UiO : Department of Informatics
University of Oslo

A Benchmarking System for the SAMBA/CIFS Protocol

An Applied Learning and Replay Methodology

Guang Yang

Master's Thesis Spring 2014

Abstract

This project focuses on how to benchmark a SMB/CIFS storage with trace and replay methodology. Traces are used primarily by file system researchers in an attempt to understand, categorize, and generalize file system workloads. However, because such traces provide a detailed information about how a specific system is actually used, they should also be of interest to system administrators. The goal of this thesis is to produce a trace-driven synthetic workload and show that it is similar with the original workload for all practical purposes. This has been achieved by examining properties of the original workload such as the inter-arrival time and request length. Upon examination of the original workload, it is shown that our system regenerate a synthetic workload with the same properties. The synthetic workload is also been used on several SAMBA/CIFS servers with different hardwares, the variety of the results show the system can be used to benchmark the performance of a system.

Acknowledgements

I would like to express my very great appreciation to Hårek Haugerud. Advice given by him has been a great help in both research area and writing skills. Dr Begnum Kyrre provided me with very valuable advices during the research as well.My grateful thanks are also extended to my classmates, their willingness to give their time and ideas so generously has been very much appreciated.

Finally, I wish to thank my lovely wife for her consistent support and encouragement throughout my study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter explains why we are focusing on storage benchmarking and what challenges we variety are facing?

1.1 Motivation

According to the IDC report[9], world wide IT spending is expected to reach 2.14 trillion dollars in 2014. The investment on storage reaches $37.3 billion by 2015. IDC also predicts that in the year of 2017, the raw digital data storage capacity could rise to 7,235 EB. This tremendous amount of investment is great news for all the manufacturers in this industry, but it introduces new challenge as well.

Today we are in an information-explosion era, more and more companies create their private or public cloud environments. The enterprise storage system is a critical component of the entire system. The performance of the storage system affects all upper layer applications and devices.
Realizing this fact, all the manufacturers advertise their storage system by performance specs. But the different products have been optimized for the different data structures and workloads. In some cases it is determined by the hardware, in other cases it is affected by manufacturers' operation system on the top of the raw storage box. Hence there is not universal solution that can satisfy all kinds of workloads. The storage administrator has to conFigure a proper synthetic workload carefully, which is demanding task for them.

Normally users always have one typical and common question for the pre-sale engineers, "I understand all the specs you have shown, but I am more interested in how the performance will be in our product environment." Thus the best way to answer the question is testing all target storages under real workload. It requires two steps. First one has to implement all target storages in their product environment, then create a mirror or snapshot of all their production data from the

original storage to the new target. By monitoring the performance matrix of the new implemented storage system, one could make a solid conclusion about each candidate system. But in real cases, no company would do this, since the procedure requires a too large amount of human resource, time, financial support, etc. An alternative solution is naturally required. How one can test all target systems without affecting the product environment is very important.
Some research has been done addressing this practical problem [2, 1, 3]. The basic idea is so called Trace and Replay by [1]. The first step is to learn from the current workload, and then create a synthetic workload based on what one learned previously. According to [3], they have implemented a benchmarking tool, known as TBBT, for NFS workload. They also applied with the trace and replay methodology. There is also an solution for iSCSI protocol, developed by [2]. But there is not any trace and replay benchmarking tool for SMB/CIFS protocol.

The thesis is to target this empty area. Our solution makes a methodology without full application deployment and which does not affect the performance of the real production environment. This paper is focusing on SMB/CIFS storage protocol environment, it is the most common storage protocol used in Windows and Linux environment. By sniffing the network, one can trace storage transections, and reproduce the workload based on it. The system can serves as a decision making tool for storage administrator when purchasing.

1.2 Problem Statement

To get a more accurate benchmark result for every unique SMB/CIFS storage environment, we should learn from each environment and then generate the test workload based on it. In this thesis, we focus on SMB/CIFS storage system instead of the local disk or storage area network (SAN) system. The reason for bypassing SAN/IP-SAN is that one has to implement an initiator to do that. Unlike the file level request replaying, one has to implement a client or an initiator for the block level replay. This would require a long period of development. The time is limited for this thesis work, so we only focus on SMB/CIFS benchmarking.

To accomplish our goal, we implement a system which solves the following challenges,

- How to learn the original workload without compromising users' storage performance
- How to generate a synthetic workload based on the first step
- How to show that the synthetic workload has the same impact as the original workload

- Run the simulation against different storage systems and verify that the results are useful when evaluating the capacity of the storage system

By answering these three questions, we can show that our system is reliable to benchmark the performance of a new SMB/CIFS storage.

Chapter 2

Background

In this chapter, we introduced all related knowledge. They include all storage structures, benchmarking methodologies and related works.

2.1 Storage structures

Today, modern storage system is divided into three parts, DAS (Direct-Attached Storage), SAN (Storage Area Network) and NAS (Network Attached Storage). The following sections introduce each of them respectively

2.1.1 DAS

DAS is most widely used in our daily life. It is a storage system directly attached to a computer. It is a non-networked storage. Typically all hard disks connect to a server or a workstation through HBA (Host Bus Adapter). Since the structure of DAS does not rely on the network, and usually HBA is embedded in computer, DAS turns to be easy to use.

2.1.2 SAN

2.1.2.1 History

Along with the data burst era came, DAS can no longer satisfy users' requirements on capacity and performance. SAN is designed under such circumstance, which introduces the idea of separating storage system from computer system. SAN is defined by Storage Networking Industry Association (SNIA) as a network. Its major mission is to transfer data among computer systems under high performance[4]
.

The most common protocol used by SAN is SCSI (Small Computer System Interface). In the early era of internet, SCSI protocol was designed for faster data processing. This protocol is still commonly used in DAS structures today. SCSI is recognized by its high performance and stability. Since most SAN are connected through fiber channel, SCSI commands are transported over FCP (Fiber Channel Protocol). SAN storage meets requirements on large scale storage capacity and high performance.

Although SAN has dominate advantages on capacity and performance, it is only used in large companies or government at first. The reason is the prices of fiber channel switch and cable are expensive compare to the Ethernet. By realizing the price is the major limitation of development in SAN technology, iSCSI (Internet Small Computer System Interface) protocol is innovated. Instead of transferring SCSI command over FCP, iSCSI, known as IP SAN, are transferred by IP protocol through Ethernet network. IP SAN solution has the same structure as traditional FC SAN. The transfer speed of FC is 8 Gb/sec. It is higher than Ethernet cable with 1 Gb/sec. But the speed of new Ethernet is about to leapfrog fiber, with 10Gb/sec. In conclusion, IP SAN can run in the same speed as standard SAN storage. Meanwhile one can leverage from current enterprise IP network. It lower the TCO (total cost of ownership) by sharing the existing IP network and network administrators. More and more enterprises are migrating from FC SAN to IP SAN recently.

2.1.2.2 SCSI

SCSI is a set of ANSI(American National Standards Institute) standard electronic interfaces that used for system communicate with hard disks, tape drivers, CD-ROM drivers or printers.

SCSI commands are transferred in CDB (Command Descriptor Block). Each CDB can be a total of 6, 10, 12, or 16 bytes, but later versions of the SCSI standard also allow for variable-length CDBs. The CDB consists of a one byte operation code followed by some command-specific parameters.

There are two parts of SCSI architecture, clients and server. Client is defined as initiator in this module as server is target. Initiator will initiate any communications between two parts. The target devices accessed by initiator, it uses as block devices. A block device is a computer data storage device that supports reading and writing data in fixed-size blocks, sectors, or clusters. This also means the file structure is transparent to SCSI devices. They need to break down a file to blocks. So the host of initiators has the responsibility to mange how files are stored and retrieved. In the other word SAN devices only provided storage but not file system.

