (Fasnode1-Fasnode4). Each storage node has a gigabit link for the email service. Fasnode2 also stores the web pages for the ISP Portal and its basic services. Video and photos services only use Fasnode3 and Fasnode4. The email service also uses two additional storage nodes that are being administrated by another department.

These six storage nodes, handle over 2.4 billion NFS operations per day.

4.2 Email Platform

There are 1.2 million active accounts (within a 90-day period) from a total of 7 million accounts. Storage has over 120Tb of used space, and the storage growth in the last six months was on average 4.6Tb per month.

The email platform is comprised of several service-dedicated farms. For instance, there is a SMTP farm, an IMAP farm, a POP3 farm and a management farm, among others.

The Email platform uses QMail software[14] to provide SMTP and POP3 services and Dovecot[15] to provide the IMAP service. It is important to note that the captured workload represents the NFS operations for an ISP email service based on both QMail and Dovecot. If the ISP was running different software, the results could be significantly different. For example, Dovecot IMAP uses indexes to avoid reading all files from a directory in order to build a message list. Courier IMAP [16] on the other hand, has to read the headers of all messages (files) to build the same message list. If one customer is using Webmail and has 20 email messages in his/hers Inbox folder, Dovecot would need to read only one file (the index), while Courier IMAP would need to read 20 files (to read the email headers of each message).

Table 4.2 provides the number of servers per relevant service, and the typical write and read operations for each service. For instance, the predominant operations for the SMTP service

are write operations (to store the email messages). On the other hand, the predominant operations for the POP3 services are read operations (to read the email messages).

Hosts	Type	Typical Writes	Typical Reads
MTA1-MTA11 (11 servers)	SMTP Incoming (MX)	- QMail/Dovecot indexes & control files - email messages	- QMail/Dovecot indexes & control files
MTA12-MTA15 (4 servers)	SMTP Outgoing (Auth)	- QMail/Dovecot indexes & control files - email messages	- QMail/Dovecot indexes & control files
POP2-POP6 (5 servers)	POP	- QMail/Dovecot indexes & control files	- email messages
IMAP01-IMAP09 (9 servers)	IMAP	- QMail/Dovecot indexes & control files - email messages	- QMail/Dovecot indexes & control files - email messages

Table 4.2: Email servers

Each email account has a personal mailbox, and the mailbox format is Maildir++ [17][18]. One relevant property of Maildir is that each email message is stored in a single file. This dictates the characteristics of the studied storage. Because the average email message is small and each email message is stored in one file, the majority of the files in storage are small and the total number of files is high.

Chapter 5

Capture

Capturing high data rates involves several challenges. The first challenge is the capturing architecture, i.e., where the capture is going to be performed and if the involved hardware can handle the capture at the underlying data rate. It is crucial to choose the right software tool to do the job, since many of the existing tools perform quite differently. Another challenge is disk space to store the high volume of workload. In this chapter and in the next chapter, we will go through each one of these challenges in more detail.

5.1 Architecture

When faced with the desire to capture NFS workloads, the first question considered was where the capturing should be done. We could capture the workloads on the client side (mail servers), on the server side (storage nodes), or in the network (through port mirroring).

The storage nodes run proprietary software, thus capturing the workloads directly on the storage nodes was not possible. Even if it was a possibility, it is not difficult to understand that it should have an impact on the servers' performance. On the other hand, capturing the workload through port mirroring is like capturing it on the server side but without significant side effects, given that most switches implement port mirroring in their hardware. Capturing the workload on the client side, besides having an impact on the clients' performance, would also require modifications to the production email servers.

Despite the disadvantages mentioned above, the first capturing attempts were performed directly in production servers (on the client). We chose this approach since it would allow us to start working immediately and would not require any extra resources from the ISP.

We were also not sure if the impact of the capture on the server would be significant or not. A server from each type of service was selected and those systems were traced. After performing some OS tuning and some captures, we realized that packet loss was high and that it had a significant impact on the services, so we approached the ISP for alternative solutions.

After a brief discussion, we concluded that the best approach would be to do port mirroring of the storage nodes to a dedicated server. The ISP provided us with an HP Proliant DL360 G5 server (two quad-core), with 8Gb of RAM and two 72Gb SAS 15k rpm disks in RAID 1 for the root partition (Operating System) and four 72Gb SAS 15k rpm in RAID 1 + 0 (mirror + stripping) for the capture partition.

Due to architectural limitations on the ISP side, it was not possible to do the port mirroring on the network switch where the storage nodes were connected. Instead, we did the port mirroring in the neighbor switches, which aggregate the mail traffic to the storage nodes, as depicted in Figure 5.1. The switches with mail traffic were switches 1, 2, 5 and 6.

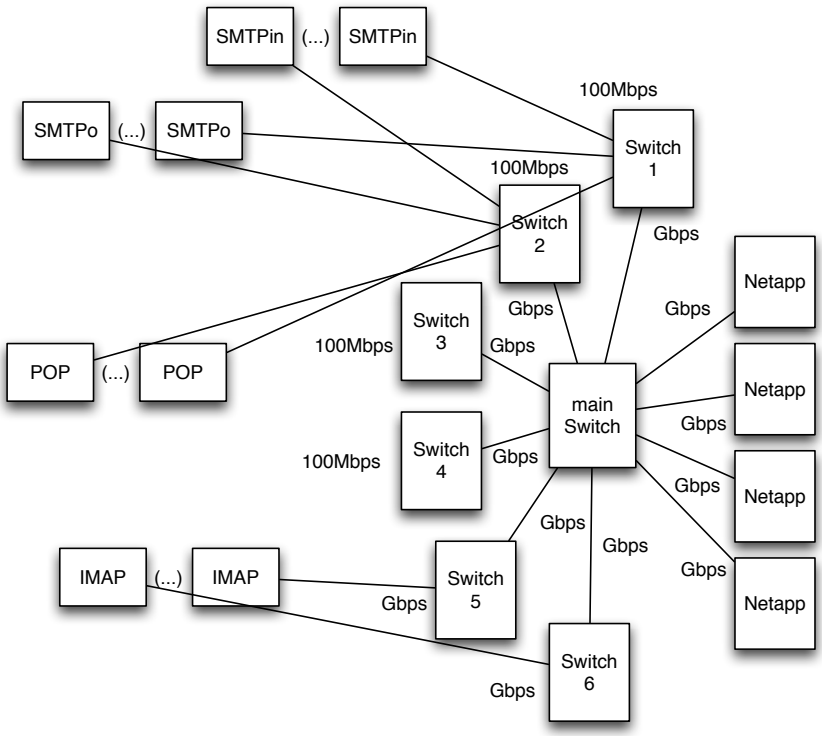
Another problem was that switches 1 and 2 only had downlink ports of 100Mbps (which connect to the mail servers) and one uplink port of 1Gbps (which connects to the main switch). But the aggregate bandwidth of each switch was over 100Mbps, thus we would miss some traffic. On the other hand, switches 5 and 6 are gigabit switches, so it would be feasible to capture the aggregate of the traffic. Each one of these switches has an average traffic of 400Mbps. As it can be seen in Figure 5.1, SMTP and POP3 servers are connected to switches 1 and 2, and the IMAP servers are connected to switches 5 and 6.

After analyzing various options, we decided to capture the entire IMAP traffic, thus port mirroring the aggregate traffic of switches 5 and 6. As for the other services, ideally we would like to capture the whole traffic, but due the limitations mentioned above, we moved one server of each type (SMTPi, SMTPo and POP3) to switch 4 and did port mirroring of the aggregate traffic. For this scenario, needed three network cards in our dedicated server to listen to the three mirrored ports.

After properly setting up the system, we had to decide which software to use for the capturing.

5.2 Software tools

A typical capture setup is depicted in Figure 5.2. The Network Interface Card (NIC) signals the availability of new data in the NIC driver through an interrupt request. The capture



Switch1-4: N*100Mbps downlink ports (servers) and 1Gbps uplink port (main switch)
Switch5-6: M*1Gbps downlink ports (servers) and 1Gbps uplink port (main switch)

Figure 5.1: Simplification of the Network layout

driver, at kernel-level, starts the custom processing, like filtering. If the filtering rules are matched, the packets are copied to another system memory location (called kernel buffer), from where they are delivered to the application that can perform further processing.

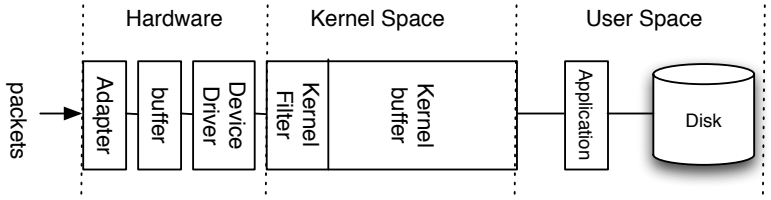


Figure 5.2: Capture setup

Packet loss[1] can occur due to insufficient performance of the network cards and drivers, lack of enough kernel memory, CPU shortage and ultimately due to the overhead introduced by saving packets into hard disk, etc. Since parsing NFS packets requires capturing the complete packet, it is important that we choose a tool that can either implement mecha-

[1]Packet loss, in this context, is the percentage of packets received by the NIC that are not delivered to the user-space application.

nisms that prevent packet loss or is flexible enough when it interacts with other components responsible for packet loss.