2.1.2.3 iSCSI and IP SAN

The iSCSI protocol maps the SCSI to TCP/IP. It defines the way that how could initiator encapsulate the SCSI data to TCP/IP packet, and also how should target decode the coming TCP/IP packet to SCSI command and data. The benefits of iSCSI protocol can be obviously, SCSI is the most common applied protocol by most storages, tapes and other devices. Meanwhile TCP/IP is also the most stable and widely used network communication protocol. The combination of the two protocols makes iSCSI been an cheaper alternative solution of FC SAN.

2.1.3 NAS

Compare with SAN, NAS provides both filesystem and storage to clients. NAS devices will contain file system their self, which is totally isolated with the filesystem of clients. The architecture of NAS is also a client-server structure. Clients use particular protocol to communicate with server, the most widely used NAS protocols are NFS (Network File System) and SMB/CIFS(Server Message Block-/Common Internet File System). Both predate the modern NAS by many years; original work on these protocols took place in the 1980s. NAS is a remote file protocol. By definition, it must be accessed from remote devices through a certain network, for example LAN. CIFS and NFS are also encapsulated by TCP/IP or UDP. That means NAS device can leverage from the current network environments. NAS devices have their own IP address over network, so all users could access to all files on NAS devices directly as long as their have the privilege to do such operation.

By setting up the file system on storage sides, the NAS devices could provide more file level function then SAN devices, for example user privilege control. The major task of NAS is how to find and mange files, instead of file itself, so this is a different scenario compare with block device such as SAN. This will be a critical advantage in enterprise environment, since they usually have very complicated privilege structure. NAS could employe the current user administration system, for example Active Directory for privilege control. The storage administrator will be relieved from user management requirements. In the following sections, we will introduce more details about NFS and SMB.

2.1.4 NFS

There are four versions of NFS since the first time it has been introduced by Sun. The most recently version of it is NFS V4. In this paper we will only discuss about version 4.

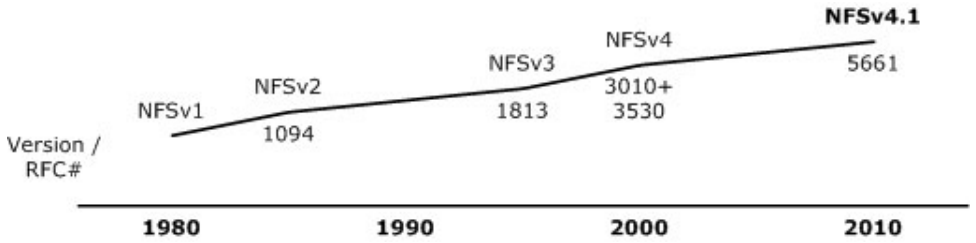

Figure 2.1: NFS history[5]

NFS is the first network file system. It began in the early 1980s as an experimental file system at Sun Microsystems. Since NFS protocol was widely used, it was documented as a Request for Comments (RFC) specification. That was the time NFSv2 was introduced. It evolved into version 3, by large file supporting, asynchronous writes, also used TCP as the transport protocol, which enabled it to extend to world wide network. Today, the newest version is 4.1, documented as RFC 5661. The major change is they add protocol support for parallel access across distributed servers. IBM illustrated the history of NFS as Figure 2.2 shows.2.1

NFS is a client-server structure, as Figure 2.3 shows [11]. It encapsulates NFS CMD into TCP/IP packet. By capture the network transections, one can learn attributes of files on serve, such as file name, path. NFS is also a stateless protocol. File server does not store client information, and server and client do not maintain a connections between them. For example, NFS has no operation to open a file, since this would require the server to store state information. Instead, NFS supports a Lookup procedure, which converts a filename into a file handle. This file handle is an unique, immutable identifier, usually an i-node number, or disk block address. NFS does have a Read procedure, but the client must specify a file handle and starting offset for every call to Read. Two identical calls to Read will yield the exact same results. If the client wants to read further in the file, it must call Read with a larger offset.

2.1.5 SMB/CIFS

2.1.5.1 History

SMB stands for Server Message Block, and CIFS stands for Common Internet File System.

From CIFS to SMB version1 and SMB version 2, they have evolved SMB protocol. They are developed by many of storage vendors and operating system vendors for NAS solution. The first invention of SMB is by Dr. Barry Feigenbau, an IBM employee[6]. He first named it after his own name initial “BAF”, and then

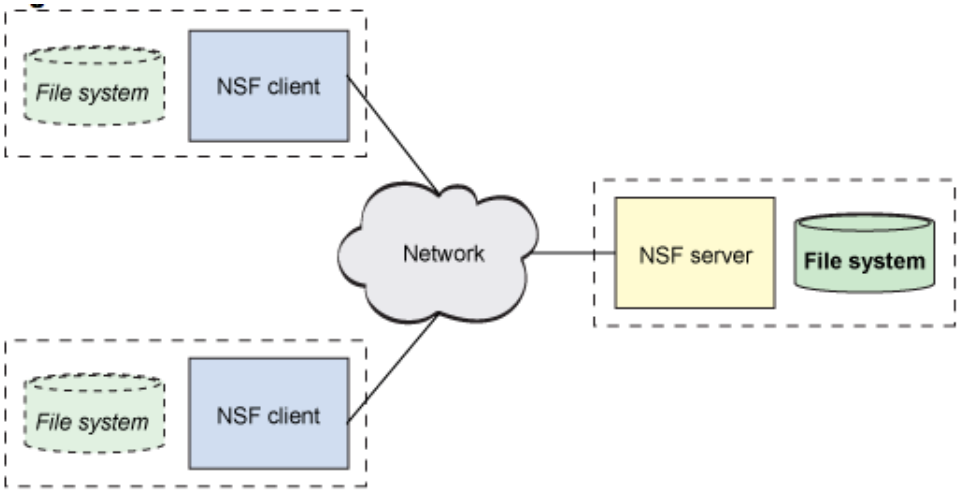

Figure 2.2: NFS structure[5]

changed it to “SBM”. Currently, two versions of SMB are widely used, they are SMB 2 and 3. Version 2 came along with Windows Vista in 2006. In 2011 SNIA announced the SMB 3.0 protocol. In the end of next year its first commercial products by Microsoft, NetApp and EMC has been published. Right now SMB has been implemented by the following venders: Microsoft, NetApp, EMC, IBM, CISCSO and so on.

2.1.5.2 Protocol detail

Since SMB is a stable and complicate protocol, the structure of this section will be organized as two parts. One could have a understand about SMB message structure at first part, and learn its process in the second part.

SMB Message Structure

- The SMB header

The header of SMB message will identify itself as a SMB message, also will inform the receivers which command is included, and its context. Meanwhile the status also is part of it. According to pa2.3, SMB header is fixed length of 32-bytes. Protocol (4 bytes): This field MUST contain the 4-byte literal string ’\xFF’, ’S’, ’M’, ’B’, with the letters represented by their respective ASCII values in the order shown. In the earliest available SMB documentation, this field is defined as a one byte message type (0xFF) followed by a three byte server type identifier. [8] Each SMB server or client will use this part to identify each others.

Command sector is a 8-bytes length structure. In the last version of SMB, version 3, there are 26 current used commands[8], we will discuss their details in later section.

There are two kinds of flags. The first one of them indicates different features in effect for the message. Flags2 is a 16-bit field of 1-bit flags that represent various features in effect for the message. Unspecified bits are reserved and MUST be zero.[8]

Tree ID (TID): The TID is a 16-bit number that identifies which resource (disk share or printer, typically) this particular CIFS packet is referring to. When packets are exchanged which do not have anything to do with a resource, this number is meaningless and ignored.

If a client wishes to gain access to a resource, the client sends a CIFS packet with the command field set to SMB_COM_TREE_CONNECT_ANDX. In this packet, the share or printer name is specified (i.e. \\SERVER\DIR). The server then verifies that the resource exists and the client has access, then sends back a response indicating success. In this response packet, the server will set the TID to any number that it pleases. From then on, if the client wishes to make requests specific to that resource, it will set the TID to the number it was given.

Process ID (PID): The PID is a 16-bit number that identifies which process is issuing the CIFS request on the client. The server uses this number to check for concurrency issues (typically to guarantee that files will not be corrupted by competing client processes).

User ID (UID): The UID is 16-bit number that identifies the user who is issuing CIFS requests on the client side. The client must obtain the UID from the server by sending a CIFS session setup request containing a username and a password. Upon verifying the username/password, the server responds to the session setup and includes a generated UID. The client then uses the assigned UID in all future CIFS requests. If any of these client requests require file/printer permissions to be checked, the server will verify that the UID in the request has the necessary permissions to perform the operation.

A UID is valid only for the given NetBIOS session. Other sessions could potentially be using an identical UID that the server correlates with a different user. Note: if a server is operating in share level security mode (see above), the UID is meaningless and ignored.

Multiplex ID (MID): The MID is a 16-bit value that is used to allow multiple outstanding client requests to exist without confusion. Whenever a client sends a CIFS packet, it checks to see if it has any other unanswered requests pending. If it does, it insures that the new request will have a different MID then the previously outstanding requests. Whenever a server replies to a CIFS request, it insures that the response it sends matches the request MID that it received. In following this procedure, the client can always know exactly which outstanding request an

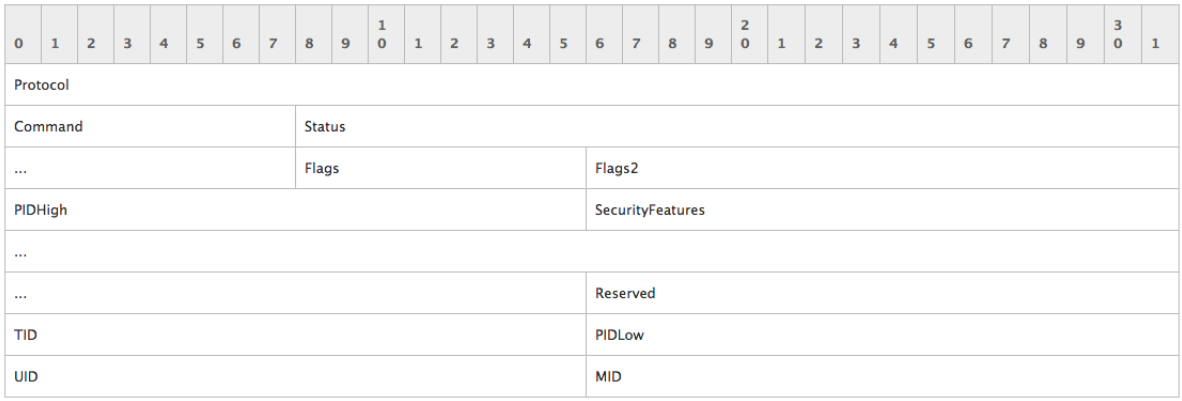

Figure 2.3: SMB header[8]

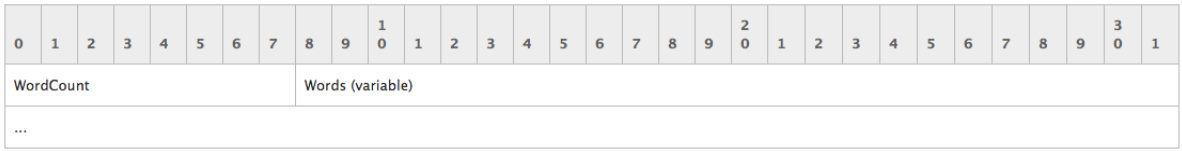

Figure 2.4: SMB parameter[8]

incoming reply is correlated to.

- The SMB parameter block

SMB was originally designed as a rudimentary remote procedure call protocol, and the parameter block was defined as an array of "one word (two byte) fields containing SMB command dependent parameters". In the CIFS dialect, however, the SMB_Parameters.Words array can contain any arbitrary structure. The format of the SMB_Parameters.Words structure is defined individually for each command message. The size of the Words array is still measured as a count of byte pairs. The general format of the parameter block is as follows. 2.4

WordCount and parameter words: CIFS packets use these two fields to hold command-specific data. The CIFS packet header template above cannot hold every possible data type for every possible CIFS packet. To remedy this, the parameter words field was created with a variable length. The wordcount specifies how many 16-bit words the parameter words field will actually contain. In this way, each CIFS packet can adjust to the size needed to carry its own command-specific data.

The wordcount for each packet type is typically constant and defined in the CIFS1.0 draft. There are two wordcounts defined for every single command; one wordcount for the client request and another for the server response. Two counts

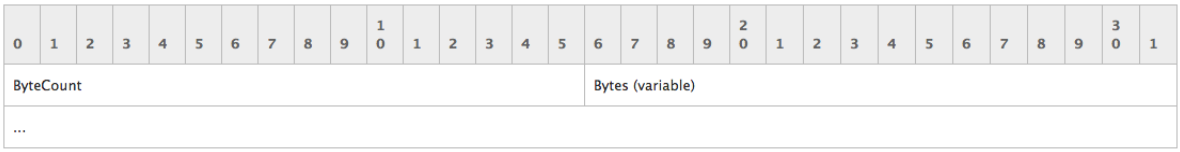

Figure 2.5: SMB data block[8]

are needed because the amount of data necessary to make a request is not necessarily the same amount needed to issue a reply.

ByteCount and buffer: These fields are very similar to the wordcount and parameter words fields above; they hold a variable amount of data that is specified on a per packet basis. The bytecount indicates how many bytes of data will exist in the buffer field that follows.

The major difference between the parameter data section above and the buffer is what type of data they store. The parameter words data section typically holds a small number of packet options, while the buffer data section typically holds large amounts of raw data (e.g. file data).

- The SMB data block

The general structure of the data block is similar to that of the Parameter block, except that the length of the buffer portion is measured in bytes.2.5

SMB Message process detail [30] Since SMB is a network based protocol, the way to locate others on network is the first step. In SMB server can be founded or located by three method.

- NetBIOS (Network Basic Input/Output System) names
- DNS names
- IP addresses

DNS names and IP address are the most common network knowledge, so we will only review NetBIOS details in this paper.

NetBIOS is an interface specification for access to network services, such as name-to-address resolution and sending and receiving data. The purpose of it is to isolate the application program from the actual type of hardware used in the LAN. It also spares the application programmer the details of network error recovery and low level message addressing or routing. As the another two protocol, NetBIOS will be required as the session start, before the connection is established.

To establish a connection between client and server, there are three phases, initial contact, login, and tree connect:

When a CIFS client try to access resources on a CIFS server, it will process the following packets in sequence. The NetBIOS session is established at first in order to provide a reliable message sequence transport service. Then, the client and server negotiate the CIFS dialect in which version they will use. The client then try to login to the server, sending its username and password. Then if server verified the coming user has the rights to access this shared storage, then the connection is established.

Packet #1 request, client –► server Command id: SMB_COM_NEGOTIATE. Purpose: Establish NetBIOS session Summary: The client, wishing to exchange CIFS packets with the server, initiates a NetBIOS session between itself and the server (referred to as "calling the server" in the previous NetBIOS section). This provides for sequenced, reliable message delivery between the two endpoints. Note that the client must know the server's NetBIOS name in order to call it and also must indicate its own NetBIOS name.

The events to establish the NetBIOS session are as follows. First, the client establishes a full duplex TCP connection with the server on port 139. Once this is accomplished, the client builds and sends a NetBIOS session request packet (not diagrammed in the NetBIOS section above, but described in RFC1002) over the TCP connection. In summary, the session request packet contains the client's NetBIOS name, the server's NetBIOS name, and an integer constant which indicates the packet's purpose is to establish a NetBIOS session. Please see RFC1002 for more details.