From the available tools for capturing traffic, we narrowed our options to the following ones:

- tcpdump
- tshark
- lindump
- gulp

We tested these four tools on the capturing server, capturing the traffic of one of the three network cards in port mirroring. After coming up with a winner, we then further tested it by capturing the traffic of the three network cards in parallel.

5.2.1 Tcpdump/libpcap

A lot of capturing software tools are based on libpcap [11], a common system-independent interface for user-level packet capture. Tcpdump [11] is the most famous tool using libpcap for packet capture, thus it was the first option to be evaluated.

After some captures, we realized that the packet loss was high, thus we tried to tune some OS parameters, like */proc/sys/net/core/rmem_default* and */proc/sys/net/core/rmem_max*, which showed improvements. However, it was not sufficient, we still had a significant rate of packet loss. Some studies also showed that tcpdump is not suitable for high data rates, like 1Gbps and more [4][5][19][20]. We believe tcpdump's inefficiency is due to its single-thread model, i.e., it uses the same thread for capturing, filtering and outputting packets (either to disk or standard output).

Tcpdump was run as in the following example:

nice –10 tcpdump -w foo.cap -s 0 -n -i eth5 -C 200

The parameter *-w* means that the capture should be written to file foo.cap. The parameter *-s 0* means that it should capture the entire packet. The parameter *-n* means that tcpdump should not try to convert addresses (i.e., host addresses, port numbers, etc.) to names. The parameter *-i eth5* means that we are capturing traffic from the interface *eth5*. The parameter *-C 200* means that the capture file size can be at most 200Mb and, when it reaches that limit, it should start writing to a new file.

5.2.2 Tshark

Tshark[21] is a command line tool and is part of Wireshark, a tcpdump-like tool with a graphic front-end and many more information sorting and filtering options. Tshark/Wireshark, formerly known as ethereal, is also based on libpcap like tcpdump but there is a big difference between tshark and tcpdump in terms of performance.

Tshark follows a multi-threaded model. It uses two different threads, one for capturing data from the network and one for processing it. Thus, on multi-processor servers, it is easier to achieve better performance than when compared to tcpdump.

We tried tshark, and, even though the results were better than with tcpdump, we still had significant packet loss, so we tried other options.

Tshark was run as in the following example:

> *nice –10 tshark -w foo.cap -s 65536 -n -i eth5 -b filesize:200000 -f "tcp port 2049"*

The parameter *-w foo.cap* means that the capture should be written to file foo.cap. The parameter *-S 65536* means that no more than the first 65536 bytes of a packet should be captured (i.e. the entire packet should be captured). The parameter *-n* means that tcpdump should not try to convert addresses (i.e., host addresses, port numbers, etc.) to names. The parameter *-i eth5* means that we are capturing traffic from the interface *eth5*. The parameter *-b filesize:200000* means that the capture file size can be at most 200MiB and when it reaches that limit, it should start writing to a new file. The parameter *-f "tcp port 2049"* means that we are applying a capture filter that will only capture TCP traffic on port 2049.

5.2.3 Lindump

Due to the inefficiency of tcpdump for high data rates, HP Labs developed Lindump[5]. Lindump is based on an example program that comes with the Linux kernel, that implements a memory-mapped, shared finger buffer for packet capture. HP Labs changed it to write out pcap files [22] and to be able to capture from more than one interface at the same time. They claim that lindump was able to capture about 2x the packets per second as tcpdump and about 1.25x the bandwidth.

We tried lindump, and we had satisfactory results. The only problem with lindump is that it does not accept filters, and, due to the architecture of our system, we needed to use filters.

The capturing server is also connected to the storage network, and we need to access the storage nodes to store the workload files when the local disk is almost full. Thus, we need to use filters to exclude the IP address of the network interface that is connected to the storage network.

We considered changing lindump to accept filters, but even though lindump uses pcap data structures, it does not use libpcap for the capture itself. Thus it would require a significant effort to implement filters. If it used libpcap functions for the capture, it would be rather easy to implement filters, through the *pcap_setfilter()* function.

Lindump was run as in the following example:

nice –10 lindump-mmap eth5 testtrace

The first parameter is the interface(s) that we are capturing traffic from (*eth5*). The last parameter is the prefix for the capture files.

5.2.4 Gulp

Gulp [12], was developed by Corey Satten from the University of Washington Network Systems. The author's motivation was also the inefficiency of tcpdump under high data rates, and his goal was to achieve lossless gigabit packet capture to disk with unmodified Linux on ordinary/modest PC hardware.

Satten realized that his system showed plenty of idle resources when packets were dropped, and writing packets to disk seemed to have a disproportionate impact on packet loss, especially when the system buffer cache was full. It eventually occurred to him to try to decouple disk writing from packet reading, and that was how gulp was born.

Gulp is a simple multi-threaded ring-buffer packet capture program designed to be completely lock-free. The multi-threaded ring buffer worked remarkably well and considerably increased the rate at which he could capture without loss. However, at higher packet rates, it still dropped packets, especially while writing to disk. The problem was that the Linux scheduler sometimes scheduled both the reader and writer threads on the same CPU/core, which caused them to run alternately instead of simultaneously. When they ran alternately, the packet reader was again starved of CPU cycles and packet loss occurred. The solution was simply to explicitly assign the reader and writer threads to different CPU/cores and to increase the scheduling priority of the packet reading thread. These two changes improved performance so dramatically that dropping any packets on a gigabit capture, written entirely to disk, became a rare occurrence.

Furthermore, he noticed that when a system has more than two cores, the L2 cache is only shared in pairs. As an example, in our two quad-core system, core 0 shares L2 cache with core 4, the same way that the pairs of cores 1 and 5, 2 and 6, and 3 and 7 share L2 cache. This means that we should have the reader in one core and the writer in the other core that shares the L2 cache with the reader, so that the access to the ring buffer is faster.

We tried gulp and the results were satisfactory, we were able to capture with a very small percentage (<1%) of packet loss. Furthermore, gulp uses libpcap functions and implements filters, so we used gulp for the capture.

Gulp was run as in the following example:

nice –10 gulp -i eth5 -r 200 -V xxxxxxxxxx -o gulp_cap -C 5 -f "tcp port 2049"

The parameter *-i eth5* means that we are capturing traffic from the interface *eth5*. The parameter *-r 200* specifies that the size of the ring buffer is 200Mb. The parameter *-V xxxxxxxxxx* defines an argument string that will be overwritten twice per second with a brief capture status update (statistics). The parameter *-o gulp_cap* specifies the directory where the captured files will be stored. The parameter *-C 5* means that gulp will start a new pcap file when the old one reaches about 5 times the size of the ring buffer. The parameter *-f "tcp port 2049"* means that we are applying a capture filter that will only capture TCP traffic on port 2049.

Chapter 6

Conversion

For this project, we could not decouple the capture process from the conversion process, since both would have to be run in parallel due to storage limitations.

In our preliminary tests, we estimated that we would capture around 4Tb of raw data (pcap format) per day, and we only had 144Gb of disk space in our capture server. Thus, it was necessary to convert the original pcap format to something substantially smaller or to use gzip or similar to compress the files. A quick test using gzip and bzip2 showed that, with gzip, we could compress a pcap file by 60% and with bzip2 by 65%. That would be around 1.6Tb per day or 11.2Tb a week, which is still well over our capacity.

Besides the disk space limitations, we also needed to convert the data to an easily usable format, in order to facilitate the analysis. Furthermore, the raw packet format contains a substantial amount of unnecessary data and would require expensive parsing to be used for NFS analysis.

We have looked into several formats for data representation and storage like Ellard's and Anderson's formats [10][7].

Ellard et al. [10] used a text format to represent the captured data: one line per request/reply with several fields to identify the different parameters in the RPC. The problem with a text format is that it is too large, too slow to parse, and does not accommodate properly those responses that have a different number of fields. Therefore, a more relational structured data format would be desired.

We also pondered using SQL, which would definitely improve flexibility, but most RDBMS, lack extensive compression, and we lack storage space to store our traces. It is also questionable whether an RDBMS would perform reasonably under more complicated SQL queries that would traverse the billions of records in the database.

Anderson's DataSeries format [7] meets the requirements that we were looking for. Therefore, we chose this format.

DataSeries is an efficient and compact format for storing traces. It uses a relational data model, so there are rows of data, with each row comprised of the same typed columns. A column can be nullable. Groups of rows are compressed as a unit. Prior to compression, various transforms are applied to reduce the size of the data. DataSeries is designed for efficient access. Values are packed so that once a group of rows is read in, an analysis can iterate over them simply by increasing a single counter. Efficient access to subsets of the data is supported by an automatically generated index. Furthermore, DataSeries is also designed for integrity. It has internal checksums on both the compressed and the uncompressed data to validate that the data has been processed appropriately. Additional details on the format, transforms and comparisons to a wide variety of alternatives can be found in the DataSeries technical report [23].