Packet #2 response, server –► client Purpose: Positive NetBIOS session acknowledgement Summary: If the above session request packet contained the server's NetBIOS name, and the packet was formatted correctly, the server replies with a simple session established acknowledgement. This 4-byte packet is also described in RFC1002. In summary, it indicates either a successful session establishment or an error code.

Packet #3 request, client –► server Purpose: Negotiate CIFS dialect Summary: Now that the NetBIOS session is established, the client is ready to send the first real CIFS request. The client sends the SMB_COM_NEGOTIATE command and includes a list of CIFS dialects that it understands. Packet: Command: SMB_COM_NEGOTIATE (0x72) TID: Ignored in this packet. PID: Set to process ID of client process. UID: Ignored in this packet. MID: Any unique number. WordCount: 0 ParameterWords: There are none because wordcount is 0. Bytecount: Set to 119 (variable depending on how many CIFS dialects the client understands). Buffer: Contains 119 bytes worth of dialect descriptions, examples would be as follows: "PC NETWORK PROGRAM 1.0", "MICROSOFT NETWORKS 3.0", "DOS LM1.2X002", "DOS LANMAN2.1", "Windows for Workgroups 3.1a", "NT LM 0.12".

Packet #4 response, server –► client Purpose: Choose CIFS dialect from request list Summary: The server is now responding to the negotiate protocol request by selecting the dialect that it wishes to communicate in. Packet: Command: SMB_COM_NEGOTIATE (0x72) TID: Ignored in this packet. PID: Ignored when packet is from server. UID: Ignored in this packet. MID: matches unique number chose above. WordCount: This number depends on the dialect that is chosen. For this example, we will assume that the server chose "NT LM 0.12" [8] . In this case, the wordcount is 17. ParameterWords: The 17 words contained here indicate the chosen dialect and many server properties. Of note is the MaxMpxCount (which states the max number of pending requests the client can initiate) and the 32-bit capabilities flags (which indicate if UNICODE is supported, if large files are supported, if NT commands are supported, and more). Bytecount: Variable, usually greater than 8. Buffer: Typically contains an 8-byte random string that the client uses in the next packet for encryption purposes.

Packet #5 request, client –► server Purpose: User login Summary: Now that the CIFS dialect has been agreed upon, the client sends a packet containing a username and password to gain a user ID (UID). This packet also relays client capabilities to the server, so the packet must be sent even if the server is using share level security. Packet: Command: SMB_COM_SESSION_SETUP_ANDX (0x73) TID: Ignored in this packet. PID: Set to process ID of client process. UID: Ignored in this packet. MID: Any unique number. WordCount: 12 ParameterWords: This section is very similar to the server's negotiate protocol parameter words response. However, instead of listing the server's capabilities, it lists the client's. It also contains the size of the passwords to be supplied in the buffer section below. Bytecount: Variable, the buffer below contains the encrypted password, the username, the name of the operating system and the native LAN manger. Therefore, the size listed here depends on the string sizes of all these entities. Buffer: As mentioned above, this field actually contains the password, username, and other strings that identify the operating system involved.

Packet #6 response, server –► client Purpose: Indicates User ID (UID) or returns error if bad password Summary: Once the server receives the encrypted password and username, it checks if the combination is valid. If the password is invalid, this response packet will be returned with the error class and code set to the appropriate error value. If the username/password is correct, then this packet contains the UID that the client will begin to send with every packet from here on. Packet: Command: SMB_COM_SESSION_SETUP_ANDX (0x73) TID: Ignored in this packet. PID: Ignored when packet is from server. UID: The 16-bit number that the server has assigned to represent client user identity. MID: Matches unique number chose above. WordCount: 3 ParameterWords: Nothing relevant to normal operation. Bytecount: Variable, the buffer below contains strings stating the server OS and native LAN manager type. Buffer: Contains

strings indicating the server OS and LAN manager type.

Packet #7 request, client –► server Purpose: Connect to particular resource Summary: At this point, the client has authenticated itself to the server and may proceed to connect to the actual share. In this packet, the client specifies the share that it wishes to access. Share names are specified in UNC format (i.e. \\SERVER\SHARE). Packet: Command: SMB_COM_TREE_CONNECT_ANDX (0x75) TID: Ignored in this packet. PID: Set to process ID of client process. UID: Set to the server returned UID from the above session setup response. MID: Any unique number. WordCount: 4 ParameterWords: Nothing relevant to normal operation. Bytecount: Variable, depends on the size of the UNC string that is requested below. Buffer: Contains the share name that the client wishes to access.

Packet #8 response, server –► client Purpose: Indicates Tree ID (TID) or error if share name does not exist Summary: If the share specified above exists and the user has access permission, then the server returns a successful response with the TID set to the number it wishes to refer to the resource as. If the share does not exist or the user does not have access permission, the server will return the appropriate error class and error code here. Assuming that this packet indicates success, the client now has everything it needs to access files from the specified share. This is the final packet in this client/server exchange. Packet: Command: SMB_COM_SESSION_SETUP_ANDX (0x73) TID: 16-bit number which server has assigned to represent the requested resource. PID: Ignored when packet is from server. UID: 16-bit number representing the user. MID: Matches unique number chosen above. WordCount: 3 ParameterWords: Nothing relevant to normal operation. Bytecount: Variable, the buffer below contains strings stating the native file system and device type of the requested resource. Buffer: Contains strings that state the native file system and device type.

Then it is the process of file open and read.

Once a client has completed the initial packet exchange sequence described above, it may open and read files from the share that was requested. The file open consists of one CIFS request and one CIFS response. The read request also consists of one request and one response packet.

Packet #1 request, client –► server Purpose: Open a file Summary: In order to read or write to a file, it first must be opened. This CIFS packet does exactly that. Packet: Command: SMB_COM_OPEN_ANDX (0x2D) TID: Set to the server returned TID from the tree connect response above. PID: Set to process ID of client process. UID: Set to the server returned UID from the session setup response above. MID: Any unique number. WordCount: 15 ParameterWords: Specifies many open options such as mode (read, write, or readwrite) and sharing mode (none, read, write). Bytecount: Variable, depends on the size of the string that contains the filename. Buffer: Contains the name of the file to be opened.

Packet #2 response, server –► client Purpose: Indicate File ID, or error code if problem Summary: The server checks to see if the filename above exists and if the user specified in the UID has permission to access the file. If these conditions are not met, the server will return the appropriate error class and error code indicating what that problem is. If there are no errors, the server returns a response packet that includes a File ID (FID) that can be used in subsequent packets for accessing the file. Note that the FID is returned to the client in the parameter words field of the response. There is no FID field in the standard CIFS header. Packet: Command: SMB_COM_OPEN_ANDX (0x2D) TID: 16-bit number which the server assigned to represent the requested resource. PID: Ignored when packet is from the server. UID: 16-bit number representing the user. MID: Matches unique number chosen above. WordCount: 15 ParameterWords: Many flags indicating what type of actions occurred and the very important 16-bit FID. Bytecount: 0 Buffer: No data in buffer.

Packet #3 request, client –► server Purpose: Read from a file Summary: Assuming that the above response indicated a FID for the client to use, an actual read request for file data can now be issued. Packet: Command: SMB_COM_READ_ANDX (0x2E) TID: Set to the server-returned TID from the tree connect response above. PID: Set to process ID of client process. UID: Set to the server-returned UID from the session setup response above. MID: Any unique number. WordCount: 10 ParameterWords: Here, the FID is stated so the server knows which opened file the client is referring to. Also indicated here are a 32-bit file offset and a 16-bit count value. These two numbers dictate where and how much data to return from the file. Bytecount: 0 Buffer: No data in buffer.

Packet #4 response, server –► client Purpose: Returns file data requested Summary: This packet contains the requested file data. Assuming the UID, TID, and FID were all valid numbers in the request, an error here should be unlikely. Packet: Command: SMB_COM_READ_ANDX (0x2E) TID: 16-bit number which server has assigned to represent the requested resource. PID: Ignored when packet is from the server. UID: 16-bit number representing the user. MID: Matches unique number chosen above. WordCount: 12 ParameterWords: Here, the number of bytes that were actually read is indicated. This does not necessarily match the number requested (in case the request exceeded the file boundary). Bytecount: Variable, the buffer holds the file data, so this number is also the number of bytes that were actually read. Buffer: The file data requested.

Then it is the process of file open and write.

Once a client has completed the initial packet exchange sequence described above, it may open and write files from the share that was requested. The file open consists of one CIFS request and one CIFS response. The read request also consists of one request and one response packet.

Packet #1 request, client –► server Purpose: Open a file Summary: In order to read or write to a file, it first must be opened. This CIFS packet does exactly that. Packet: Command: SMB_COM_OPEN_ANDX (0x2D) TID: Set to the server returned TID from the tree connect response above. PID: Set to process ID of client process. UID: Set to the server returned UID from the session setup response above. MID: Any unique number. WordCount: 15 ParameterWords: Specifies many open options such as mode (read, write, or readwrite) and sharing mode (none, read, write). Bytecount: Variable, depends on the size of the string that contains the filename. Buffer: Contains the name of the file to be opened.

Packet #2 response, server –► client Purpose: Indicate File ID, or error code if problem Summary: The server checks to see if the filename above exists and if the user specified in the UID has permission to access the file. If these conditions are not met, the server will return the appropriate error class and error code indicating what that problem is. If there are no errors, the server returns a response packet that includes a File ID (FID) that can be used in subsequent packets for accessing the file. Note that the FID is returned to the client in the parameter words field of the response. There is no FID field in the standard CIFS header. Packet: Command: SMB_COM_OPEN_ANDX (0x2D) TID: 16-bit number which the server assigned to represent the requested resource. PID: Ignored when packet is from the server. UID: 16-bit number representing the user. MID: Matches unique number chosen above. WordCount: 15 ParameterWords: Many flags indicating what type of actions occurred and the very important 16-bit FID. Bytecount: 0 Buffer: No data in buffer.

Packet #3 request, client –► server Purpose: Write to a file Summary: This request is used to write bytes to a regular file, a named pipe, or a directly accessible I/O device such as a serial port (COM) or printer port (LPT). Packet: Command: SMB_COM_WRITE_ANDX (0x2F) TID: Set to the server-returned TID from the tree connect response above. PID: Set to process ID of client process. UID: Set to the server-returned UID from the session setup response above. MID: Any unique number. If the client negotiates the NT LAN Manager dialect or later the client SHOULD use the 14-parameter word version of the request, as this version allows specification of 64-bit file offsets. This is the only write command that supports 64-bit file offsets.

Packet #4 response, server –► client Purpose: Returns write result Summary: This packet contains the write result status. WordCount (1 byte): This field MUST be 0x06. The length in two-byte words of the remaining SMB_Parameters.AndXCommand (1 byte): The command code for the next SMB command in the packet. This value MUST be set to 0xFF if there are no additional SMB command responses in the server response packet. AndXReserved (1 byte): A reserved field. This MUST be set to 0x00 when this response is sent, and the client MUST ignore this field. AndXOffset (2 bytes): This field MUST be set to the offset in bytes from the start

of the SMB Header (section 2.2.3.1) to the start of the WordCount field in the next SMB command response in this packet. This field is valid only if the AndX-Command field is not set to 0xFF. If AndXCommand is 0xFF, this field MUST be ignored by the client. Count (2 bytes): The number of bytes written to the file. Available (2 bytes): This field is valid when writing to named pipes or I/O devices. This field indicates the number of bytes remaining to be written after the requested write was completed. If the client wrote to a disk file, this field MUST be set to 0xFFFF.<63> Reserved (4 bytes): This field MUST be 0x00000000. ByteCount (2 bytes): This field MUST be 0x0000. No data is sent by this message.

2.2 Benchmarking

Benchmarking technology is widely used by all industries. For example, mobile manufacturers will post their benchmarking scores for every new device they post. By reading those scores users or customers could evaluate their performance. It applies for storage system. Unlike a mobile phone, storage systems usually work under huge pressures, accessed by multiple servers, and meanwhile storage system is the slowest components compare with CPU, memory. The same methodology applies for system permanence test.

2.2.1 Network benchmarking

In his 1999 HotOS paper, Mogul insisted that benchmarks must predict absolute performance in a production environment, rather than simply focusing on quantified, repeatable results in a carefully constructed laboratory setting [10]. One should generate synthetic workload based on the current environment. By analyzing original workload module, tools could adjust itself to simulate the TCP and UDP packets. There are several systems apply trace and replay methodology, for example, TCPivo. They use tcpdump to catch the original workload, and use TCPivo to reproduce the synthetic workload. The trace and replay skills they use in such system, that is very similar with the system we are using in SAMBA benchmarking.

2.2.2 Storage benchmarking

Although many manufacturers offer SSD (Solid State Drive) for better performance, but since the price of it is almost ten times then traditional hard drive disk, so in most cases customers have to use HDD (Hard Drive Disk) as their primary storage system. So the performance results will affect users' finial decision a lot.

But it is not that easy to benchmark a target storage system. Because storage systems are in the bottom layer of OS kernel. It will be affected by all up-layers facts, such as kernel daemons, applications access patterns, properties of files. Some storage systems are optimized for one or more applications such as DB(Data Base), or video applications. So it is hard to generate a universal workload to test all target storages.
By AVISHAY [1] there are three popular methods to evaluate a storage system, they are

- Macro-benchmarks.
- Trace-Based.
- Micro-benchmarks.

For macro-benchmarks, one would employee this method under general purpose. Which means the workload or pressure generate by this kind of tools will represent standard requirement. That usually will not suitable for one's interest, because most time the real workload in their product environments are unique.

Micro-benchmark is a adjustment for macro-benchmark, it will change some configurations or embedded with different types of operations to highlight one or more aspects in that one. This requires a solid understand of current environments in order to adjust a proper simulate by optimization all operations. But it is still hard to say whether the two workloads, the synthetic workload and the original workload, are the same. One could always argue with some other aspects will also affect the result in someway.

Trace-Based tools is divided benchmarking with two steps. The first one is trace. The tool will learn the current environment by some method at first. Then replay it according to the pattern we learned from the first step. By doing these steps, we could generate the simulation more accurate as original workload, meanwhile the results are more reliable for anyone depends on them. But to achieve that goal, one need to solve two challenges, one is how to collect the trace, and how to replay it efficiently.

There are two types of tracing. The first one is to capture system calls. One can learn about application dependencies between file system operations. The second type of tracing is to capture NAS protocol packets from network. Compare with first type, the advantage of the second one is that, it will no affect or lower the system performance when running capture tools. But it is difficult for user to know about application dependencies and application think time from network capture.

2.3 Related work

In the paper of AVISHAY's[1], it surveyed 415 file system and benchmarks from 106 papers. It introduces all the currently used benchmarking methodologies, and also it argued how to present and discuss the test results. It categories the three kinds of benchmark system.

- Macrobenchmarks
- Trace Replays
- Microbenchmarks

Macrobenchmarks is that one test storage system against a particular workload. The workload can usually represent some real industrial workload. The advantage of this method is that it is good for overall view of the storage system and easy to implement. But the tradeoff is that the result maybe not reliable, since the workload may not be realistic. Postmark[13, 14] is one of the most famous benchmark tools in this category. SPEC (The Standard Performance Evaluation Corporation)[16], TPC (The Transaction Processing Performance Council)[15] and SPC (The Storage Performance Council)[17] are three organizations which focusing on macorbenchmarks tools and workloads development.

Microbenchmarks test the same storage system serval time, and modify a variable or some variables each time. So one can isolate the bottleneck from the system. Bonnie++[18] is a fairly well-known benchmark tool. It performs a series test on a single file. Based on the test, it reports the process capacity per second for CPU and the percent of CPU usage.