Some other important reasons to choose DataSeries were:

- The familiarity and past experience that some members of the Parallel Data Lab (PDL) had with this format
- It was already used in an equivalent study in terms of the volume of data [5]
- It was referred as the preferred trace format in the first annual file and storage systems benchmarking workshop [24]
- It can do trace anonymization

While evaluating DataSeries, we realized that a converted file could be as much as 100x less the size of the original raw-capture file. Our 400Mb raw-captured files were converted to 5Mb files in only 6 to 7 seconds on our servers. We were able to convert the raw-captured files faster than the capture itself.

The only problem we experienced with DataSeries was that it can be complex for getting some information it was not designed for. For such cases, we dumped the DataSeries extent with the information we wanted to parse to a text format, and we piped the information into a script for further analysis.

6.1 Trace Anonymization

In order to release the traces, the ISP required anonymization of some parts of the traces, such as filenames and paths.

Maildir [17] or Maildir++ [18] formats disclose a lot of information within the file names.

Every delivery to a maildir must have its own unique name. When a maildir is shared through NFS, every machine that delivers to the maildir must have its own hostname. Within one machine, every delivery within the same second must have a different delivery identifier. For that purpose, each filename includes a timestamp of its creation and which server received the email message.

In order to improve performance when a Mail User Agent (MUA) lists a mailbox, and in order to avoid extra I/O operations, filenames also include some information which is equivalent to the status field used by mailbox readers. The list of information semantics is as follows:

- Flag "P" (passed): the user has resent/forwarded/bounced this message to someone else.
- Flag "R" (replied): the user has replied to this message.
- Flag "S" (seen): the user has viewed this message, though perhaps he didn't read all the way through it.
- Flag "T" (trashed): the user has moved this message to the trash; the trash will be emptied by a later user action.
- Flag "D" (draft): the user considers this message a draft; toggled at user discretion.
- Flag "F" (flagged): user-defined flag; toggled at user discretion.

Furthermore, and due to Maildir++ extensions, the filename also includes the size of the message in the following format:

- **,S=<size>:** <size> contains the file size. Getting the size from the filename avoids doing a stat(), which may improve the performance. This is especially useful with Maildir++ quota.

- **,W=<vsize>:** <vsize> contains the file's RFC822.SIZE, ie. the file size with linefeeds being CR+LF characters. If the message was stored with CR+LF linefeeds, <size> and <vsize> are the same. Setting this may give a small speedup because now Dovecot does not need to calculate the size itself.

An example of a maildir filename can look like this:

1250782884.M696935P29266.mta3,S=5971,W=6171:2,RS

Not only does the maildir format disclose a lot of information, but it also includes the mailboxes path names and the email address of the users. Therefore, we must anonymize not only the file names but also the path names.

One bonus that we gained by choosing DataSeries was that it already supports trace anonymization. DataSeries uses encrypted values since it preserves maximum flexibility, and can be reversed, i.e., it can be converted back to real filenames by decrypting the encrypted values.

Chapter 7

Capture and Conversion Process

As previously mentioned in section 5.1, we had a dedicated server for the capture, an HP Proliant DL360 G5 server (two quad-core), with 8Gb of RAM and two 72Gb SAS 15k rpm disks in RAID 1 for the root partition (Operating System) and four 72Gb SAS 15k rpm in RAID 1 + 0 (mirror + stripping) for the capture files. This server was equipped with six network cards: one that connects to the management network, one that connects to the storage network, three for the port mirroring and one that was unused. The server was running Debian GNU/Linux 5.0.2 and Linux Kernel 2.6.26-1-amd64.

When we tested gulp to capture the traffic of the three network cards, the results remained good and the percentage of packet loss was still notorious. However, we faced several problems and challenges that we describe in the next section.

7.1 Faced Problems and Challenges

Due to storage limitations, we were forced to convert the raw capture files at the same time we were capturing traffic. When we did it, we experienced a significant percentage of packet loss. While analyzing the problem, we realized that we were having I/O problems. Running "*iostat -x 1*" showed a constant 100% utilization [1] for the disk where the capture is stored (c0d1).

The capture involves three simultaneous files that are being written to the disk (pcap files), and in parallel we are converting them to the dataseries format. The rate of data written to

[1] Percentage of CPU time during which I/O requests were issued to the device or bandwidth utilization

```
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          47.67    0.00    7.60   44.49    0.00    0.25

Device:         rrqm/s   wrqm/s     r/s     w/s    rsec/s    wsec/s avgrq-sz avgqu-sz   await  svctm  %util
cciss/c0d0        0.00    46.00    1.00   36.00      8.00    656.00    17.95     0.00    0.11   0.11   0.40
cciss/c0d0p1      0.00     0.00    0.00    0.00      0.00      0.00     0.00     0.00    0.00   0.00   0.00
cciss/c0d0p2      0.00    46.00    1.00   36.00      8.00    656.00    17.95     0.00    0.11   0.11   0.40
cciss/c0d1       34.00    11.00  171.00  554.00  20744.00 173536.00   267.97   153.68  133.48   1.38 100.00
cciss/c0d1p1     34.00    11.00  171.00  554.00  20744.00 173536.00   267.97   153.68  133.48   1.38 100.00
```

Figure 7.1: Output of the command: *iostat -x 1*

disk per minute during busy hours was approximately 4Gb and in parallel those 4Gb were being read for conversion.

The first step was to convert the RAID type. By converting RAID 0 + 1 to RAID 0, the gain was not only performance but also more disk space. We increased the bandwidth from 169 MB/s to 258 MB/s for sequential writes [2], and the disk space from 144Gb to 288Gb. Even though the capture results were better, it wasn't enough. The disk bandwidth utilization was still at 100%.

The obvious solution to the problem would be to not convert the raw capture files in parallel, to just do the conversion during idle time. The volume of traffic is high during the entire day, therefore we would need enough storage to hold at least 14 hours of traffic. We captured 14h of traffic and the result was 2.1Tb of raw capture files (with 20% packet loss), and since we did not have that much space on local drives, it was crucial that the conversion was done in parallel so that the original raw files could be deleted.

We then tried to use an in-memory filesystem to store the captured raw files, and use dataseries tools to convert the files to disk. The preliminary results were very good, but the 8Gb of RAM was not sufficient, since 8Gb is the average size for two minutes of captured data. We asked the ISP to increase the memory of our server and they upgraded it to 16Gb of RAM. They also changed the disks to store the capture files, to four 146Gb SAS 10k rpm, also in RAID 0.

We started the capture again, hoping that it would be the final run. But after a few hours, during peak hours, we realized that the capturing rate was faster than the conversion rate and all the six cores used for conversion were 0% idle. Eventually we ran out of memory so we asked the ISP for another server. The ISP provided the requested server, which was similar to the first one.

We connected the network cable used for the port mirroring of switch 5, which was the interface with the most traffic, to the new server, and left the two other ports connected to the first server.

[2] Measured using the "*dd*" utility to write sequentially to the disk

Server/Spec	CaptureSrv1	CaptureSrv2
Brand/Model	HP Proliant DL360G5	HP Proliant DL360G5
Processor	2 Quad-Core Intel Xeon E5345 at 2.33GHz	2 Quad-Core Intel Xeon E5345 at 2.33GHz
L2 Cache	8,192 KB	12,288 KB
Memory	16Gb (8x2Gb)	16Gb (8x2Gb)
Network Cards	6x Gigabit Ethernet cards	6x Gigabit Ethernet cards
RAID/Disks	2x 72Gb SAS 15k rpm in RAID 1 (root) 4x 146Gb SAS 10k rpm in RAID 0 (disk2)	2x 72Gb SAS 15k rpm in RAID 1 (root) 2x 146Gb SAS 10k rpm in RAID 0 (disk2)

Table 7.1: Specifications of the capture servers

Finally, we were able to capture the traffic with no packet loss or with a residual percentage of packet loss.

7.2 Gulp setup

After we downloaded and compiled gulp, we realized that the dataseries conversion tool (*nettrace2ds*) was unable to read raw captured files (pcap). The reason was that in libpcap, *struct pcap_pkthdr*, depends upon *sizeof(long)*. This makes pcap files from 64-bit linux systems incompatible with those from 32-bit systems. As a workaround to this problem, since our linux distribution was 64-bit, we compiled libpcap and gulp as a 32-bit version (using the compiler flag *-m32*).

As mentioned above we increased the default size of Linux's receive socket buffers to 4Mb, by issuing the following commands:

- *echo "4194304" > /proc/sys/net/core/rmem_max*
- *echo "4194304" >/proc/sys/net/core/rmem_default*

Next, we downloaded and ran Satten's microbenchmark [12], to find out which cores share L2 cache. We concluded that core 0 shares L2 cache with core 4, the same way that the pairs of cores 1 and 5, 2 and 6, and 3 and 7 share L2 cache.

We changed gulp.c to define the CPU affinity for the reader and writer processes, so that the cores are one of the pairs that share the L2 cache mentioned above. The affinity is set through two defines, *READER_CPU* and *WRITER_CPU*.

After setting the affinity of the *READER_CPU*, it was important to set the CPU affinity for handling the IRQs of the network card being monitored, i.e., if we are capturing traffic on

the network card eth0 and set *READER_CPU* to 4, then we need to set the interrupt handler for *eth0* to be executed on core 4. This can be done, by checking the IRQ for the device in */proc/interrupts*, and by setting the core in the *smp_affinity* of the IRQ in question. An example of the relevant parts of */proc/interrupts* is depicted in figure 7.2.

```
           CPU0       CPU1       CPU2       CPU3       CPU4       CPU5       CPU6       CPU7
1263: 1537208393         51         48         47 1537208799         57         56         56 PCI-MSI-edge eth0
1265:     275258   12574355   12595340   12592123     274646   12585099   12600577   12583500 PCI-MSI-edge eth5
1266:     102289    7222017    7197224    7204026     102352    7210345    7193755    7212458 PCI-MSI-edge eth4
1267:        585      25608      25962      25810        549      25627      25594      25791 PCI-MSI-edge eth3
1268:        524      22616      23450      22840        450      22868      22830      22725 PCI-MSI-edge eth2
```

Figure 7.2: Contents of */proc/interrupts*

From Figure 7.2 one can see that the IRQ for eth0 is *1263*. The CPU affinity is represented as a bitmask, with the lowest order bit corresponding to the first logical CPU and the highest order bit corresponding to the last logical CPU. Not all CPUs may exist on a given system, but a mask may specify more CPUs than are present. So, for example, the bitmask that represents core 0 and 4 is 0001 0001 or 0x11.

Now, we need to set this value in IRQ 1263's *smp_affinity* by executing the following command:

echo 11> /proc/irq/1263/smp_affinity

From now on, for every packet that arrives on eth0, an IRQ will be issued on cores 0 or 4. This explains the fact that, in Figure 7.2, the counter for CPU0 and CPU4 is in the order of billions of packets, while for every other CPU is below 100.

The last change that we made to gulp was to enhance the capturing performance when using filters. Fortunately, libpcap provides an interface to set a filter as a Berkeley Packet Filter (BPF) program, so one does not need to do the filtering into his/her user-space program. To do this, one needs to pass the filter as a string to *pcap_compile()* and then set it using *pcap_setfilter()*. The function *pcap_compile()* takes as the third parameter an optimization flag, which can be on or off. Gulp by default had it off, so we set it on.

As mentioned before, we used an in-memory filesystem to store the captured raw files. To create this ramdisk, we used the following command:

mount -t tmpfs -o size=14g tmpfs /mnt/tmp

Before we started the 7-day capture, we created a few scripts to automatize the process. The script that starts the capture sets the CPU affinity automatically, according to the NIC that we are capturing traffic from. If the NIC is the one that is capturing traffic from switch 5, then we also add a filter to not include the traffic from/to the capture hosts. This is because the two capture servers are connected to the storage network through switch 5, and we do not want to capture traffic from these hosts.

7.3 Conversion at Capture Time

In order to convert the raw capture files into DataSeries files while the capture is running, we wrote a script that is always running and checking for raw capture files with unchanged status information in the last 4 seconds. When such a file is found, it is converted into the dataseries format. The way we came up with the 4 seconds rule was through trial and error. We wanted it to be the smallest number possible, so that the files could be converted as soon as possible and deleted from the ramdisk.

If there are more than one unchanged file in the past 4 seconds, then those files are converted with only one call of *nettrace2ds,* since it is more efficient to convert three or four files with one call of *nettrace2ds* than running *nettrace2ds* three or four times, one per each raw file.

Since our ramdisk only has 14Gb of size, if its used space is over 80%, the files are moved to the local hard drive first and then converted. This is another step to guarantee that the ramdisk never fills up, because if it does, the capture process will be aborted and then we would have to start over.

Moreover, during relatively idle time (typically at night), we moved the converted DataSeries files to one of the NetApp volumes, in order not to fill up the local hard drive.

One last but important detail is that we do not want the conversion program nettrace2ds to run on the same cores used by the capture process. Thus, it is important to set the affinity for nettrace2ds. This can be done using the linux command taskset, which can be invoked like in the following example:

> *taskset -c 1-3,5-7 nettrace2ds –convert –pcap 0 999999999 eth0_ds/trace_1000.ds gulp_eth0/pcap17864 gulp_eth0/pcap17865 gulp_eth0/pcap17866*

The parameter *-c* in the example above defines which cores can be used for *nettrace2ds*. In this example, all the cores except 0 and 4 can be used.

After the 7-day capture, we had 93743 DataSeries files. To reduce this number, we repacked the DataSeries files into one file per capture-hour per NIC, using the command *dsrepack*. When the repacking was completed, we had a total of 507 DataSeries files.

Chapter 8

Analysis

The next step, after having the raw-files converted to the DataSeries format, was to start our analysis. Thanks to DataSeries, our traces were using 490Gb of disk space, instead of 28Tb of raw-files. Nevertheless, 490Gb of traces is still very large and analyzing this amount of data can take a long time. Even with the analysis efficiency enabled by DataSeries, some analyses took up to 10 hours of processing in our two quad-core server.

In order to analyze huge amounts of data, the most important limitation is bounded memory, which means that we need to analyze the data as a stream, where our analysis is done while reading the stream. When reading a stream, we cannot go back and forth on the stream, thus, we might need to rethink some analysis algorithms in order to work with streams. The second property that we need is efficiency, so we can analyze the data in a reasonable amount of time.

DataSeries analysis tools use approximate quantiles, since it would be infeasible to implement exact quantiles in bounded memory. For calculating aggregate or roll-up statistics, the authors implemented data cubes, which is a generalization of the group-by operation commonly used in SQL databases. They also wrote a HashTable implementation and rotating hash-maps in order to achieve better performance and reduce memory usage. More information on implementation of these techniques used by DataSeries tools can be found in Anderson's et al. paper [5].

For most of our analysis we used the *nfsdsanalysis* tool that comes with DataSeries. We also wrote some tools to parse the DataSeries files and provide some statistics that could not be done with *nfsdsanalysis* directly. Once we had summarized the data from DataSeries using the techniques described above, we inserted the data into a standard SQL database. This enabled us to combine the necessary data to plot the desired graphs.

For plotting the graphs we used gnuplot [25]. Instead of plotting the graphs manually in gnuplot, we wrote some tools that read from the SQL database and write gnuplot scripts and data files, in order to automatize the plotting process. This will allow us to reuse the tools and plot new graphs automatically on future traces.

In the next sections of this chapter, we provide some analysis of our captured NFS workload. We provide some basic NFS analysis, such as a summary of daily and hourly activity, read/write ratios and operation and bandwidth rates. We also analyze the accessed file sizes and the sequentiality. We also analyzed the types of email attachments that can be found on the ISP's storage volumes.

Recall from Figure 4.1 that we did not capture all the NFS traffic. Therefore, we extrapolated the results to match the real size of the email platform, i.e., to reflect the load that the entire email platform puts on the storage servers. To get the original numbers, one just needs to multiply all the values presented in this section with the corresponding percentage in Table 4.1.

The email service is composed of a set of sub-services, such as STMP for receiving email, POP and IMAP to read email. We believe that these services present different properties in terms of NFS workload, so we analyzed them separately.

8.1 Basic NFS Analysis

The first basic NFS analysis that we have performed was a summary of the NFS workload activity. It is important to know the size of our workload in terms of NFS operations. Table 8.1 shows a summary of the average daily activity during trace periods. We also compare our results with the same statistics of past studies, namely, Ellard's NFS traces of email workload. The other studies are Baker and Roselli's studies [8][26]. Note that RES, INS, and NT traces are kernel-level traces of local filesystem, and do not show the effect of client-side caching. The Sprite traces, on the other hand, use a different form of client-side caching than NFS.

When we analyze Table 8.1, we can see that our workload is at least one order of magnitude busier than any of the other systems.

One thing that we noticed, and did not expect, was the fact that the number of write operations is less than the number of read operations for the SMTP service. We expected the SMTP activity to be write and not read-heavy, since its main purpose is to write the email that it receives to disk. Unfortunately, we did not have the original raw-format files, so

	SMTP 10/31-11/7	POP 10/31-11/7	IMAP 10/31-11/7	CAMPUS 10/21-10/27	EECS 10/21-10/27	INS	RES	NT	Sprite
Year of Trace	2009	2009	2009	2001	2001	2000	2000	2000	1991
Days	7	7	7	7	7	31	31	31	8
Total ops (millions)	485.17	133.52	651.32	26.7	4.44	8.30	3.20	3.87	0.432
Data read (GiB)	175.04	185.83	1925.12	119.6	5.10	3.05	1.70	4.04	5.36
Read ops (millions)	26.02	8.99	103.80	17.29	0.461	2.32	0.303	1.27	0.207
Data written (GiB)	67.07	0.65	42.26	44.57	9.086	0.542	0.455	0.639	1.16
Write ops (millions)	26.34	0.52	29.97	5.73	0.667	0.15	0.071	0.231	0.057
Read/Write bytes ratio	2.61	284.27	45.56	2.68	0.56	5.6	3.7	6.3	4.6
Read/Write ops ratio	0.99	17.36	3.46	3.01	0.69	15.4	4.27	4.49	3.61

Table 8.1: . A summary of average daily activity during trace periods. The CAMPUS and EECS are from Daniel Ellard's study. The INS, RES, NT and Sprite numbers are from the Roselli and Baker trace studies. INS is an instructional workload, RES is a research workload, and NT is a Windows NT desktop workload.

in order to diagnosis the problem, we captured a few minutes of traffic and converted it to DataSeries format. We then compared the number of read and write calls on both the original raw-file and the DataSeries file, to look for missing packets in the DataSeries file. We realized that the number of reads was the same in both files, but the number of writes in the DataSeries file was roughly half of that seen in the raw-file.