They defines the benchmark tools, in the "trace-based" category, as "A program replays operations which were recorded in a real scenario, with the hope that it is representative of real-world workloads." It is critical to generate an identical synthetic workload of the original workload. Recent studies point out that storage workloads are diverse. They vary widely from different applications they serve. Therefor how to measure a workload is important. This paper[1] is our guid book for the entire system, since it listed all the key variables to evaluate a storage system by a benchmarking tool.

For all benchmarking systems belong to this category, they trace the workload first and analyze the workload, at last step they generate the synthetic workload based on previous analyze results.

- Trace capture: There is no accepted way to capture traces. Traces can be captured at the system call, networking, and driver levels.
  - System call: It is easy and the system call API is portable[19, 20, 21], but the tradeoff is it adds extra work on the system.

  - Network traffic: Specialized tools[22, 23] only work for network based storage protocol, and the trace file only contain requests that were not satisfied from the cache.
  - Driver level: Driver-level traces contain only non-cached requests. But it is unable to correlate the requests with the associated meta-data[24].

- Replay: Some extra work is required before one can generate a synthetic workload successfully, and replay method should be chosen carefully.

  - Extra work
    * Target file system must be prepared.
    * Missing operations have to be guessed[3].
  - Replay method
    * Replay level: In the most common case, one should replay traces at the level the traces were captured. Network replaying can be done entirely from the user-level.
    * Replay speed: Many believes that the trace should be played as fast as possible[25, 26, 27, 28]. The timings of the original workload should be ignored. But there are replaying tools, such as Buttress[29] , that have been shown to follow timings accurately.

According to Christina et. [2], the workload pattern of block device can be identified with Markov Chain module. They implemented the trace-based benchmarking tool for IP-SAN. In their model they used the Markov Chain to represent the characteristics of the original workload. Characteristics correspond to ranges of logical blocks on disk (LBN). Transitions related to the probabilities of switching between LBN ranges. Transition is characterized by block size, randomness, type of IO and inter-arrival between subsequent request. The probabilities for transitions is calculated as the percentage of correspond I/Os. But they did not discuss about other type of storage system, they mainly introduced about SAN or IP SAN structure.

In the paper of Ningning[3], they also implemented their benchmark system according to trace and replay methodology. They focused on NFS. They use the network sniff to capture original workload. Then they create initial file system image according to the NFS trace. Their approach is similar compare with ours. Instead of benchmarking NFS system, we implement a SMB benchmark system.

Chapter 3

Method

This chapter will introduce the reader to the methods, tools and equipment used in this project.

3.1 System model

As explained in the introduction, one will need two subsystems to benchmark the target storage. First it requires to trace the current system workload.

The SMB/CIFS protocol transfers all information through network. We trace all network packets from different network structure. But a single point to point network structure is enough for us to evaluate our results and design.

The first subsystem is learning system. Two major components are involved in it. The first part is product storage, which is responsible for the current production data access requirements. The second one is the learning system, with this system, one grabs the SAMBA/CIFS data flow by copying the TCP traffic flow from the port connected with storage. The port number for SAMBA/CIFS service is TCP 445. It shows in Figure 3.1

The learning system is running a Ubuntu 13.04 server version. The system will use TCPDUMP to track the input and output traffic on product storage. The output of the learning system is a pcap file. After data collection, the learning system will also response for generating a analyze report for previous collected workload, and an operation list. The goal of this step is that one can learn all the characteristics of the workload. As mentioned before, the key for generating a realistic synthetic workload is to trace the all characteristics of original one and replay according to it.

We named the second system as simulation system, it will simulates the workload based on the operation list, that is generated by learning system. We employed the second system at two stage.

1. Under the different environments, one should evaluate our system first before one can use it as a benchmarking system. We discussed the reason in Chapter Discussion. The key to evaluate our system is to compare the synthetic workload with the original workload. The two workloads should have the same characteristics. After all the informations of the original workload are learned by learning system, a synthetic workload is going to be made. At this stage, the synthetic workload is used to test in the original environment again, which means we test the product storage with synthetic workload, and all network informations are captured in the same way by learning system. Then the two workloads are compared by their characteristics.

2. Once one has evaluated the system is capable to generate the similar workload as the original one, then the system is used to benchmark the multiple NAS storage systems. In this stage, the synthetic workload is used to test multiple target storage systems. Once the simulator start to generate the pressure for target storage, monitor will run again to dump the synthetic workload. It shows in Figure 3.2. Unlike the previous step, the characteristics are identical for all the synthetic workloads, since we use exactly the same workload for all targets. Then we only compare their reply times for each NAS storage system. The reply times represent their performances.

3.2 Tools and equipment

Our system is developed in a virtual environment, and the network structure is a point to point connection. But all of our tools are capable in larger and complicator environment. We picked up the current system configuration according to our resource and time limitations.

3.2.1 Workload trace

To trace the workload, one needs to use switch port forward technical on switch. Switch port forward technical is also known as port mirror, it is generally used by networking troubleshooting. By setting up port mirroring on switch, one can receive all packets on the port connected with storage. Most of modern switch devices support this function. The port mirror feature was introduced on switches because of a fundamental difference between switches and hubs. A hub broadcasts a packet to all ports whenever it receives it on one, but it will not send it to the one that receives it. Instead of broadcasting packets among all ports, the switch will create a forwarding table on the physical layer, based on MAC address. Based on

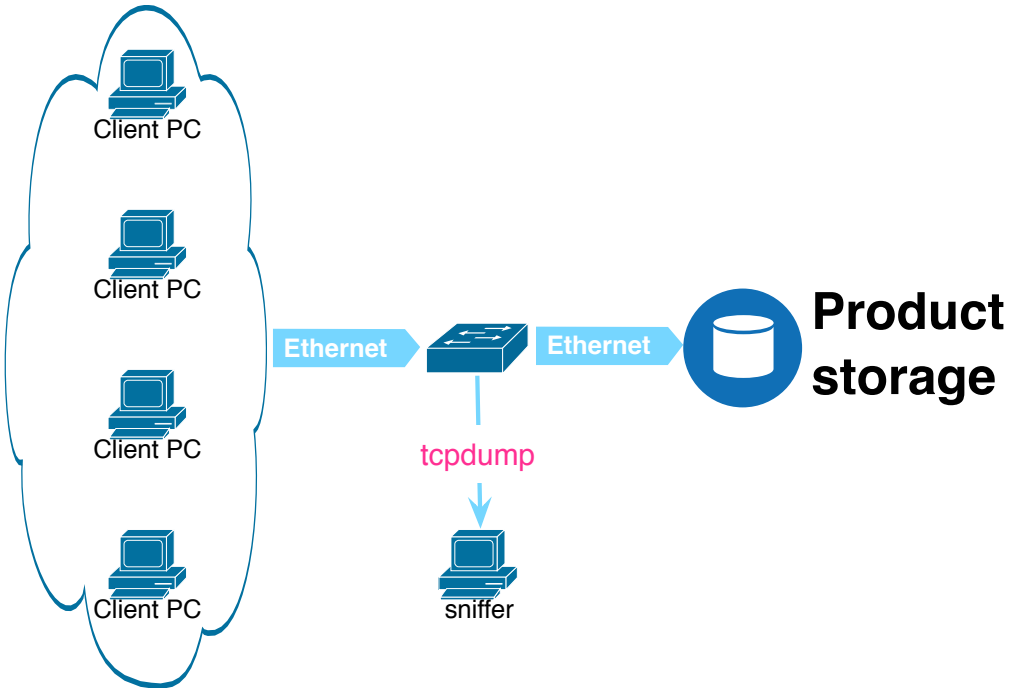

Figure 3.1: Learning system structure

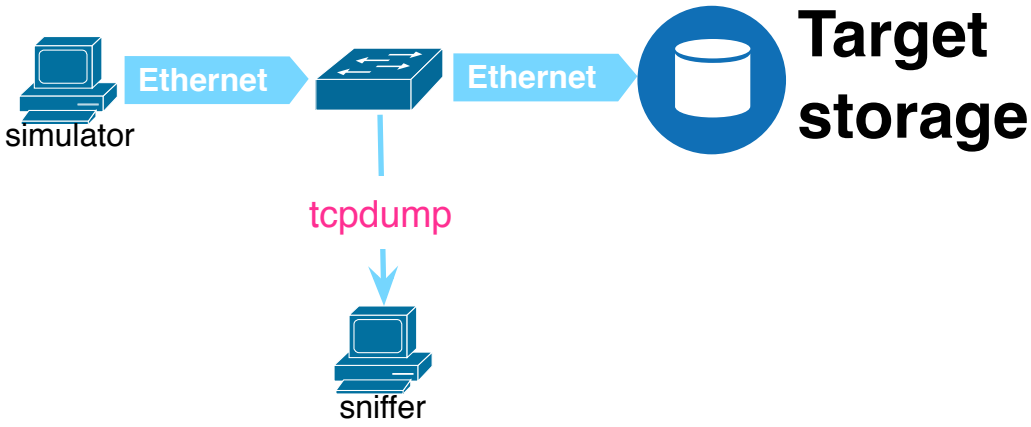

Figure 3.2: Simulation system structure

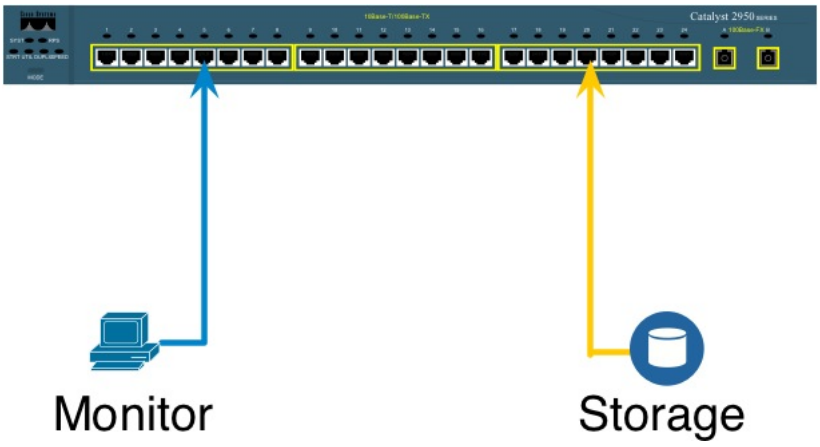

Figure 3.3: Port mirror from port 20 to port 5

the forwarding table, the switch sends packets to destination port directly without notifying others.

For larger system, one need to mirror all the ports connected to storage system to the learning system, and combine all the dumped files into one file. But in our experiment we only need to monitor the traffic on one port by listening on another port. One requires that when switch receives a packet on source port it will send out a copy to monitor port. In our case, we are using Cisco 2950 switch. The port mirror function is called SPAN (Switch port analyzer) by Cisco. As shown in an example in Figure 3.3, one receives a traffic copy of port 20 on port 5.

Next a tool for collecting data from the network is needed. Tool is introduced in background chapter, tcpdump. Tcpdump is a packet capture tool which runs on linux and Unix systems. By default it prints out a description of the head information of packets on a network interface. We use it for two major reasons: the first one is that tcpdump can save the packets to a file for later analyze in both small and large system, the second reason is that one could use tcpdump to filter out all unnecessary packets. The workload of SAMBA/CIFS traffic exists on TCP port 445 by default. Because of this, one only needs to dump TCP traffic on port 445.

3.2.1.1 System setup and configuration

In this section, we listed all the commands for setting up the learning system. As mentioned, one need to setup switch and then dump the network traffic.

1. Cisco 2950 Switch

    (a) Switch#config terminal

(b) Switch(config)#monitor session 1 source interface fastEthernet 0/25

(c) Switch(config)#monitor session 1 destination interface fastEthernet 0/5

2. Tcpdump

(a) root# tcpdump port 445 -w oringinalworkload.pcap

3.2.2 Capture file analyze

The capture file is a standard pcap (packet capture) file. We use tshark to analyze them. Tshark is known as text version of wireshark, which is one of the best packet analyzers. Tshark is similar with tcpdump but has some integrated sorting and filtering options. We take advantages from SMB filtering options of tshark. Tshark enable us to filter SMB traffic by SMB commands, and it offers a statistic result of reply time for each correspond requests. In our system, we require only write and read workload between server and clients. For each request we need to know the timestamp for this packet, start offset, length for this request, Fid and file name. One can collect all of those necessary information from tshark output as **??** shows. Besides that we also require reply time for this request, the result of reply time is used for later comparison.

One could specify the SMB command by "smb.cmd== " options. As we listed in Introduction part, the command id for read is 0x2e, and it is 0x2f for write request. By extend normal tshark command with smb.cmd==0x2e(Read request) options, tshark generates the results as shown in following log. The output of smb.cmd==0x2f (Write request) is very similar with previous one, but it shows Write AndX Request in command field. The log file is discrete log, it combines both read requests and write requests, and their replies. But this log can present all the information we used in later phrase.

Log of tshark output:

21461 53.761943 10.0.0.40 -> 10.0.0.2 SMB 117 Read AndX Request, FID: 0x2407, 61440 bytes at offset 14987264 smb.file == "\\a.rvt"

21462 53.761969 10.0.0.40 -> 10.0.0.2 SMB 117 Read AndX Request, FID: 0x2407, 61440 bytes at offset 10854400 smb.file == "\\a.rvt"

21463 53.761971 10.0.0.40 -> 10.0.0.2 SMB 117 Read AndX Request, FID: 0x2407, 61440 bytes at offset 12951552 smb.file == "\\a.rvt"

21465 53.762280 10.0.0.2 -> 10.0.0.40 SMB 237 Read AndX Response, FID: 0x2407, 61440 bytes smb.time == 0.000337000 smb.file == "\\a.rvt"

21480 53.764403 10.0.0.2 -> 10.0.0.40 SMB 237 Read AndX Response, FID: 0x2407, 61440 bytes smb.time == 0.002434000 smb.file == "\\a.rvt"

21489 53.765769 10.0.0.2 -> 10.0.0.40 SMB 237 Read AndX Response, FID: 0x2407, 61440 bytes smb.time == 0.003798000 smb.file == "\\a.rvt"

44047 92.872010 10.0.0.40 -> 10.0.0.2 SMB 1418 Write AndX Request, FID: 0x23d6, 65536 bytes at offset 208142336 smb.file == "\\a.rvt"

Column 1	Sequence number of the packet	It indicates the order of the packet
Column 2	Time stamp of the packet	Time stamp
Column 3	Source and Destination	The IP addresses of Server and Client
Column 4	Operation type	The request type of this packet
Column 5	FID	A file handle, representing an open file on the server.
Column 6	Length	Requested length
Column 7	Offset	Offset of the request
Column 8	Requested file	File name

Table 3.1: Variables for Requests

44049 92.872060 10.0.0.40 -> 10.0.0.2 SMB 1418 Write AndX Request, FID: 0x23d6, 65536 bytes at offset 208207872 smb.file == "\\a.rvt"

44051 92.872495 10.0.0.40 -> 10.0.0.2 SMB 64294 Write AndX Request, FID: 0x23d6, 65536 bytes at offset 208273408 smb.file == "\\a.rvt"

44053 92.872716 10.0.0.2 -> 10.0.0.40 SMB 105 Write AndX Response, FID: 0x23d6, 65536 bytes smb.time == 0.001794000 smb.file == "\\a.rvt"

44057 92.873066 10.0.0.2 -> 10.0.0.40 SMB 105 Write AndX Response, FID: 0x23d6, 65536 bytes smb.time == 0.001056000 smb.file == "\\a.rvt"

44058 92.873170 10.0.0.2 -> 10.0.0.40 SMB 105 Write AndX Response, FID: 0x23d6, 65536 bytes smb.time == 0.001110000 smb.file == "\\a.rvt"

We listed all the variables of requests in the Table 3.1. The time stamp is used for us to simulate the inter-arrival times between requests. Since in this thesis, we only simulate the situation of a point to point connected server and client, the addresses of the destination and the source are dropped. The operation type is used to category all requests by their type. In later stage one need to call different subroutines for different types of operation. As one can see, a single file has multiple FIDs. The FID is representing an open file on server, in that case, the multiple FIDs for a single file means that the file is opened multiple times by a client or different clients.