Anderson also noticed that the number of write operations were underestimated in his traces[5]. His further examination indicated that the problem was due to the parsing techniques for TCP packets employed in his tools. They started at the beginning of the packet and parsed all of the RPCs found that matched all required bits to the RPCs. Unfortunately, over TCP, two back to back writes will not align the second write RPC with the packet header, and they will miss subsequent operations until they re-align with the packet start. Anderson indicated that his write workload could be increased by a factor of 1.5x if these write calls were not missed.

In our case, we estimate that the number of missed write calls is a lot higher. When looking at the storage traffic on the ISP's network monitoring tools, we realized that the outbound traffic (from the SMTP server to the storage nodes) is around 5x larger than the inbound traffic. We are aware that not all traffic is NFS RPC calls. But, if we assume it is, when comparing the storage traffic with the values on Table 8.1, we can conclude that our write workload could be increased by a factor of 13x if these write calls were not missed.

We also validated the values for the IMAP and POP services. For these two services, we had the number of read and write NFS operations on the ISP's networking monitoring

operation	SMTP Mops	POP Mops	IMAP Mops	anim-2007/set-2 Mops	anim-2007/set-5 Mops
lookup	601.53	750.51	1270.89	643.85	807.13
access	373.54	604.30	920.07	4000.20	3570.40
getattr	377.31	369.71	644.26	6598.52	2756.79
readdir+	6.86	286.53	150.55	32.81	20.27
read	1658.11	215.60	742.88	1460.67	1761.20
readdir	0.471	88.58	175.60	23.32	18.35
remove	22.41	9.397	76.23	<10	<10
write	184.36	9.065	209.79	32.40	45.18
rename	23.42	2.052	61.94	<1	<1
create	42.13	0.324	70.31	<10	<10
link	0.387	0.320	0.351	<10	<10
setattr	105.66	0.153	236.14	<10	<10
mkdir	0.020	0.002	0.222	<1	<1

Table 8.2: Overview of all the operations that occurred in the traces. Anim-2007 sets are from Anderson's study and represent the NFS operations from a feature animation company.

tools. When comparing the ratio of read/write calls for both services, we found them to be comparable to our readings.

Table 8.2 breaks down the workload in terms of type of operations. The types of operations in NFS version 3, along with a short description for each type are described in Table 3.1. In Table 8.2 we also present the numbers from Anderson's study, but we will not try to compare them. However, it is important to emphasize that different types of storage workloads produce different characteristics. Thus, it is important that the community publishes and analyzes specialized workloads in specific environments rather than over-generalizing the already published studies.

When we analyze the types of operations used by each email service, besides read and write operations, we can see some interesting properties that not only are very dependent on the type of service in question, but also dependent on the type of software used to provide each service and to the mailbox format.

A property that is very specific to the mailbox format is the percentage of *rename* operations in our workload, which is a lot higher than in Anderson's workload. In the Maildir format, each new email message is stored in a file in the *new/* directory. When the client opens the mailbox, the email message is moved (renamed) to the *cur/* directory by the server software. Thus, there is a significant number of *rename* operations. If the mailbox format was a different one, the workload's properties could be very different.

An example that shows a property that is dependent on the protocol and software used by the server, is the percentage of *readdir+* calls for SMTP, which is only 0.2% of the total number of operations issued by SMTP, while for IMAP is 3% and for POP is 30%. The SMTP protocol does not need to scan the contents of the mailboxes, since it just stores arriving email in the *new/* directory of the user's mailbox. However, IMAP and POP need to scan the mailbox for existing email messages (recall that each email message is stored in a separate file), since both are protocols used for users to read their email messages. But why the discrepancy between the POP and the IMAP values?

Further examination indicated that the IMAP software uses indexes to store information about the contents of the mailboxes, so it does not need to scan the directory every time the users list a folder. On the other hand, POP software does not use any indexes. Thus, it needs to scan the mailbox for new email every time the users check their email. In fact, indexes would not be useful for POP servers, because typically the user fetches new email and the email is deleted from the server. Thus, that data will not be listed ever again.

Another example that shows the particularities of the each protocol is the *mkdir* operation. The reason that the number of *mkdir* operations is an order of magnitude higher in IMAP than in POP is that the IMAP protocol supports folders. Thus, users can create folders, while the POP protocol does not support folders.

If we were to build a specific filesystem for POP and IMAP, we would want the improve the performance for accessing the metadata, since the top five operation types are all related to metadata information.

8.2 Operation rates and read bandwidth

Figure 8.1 shows the number of the total operations per second as approximate quantiles for each email service, using hourly averages and second averages. It shows how long averaging intervals can distort the load placed in storage system. As Anderson refers in his study, if we were to develop a storage system for the hourly loads reported in most papers, we would fail to support the substantially higher near peak (99%) loads seen in the data.

One interesting observation from Figure 8.1 is the discrepancy between 90% of operation rates for the SMTP service and the remaining 10%. For the SMTP service, 90% of the operation rates are under 5000 operations per second, whereas the remaining 10% go up to 45000 operations per second. POP and IMAP on the other hand do not show any equivalent behavior. We can speculate that the reason for this behavior might be due to new spam

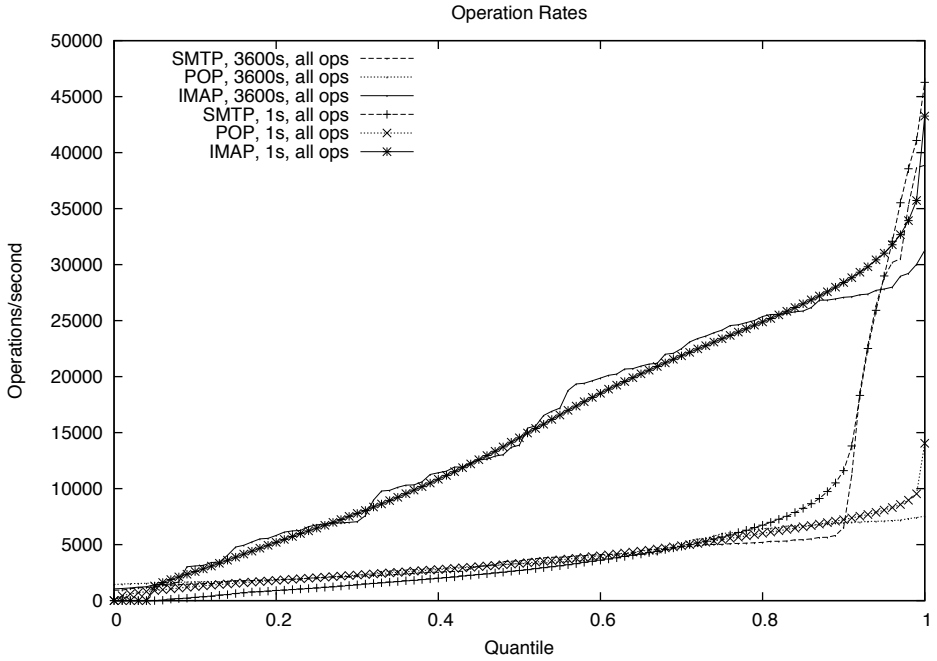


Figure 8.1: Operation rates, as quantiles

attacks, when the SMTP servers are flooded with spam messages that are not yet identified by spam filters.

Figure 8.2 shows the bandwidth used for read operations. The graph shows the payload data transferred, so it includes the offset and filehandle of the read request and the size and data in the reply, but does not include IP headers or NFS RPC headers.

When we look at second averages for the IMAP workload, we can see a substantially higher near peak (99%) load, which indicates that at the busiest times, the IMAP read rates go from under 5MiB per second to nearly 60MiB per second.

8.3 Variations of the Workload due to Time and Day

Figure 8.3 shows the variation of the hourly average of NFS operations throughout the week. We can clearly see that the number of NFS operations is at least 2x more intense from Monday to Friday than during the weekend and that the IMAP clearly stands out in terms of the total number of NFS operations when compared to the other two. When comparing

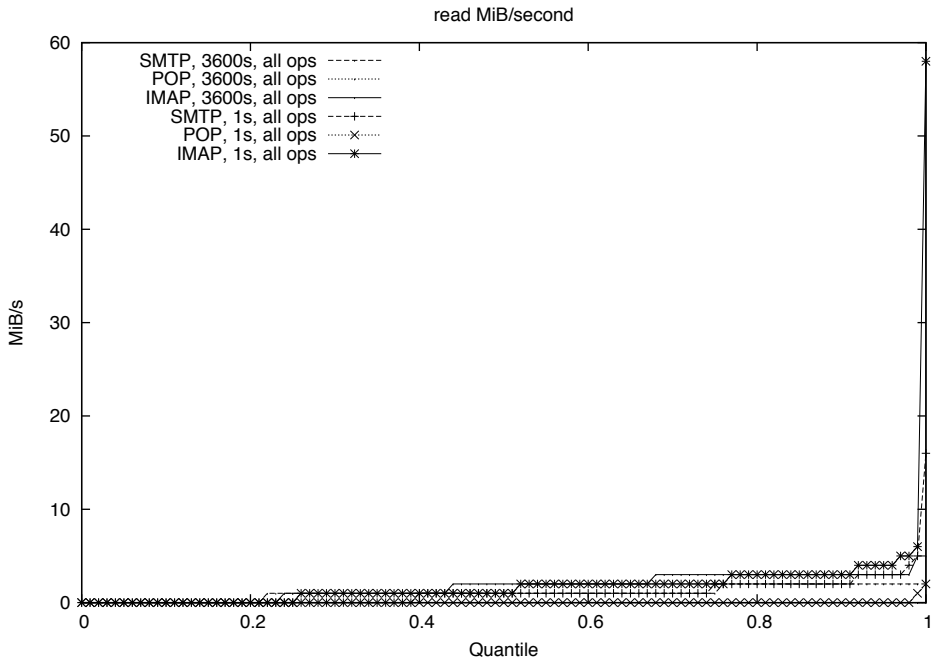


Figure 8.2: Bandwidth for reads

these values with the ISP's network monitoring tools, we realized that the Monday that we traced was atypical, since Mondays are generally like any other week day. However, the gap between weekdays and weekends still holds.

When we compare this graph with the Ellard's email study [6], we can see that the gap between the volume on weekdays and the weekend is not as high as our values. This might indicate that the typical user reads email less often on weekends, while academic users do not.

One other interesting property is that we can clearly see the breaks at lunch and dinner time. Ellard's traces show that there is only one break at around dinner time. We believe that this can be due to cultural issues between Portuguese and Americans. It is common in Portugal to have a one hour break for lunch, while in the US, the majority either take a small break for lunch or none at all.

Figure 8.4 shows the variation of the hourly read/write ratios. It is at idle times that the read/write ratio shows its tendency to spike, specially with IMAP. However, at busy times the read/write ratio is consistent. One reason for the IMAP spikes at idle times can be that few users leave their mailboxes open all the time, and when the load is at its minimum, the

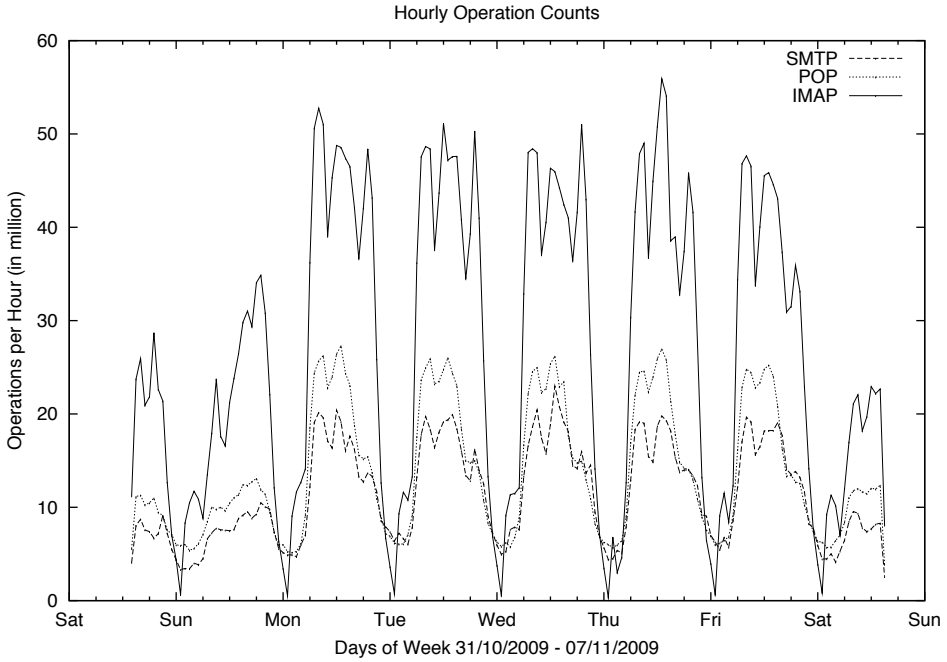


Figure 8.3: Variation of the hourly total operation count

number of writes decreases, since the user session is not really active.

Table 8.3, shows the hourly average activity. The numbers in parenthesis are the standard deviations of the hourly averages, expressed as a percentage of the mean. When we observe 24 hours of the day, which include peak and off-peak periods, we can observe a large variance of load characterization statistics over time. This correlation has been observed in many trace studies like in Ellard's study, also shown in Table 8.3.

Like Ellard, we also examined the activity at peak hours, to verify the variance between peak and off-peak hours. We can see that the variance at peak hours is three to four times less than the variance seen in the 24 hours range. This indicates periods of idleness, which could be used for maintenance operations or for file system optimizations. Systems that experience significant periods of idleness can use them to rearrange data in anticipation of the next period of heavy use.

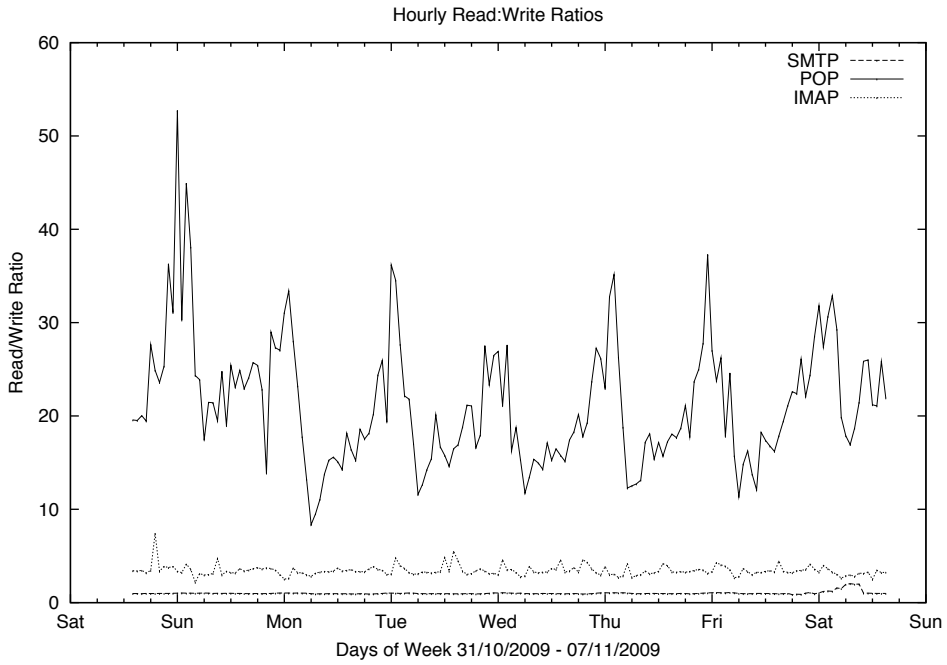


Figure 8.4: Variation of the hourly read/write ratios

8.4 File Sizes

As referred in Anderson's study [5], file sizes affect the potential internal fragmentation for a filesystem. They affect the maximum size of I/Os that can be executed, and they affect the potential sequentiality in a workload.

In NFS version 3, the maximum size in bytes of a read request is dictated by the server. In our workload, we have seen read responses with payloads of 32Kb. If the majority of the accessed files are smaller than this value, it indicates that they can be read in a single I/O operation. In Figure 8.5, we show the file size distribution for all accessed files. We can see that 80% of the accessed files are smaller than 32Kb. We can also see that only 5% of the accessed files are bigger than 1Mb, and only 1% are bigger than or equal to 10Mb.

Before the capture, the ISP provided us with some statistics on file sizes. According to those statistics, 5% of the files are bigger than 1Mb, which corroborates with what we have seen in the workload. Furthermore, those 5% of files bigger than 1Mb take 85% of the total storage space, or in other words, 95% of files in the storage only take 15% of the total storage space. This indicates that the filesystem for an email service should be designed to

	All Hours									
	SMTP		POP		IMAP		CAMPUS		EECS	
Total Ops (1000s)	11350	(47%)	13744	(52%)	26819	(61%)	1113	(48%)	185.1	(86%)
Data Read (MiB)	7381	(44%)	19588	(59%)	81172	(71%)	4989	(45%)	212.3	(165%)
Read Ops (1000s)	1072	(45%)	925.54	(56%)	4274	(67%)	.719	(48%)	19.7	(110%)
Data Written (MiB)	2828	(53%)	68.91	(78%)	1782	(75%)	1856	(58%)	378.5	(246%)
Write Ops (1000s)	1084	(49%)	53.32	(75%)	1234	(63%)	239	(58%)	28.6	(201%)
R/W Op Ratio	1.02	(17%)	21.25	(32%)	3.43	(16%)	3.27	(48%)	3.16	(242%)
	Peak Hours Only									
	SMTP		POP		IMAP		CAMPUS		EECS	
Total Ops (1000s)	17838	(13%)	23496	(11%)	44686	(13%)	1699	(7.6%)	267	(68%)
Data Read (MiB)	11471	(12%)	29368	(13%)	129554	(26%)	7153	(6.1%)	268	(146%)
Read Ops (1000s)	1655	(12%)	1472	(9%)	6991	(20%)	1088	(7.1%)	29.2	(77%)
Data Written (MiB)	4570	(14%)	132.93	(31%)	2769	(22%)	2934	(12%)	439	(228%)
Write Ops (1000s)	1728	(13%)	101.29	(26%)	2023	(12%)	377	(12%)	341	(158%)
R/W Op Ratio	0.96	(1%)	15.07	(16%)	3.45	(15%)	2.46	(10%)	1.13	(106%)

Table 8.3: Average hourly activity. The All Hours columns are for the entire week of 10/31 - 11/07/2009. The peak hours are the hours 9am-6pm, Monday 11/02-11/06/2009. The number in parentheses are the standard deviations of the hourly averages, expressed as a percentage of the average.

achieve the best performance in the presence of small files.

8.5 Sequentiality

Due to the mechanical characteristics of hard disks, sequential accesses are faster than random accesses. This makes sequentiality one of the most important properties of storage systems.

A block is accessed sequentially if it is consecutive to the previous access. The concept is simple, but the way it is calculated is not.

Prior work has presented various methods for calculating sequentiality and is summarized in Anderson's study [5]. Since we used Anderson's tools to calculate sequentiality, we will use Anderson's method.

Anderson determines sequentially by reordering within temporally overlapping requests. Given two I/Os, A and B, if the request-reply intervals overlap, then we are willing to reorder the requests to improve estimated sequentiality.

When we ran the tool to calculate the sequentiality of the workload, we got several errors

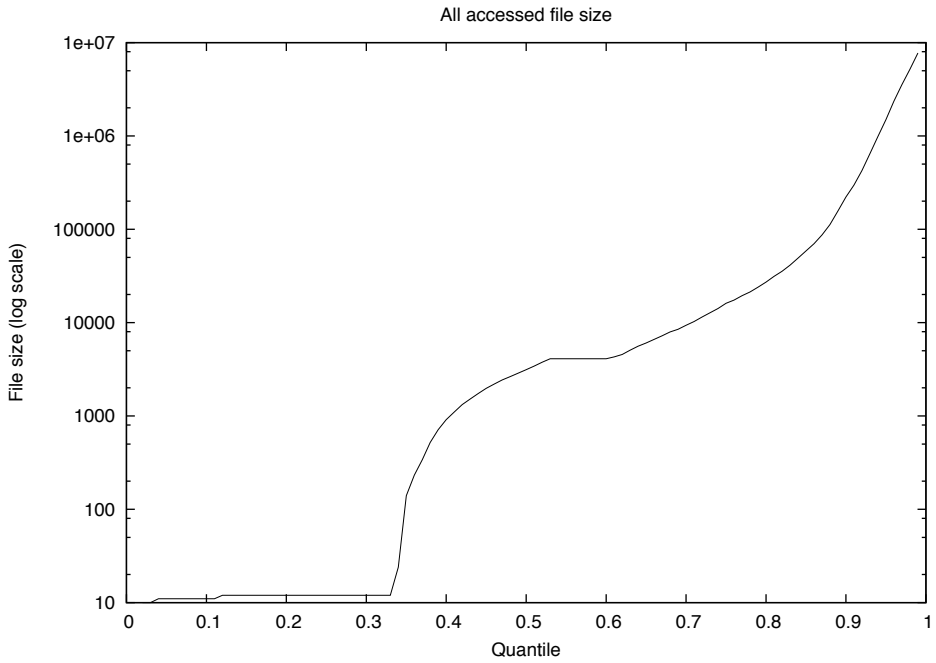


Figure 8.5: File size distribution for all accessed files

because there were duplicate record Ids. Further examination of the problem revealed that the way that we ran the conversion tool was not right. This is a typical case where the lack of available documentation led us to misuse the tool. We would not have had experienced any problems if we had invoked the conversion tool only once, to convert all of the raw-capture files. However, we were converting the raw-capture files as soon as the files were closed by the capture program. Thus, we invoked the conversion tool thousands of times, and each time it was invoked, the record Id was reset to 0. All the records are expected to have multiple unique Ids and to be ordered. Because we had multiple files with record 0, we violated that assumption.

This problem did not affect any of the other analyses, so we decided to capture another trace for 5 hours and run the sequentiality analysis on the new trace, which ran successfully.

Figure 8.6shows the number of reads in a group. We can see that 80% of the groups are single I/O groups.

Figure 8.7 shows the number of bytes accessed in sequential runs within a random group. We can see that if we start accessing a file at random, 93% of the time we will do single or double I/O accesses (8-32KiB). However, we also get some extended runs within a random

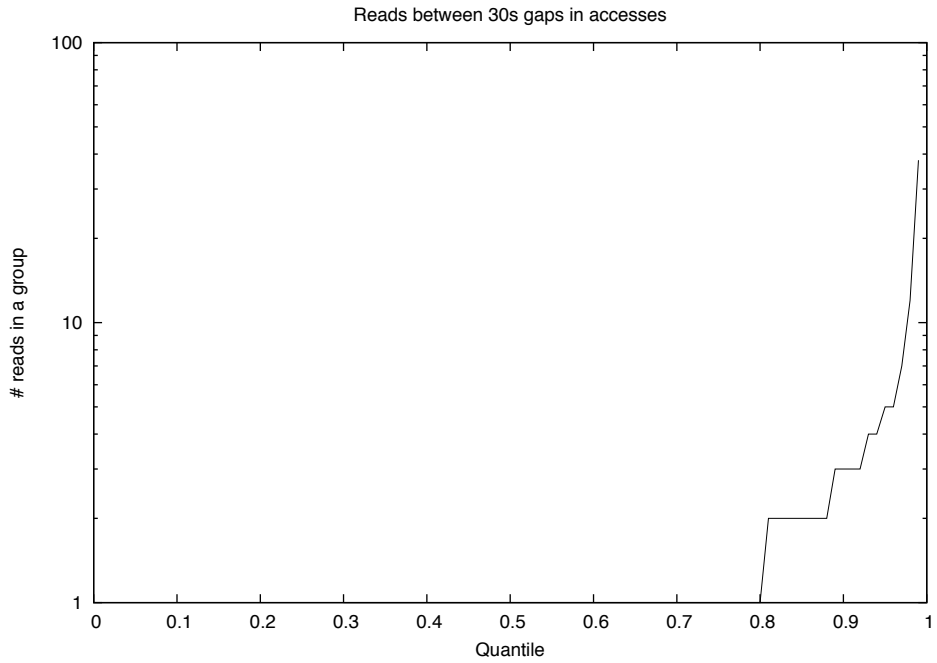


Figure 8.6: Number of reads in a group

group, although 99% of the runs are less than or equal to 1 MiB.

8.6 Attachment types

One question that we ask ourselves often is where does the free space goes. To try to answer that question, we analyzed one of the storage volumes.

As previously mentioned, the mailbox format stores one message per file. The content of the file is the message source, which includes the attachments, as defined in the Internet Message Format RFC [27] as well as in the set of MIME RFCs [28][29][30][31][32].

The analyzed storage volume had 1.6Tb of used storage space and 3.6 million MIME messages. The total number of MIME parts was 6 million. In Figure 8.8, we group each type of MIME part (attachment) by type. The graph on the left shows the top total number of attachments for each type, and the one on the right shows the top total space used by each type of attachments.

By analyzing the graphs on Figure 8.8, one can see that the total storage used by those

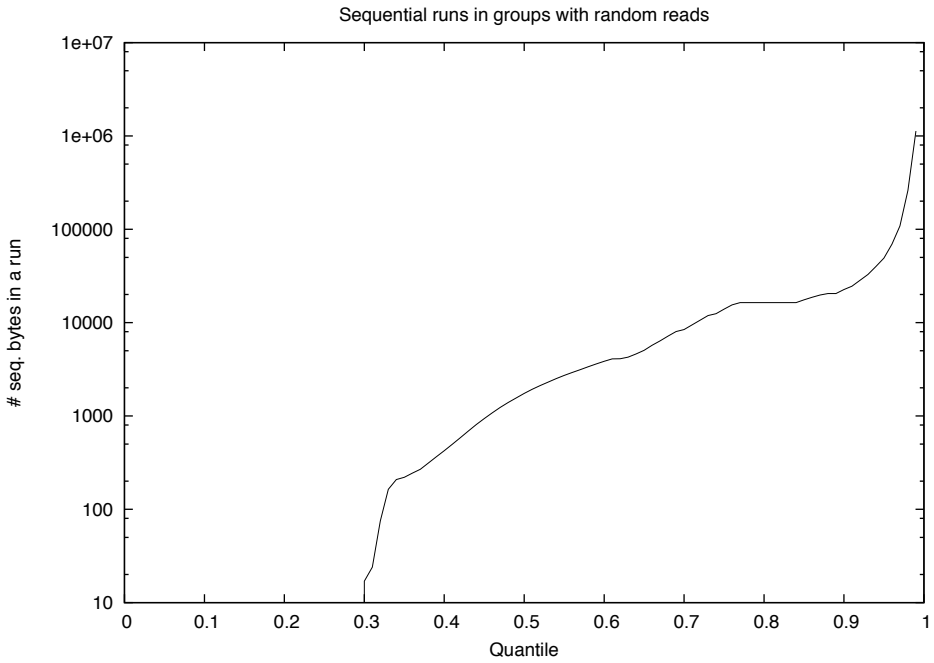


Figure 8.7: Number of bytes accessed in sequential runs within a random group

types of attachments is 1.4Tb (88% of the total used storage). The storage used by video attachments, for example, represent 32% of the total even though it only represents 3% of the total number of attachments.

The next question was how many of those MIME parts are duplicates. To answer this question, we calculated an MD5 for each MIME part. Most of the duplicate MIME parts were images, and the total amount of used space for duplicates was 689Gb (40%). Based on these numbers it is clear that, if the ISP deployed data deduplication, it could reduce the budget for storage acquisition.

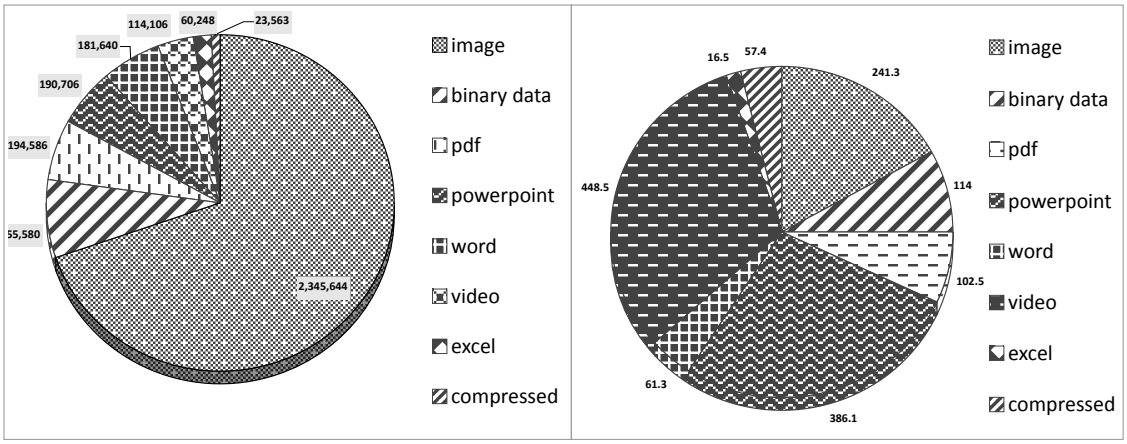


Figure 8.8: Number and Size of types of attachments. The graph on the left indicates the number of attachments of each type. The graph on the right indicates the used storage space by each type of attachment.

Chapter 9

Conclusion

In this thesis, we provide an in-depth explanation of how we captured high data rates without packet loss. We conclude that most of the experienced problems are due to disk I/O, so we used a ramdisk as temporary storage for the captured raw-files (until they are converted).

Due to storage limitations on the ISP side, we probably would not have been able to capture the email workload for 7 days if we had not known of the existence of DataSeries. DataSeries proved to be the right tool for the job. It enabled us to quickly convert and analyze the traces, and it was very efficient in compacting the original raw-files.

The lack of available storage space to store the original raw-files was problematic, since it added complexity in the capturing process and due to a misuse of the conversion tool, prevented us from re-converting the original files in order to determine sequentiality.

We have analyzed our NFS workload and identified some of the unique properties that are present in an email workload, such as the periodic nature of file system activity, the file size distribution for all accessed files, the sequentiality of accessed data, and the most common type of attachments found in a typical mailbox.

The ISP network team informed us that the email network infrastructure would be replaced in the near future. This infrastructure change would allow us to port mirroring one of the Netapp's NICs. It would be interesting to do another analysis on the same email system after the deployment of the new architecture.

Researchers interested in acquiring a copy of the anonymized traces used in this paper, or newer ones as they become available, should contact us. Please refer to http://www.pdl.cmu.edu for the contact information.

Bibliography

[1] ANACOM. Internet Access Service - 2nd Quarter 2009, Jul 2009. Available: http://www.anacom.pt/render.jsp?contentId=970389. (document), 3.1, 3.1

[2] J.K. Ousterhout, H. Da Costa, D. Harrison, J.A. Kunze, M. Kupfer, and J.G. Thompson. A trace-driven analysis of the UNIX 4.2 BSD file system. *ACM SIGOPS Operating Systems Review*, 19(5):24, 1985. 1

[3] KK Ramakrishnan, P. Biswas, and R. Karedla. Analysis of file I/O traces in commercial computing environments. In *Proceedings of the 1992 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 78–90. ACM New York, NY, USA, 1992. 1, 2

[4] A.W. Leung, S. Pasupathy, G. Goodson, and E.L. Miller. Measurement and analysis of large-scale network file system workloads. In *USENIX Annual Technical Conference*, 2008. 1, 2, 5.2.1

[5] E. Anderson. Capture, conversion, and analysis of an intense NFS workload. In *Proccedings of the 7th conference on File and stroage technologies table of contents*, pages 139–152. USENIX Association Berkeley, CA, USA, 2009. 1, 2, 5.2.1, 5.2.3, 6, 8, 8.1, 8.4, 8.5

[6] D. Ellard, J. Ledlie, P. Malkani, and M. Seltzer. Passive NFS tracing of email and research workloads. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 203–216. USENIX Association, Mar 2003. 1, 2, 8.3

[7] E. Anderson, M. Arlitt, C.B. Morrey III, and A. Veitch. DataSeries: an efficient, flexible data format for structured serial data. *ACM SIGOPS Operating Systems Review*, 43(1), January 2009. 1, 2, 6

[8] M.G. Baker, J.H. Hartman, M.D. Kupfer, K.W. Shirriff, and J.K. Ousterhout. Measurements of a distributed file system. In *Proceedings of the thirteenth ACM sympo-*

sium on Operating systems principles, pages 198–212. ACM New York, NY, USA, 1991. 2, 8.1

[9] M. Blaze. NFS tracing by passive network monitoring. In *Proceedings of the USENIX Winter 1992 Technical Conference*, pages 333–343, 1992. 2

[10] D. Ellard and M. Seltzer. New NFS tracing tools and techniques for system analysis. In *Proceedings of the Annual USENIX Conference on Large Installation Systems Administration*, pages 73–85. USENIX Association, 2003. 2, 6

[11] Tcpdump/libpcap. Available: http://www.tcpdump.org/. 2, 5.2.1

[12] Corey Satten. Gulp, Aug 2007. Available: http://staff.washington.edu/corey/gulp/. 2, 5.2.4, 7.2

[13] B. Callaghan, B. Pawlowski, and P. Staubach. NFS Version 3 Protocol Specification. IETF RFC 1813, Jun 1995. 3.2

[14] DJ Bernstein. QMail. Available: http://qmail.org/. 4.2

[15] T. Sirainen. Dovecot IMAP. Available: http://dovecot.org/. 4.2

[16] Sam Varshavchik. Courier-IMAP Server. Available: http://www.courier-mta.org/imap/. 4.2

[17] DJ Bernstein. Using maildir format, 2003. Available: http://cr. yp. to/proto/maildir. html. 4.2, 6.1

[18] Maildir++. Available: http://www.inter7.com/courierimap/README.maildirquota.html. 4.2, 6.1

[19] J. Rubio-Loyola, D. Sala, and A.I. Ali. Maximizing Packet Loss Monitoring Accuracy for Reliable Trace Collections. In *16th IEEE Workshop on Local and Metropolitan Area Networks, 2008. LANMAN 2008*, pages 61–66, 2008. 5.2.1

[20] L. Deri. High-speed dynamic packet filtering. *Journal of Network and Systems Management*, 15(3):401–415, 2007. 5.2.1

[21] Wireshark. Available: http://www.wireshark.org/. 5.2.2

[22] L. Degioanni, F. Risso, and G. Varenni. PCAP Next Generation Dump File Format, March 2004. Available: http://www.winpcap.org/ntar/draft/PCAP-DumpFileFormat.html. 5.2.3

[23] HP Labs. DataSeries technical report, March 2008. Available: http://tesla.hpl.hp.com/opensource/DataSeries-tr-snapshot.pdf. 6

[24] A. Traeger, E. Zadok, E.L. Miller, and D.D.E. Long. Findings from the First Annual File and Storage Systems Benchmarking Workshop. In *Initial workshop report*, 2008. 6

[25] T. Williams, C. Kelley, et al. GNUplot: an interactive plotting program, 1993. Available: http://www.gnuplot.info/. 8

[26] D. Roselli, J.R. Lorch, and T.E. Anderson. A comparison of file system workloads. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, page 4. USENIX Association, 2000. 8.1

[27] P. Resnick. Internet Message Format. IETF RFC 2822, Apr 2001. 8.6

[28] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part One. IETF RFC 2045, Nov 1996. 8.6

[29] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part Two. IETF RFC 2046, Nov 1996. 8.6

[30] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part Three. IETF RFC 2047, Nov 1996. 8.6

[31] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part Four. IETF RFC 2048, Nov 1996. 8.6

[32] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part Five. IETF RFC 2049, Nov 1996. 8.6