Most of the variables we get from the reply packet are the same as the requests. But the reply time is only observed in reply packet. We use the reply time to evaluate the performance of a storage system.

To store all the arguments from this step, we employ perl to go through all packets of the pcap file. In our case, a pipe connects the tshark command output and our scripts. A pipe is a unidirectional I/O channel that can forward a stream of bytes from one process to another. In that case, all outputs of tshark will be redirect as input to our script. Then we use perl REGEX (Regular expression) function to dispatch each output, and store them in to correspond variables.

The analyze script is response to generate two files. Result report is the first one, it includes summarize of all statistic result of workload. It shows all details

Column 1	Sequence number of the packet	It indicates the order of the packet
Column 2	Time stamp of the packet	Time stamp
Column 3	Source and Destination	The IP addresses of Server and Client
Column 4	Operation type	The request type of this response
Column 5	FID	A file handle, representing an open file on the server.
Column 6	Length	Replied length
Column 7	Requested file	File name
Column 8	Time	The time between the response and its request

Table 3.2: Variables for Responses

of files that been accessed, which includes the average inter-arrival between each access, the total request data length for each file, average response time for all requests of each file. The analyze script generates an operation lists file as well. This file lists all requests and their arguments, for example timestamp for each request, command type etc. It is prepared for simulation part.

3.2.2.1 System setup and configuration

The command we used to get the informations of the packets.

1. Tshark command

   (a) For read request: sun@guang-ubuntu-server:~/code$ tshark -R "smb.cmd==0x2e" -r test3.copyeclipse.pcap

   (b) For write request: sun@guang-ubuntu-server:~/code$ tshark -R "smb.cmd==0x2f" -r test3.copyeclipse.pcap

2. Perl pipe command

   (a) open (r,"tshark -R 'smb.cmd==0x2e' -z proto,colinfo,smb.file,smb.file -z proto,colinfo,smb.time,smb.time -r $f|");

3.2.3 Simulation

In this part, two tools is required. Linux "dd" command is used to generate all files that listed in operation. The "dd" command works for converting and copying a file. The reason for using dd copy function as a file generator in our system is that, by supplying appropriate arguments for dd, we can generate file with accurate size

```
r       97.317654       0       27262976        0x310c  \\eclipse\\jre7\\lib\\rt.jar
r       104.991473      27394048        28999680        0x310c  \\eclipse\\jre7\\lib\\rt.jar
r       105.203621      30343168        51581545        0x310c  \\eclipse\\jre7\\lib\\rt.jar
```

Figure 3.4: Random read example

and random content. /dev/urandom is the source file for dd command. /dev/urandom is embedded by Linux as a character special file - it provides an interface to the kernel's random number generator.

In some case, a file is accessed randomly instead of sequentially. For example, as shown in 3.4, the file was read from three different start offset. To simulate it one should set start offset and length for each request. So a SAMBA client application is required to fulfill the requirements. We are using perl Filesys-SmbClient module. It is a interface for accessing Samba filesystem. One can simulate normal read and write access and move file offset of a file handler by calling this module. A request from Filesys-SmbClient module bypasses buffering, then the simulation operations can be spotted on network exactly the same as asked. This part is very important for simulator to generate a similar workload. Otherwise serval operations might be ignored because of they stored already in cache by previous request.

3.2.3.1 Oplist

Oplist is generated at the last step of the learning system. It contains all the information which are used by simulation system. As we mention in 3.2.2, the simulator should be able to generate a similar workload according to the key characteristics. The first column indicates the operation type of the request, and the second column indicates the time of the request should be generated. The time is elapse time since the first packet. The third and fourth column present the start offset and the length of the request. The last two column show the file information. The FID information is used to identify each open operations for a file. When we replay the synthetic workload according to this oplist file, one can easily tell when should an open operation should be invoked according to different FID.

3.2.3.2 System setup or configuration

1. dd

   (a) dd -if=/dev/urandom -of=/share-location-for-a-share/filename bs=filesize count=1

2. Filesys-SmbClient

(a) Create SAMBA client instance: my $smb = new Filesys::SmbClient(username => "", password => "", workgroup => "WORKGROUP");

(b) Open a file and save the file handle to a variable:

$a=$smb->open("smb://10.0.0.2/share2/eclipse/jre7/lib/fontconfig.properties.src");

(a) Sets FILEHANDLE's system position in 0: $smb->seek($a,0);

(b) Attempts to read 2 bytes of data into variable $a from the specified file handler $a: $smb->read($a,2);

Chapter 4

System design

In this chapter, we introduced the design of our system. We designed the system for two scenarios, the first is evaluation, the other is benchmarking. As mentioned in 3.1, our system has two major subsystems, learning system and simulation system. In this chapter we first explained the different approaches for the two scenarios and then we introduced the design and implement details for both subsystems .

4.1 Work flow introduction

By this tools, we intent to develop a light weight tools, which can both monitor the current Environment and generate the synthetic workload based on previous workload. According to Avishay ET.AL [1], they mentioned such method by "trace and replay", it is the method we are using in our system. The learning subsystem is used to analyze the

4.1.1 Evaluation

As shown in Figure 4.1, we divided our system into 7 modules, they are listed on the left side of Figure 4.1. They work together to fulfill the requirement of trace and replay benchmarking. The right side of Figure 4.1 is the outputs that generated by each module.

At this point, our goal is to evaluate the synthetic workload. We use the key characteristics of different workloads to compare them. At first one should learn all the characteristics of the original workload, then a synthetic workload is generated according to the informations of the original workload. Then we ran the synthetic workload on the same product environment again and collected the network traffics. After we analyzed the network dump file of the synthetic workload,

we summarized all the characteristics of the two workloads, and compare them. Since both time we use the same storage system, so the difference of those two workloads are caused by our system only.

We use the first three subsystems to learn from original workload. They include tracing function and an analyzer script. The learning system also generate three files as output, they are a pcap file, operations list file and a result report.

Then a simulation system is involved in next step. It replays a synthetic workload and it records the traffic information as learning system. Therefor its output is also a pcap log file.

At the last part of our system, a sub system called comparison system is employed, shown in Figure 4.4. It analyzes the capture file of simulator and generate a result report for synthetic workload. Then the system compares both reports of original workload and simulation workload.

4.1.2 Benchmarking

At this point, our goal is to benchmark different NAS storage systems with the synthetic workload. We use the key characteristics of different workloads to compare them. At first one should learn all the characteristics of the original workload, then a synthetic workload is generated according to the informations of the original workload. The synthetic workload is used for all the NAS storage systems. Each time the network traffic is collected, and the mean value and standard deviation of the reply time are calculated according to each network dump file as shown in Figure 4.5.

As the evaluation, We use the first three subsystems to learn from original workload. One need to collect longer period traffic then evaluation stage.

Then we use the simulation workload to benchmark all the NAS systems. It replays a synthetic workload several times and it records the traffic information as learning system. At the last part of our system, a sub system called comparison system is employed, shown in Figure 4.4. It analyzes the capture file of simulator and generate the mean value and the standard deviations of the reply times for each time test. The mean values of the reply times are used to evaluate the performance of the target NAS storage systems.

4.2 Learning system design

First we are focusing on trace part, in this part one critical mission is how to collect all the necessary information we need. After monitor the product environment for a while, we have to understand what are the workload characterizations. In case to replay it more accurate, we generate the simulation R/W operation as original

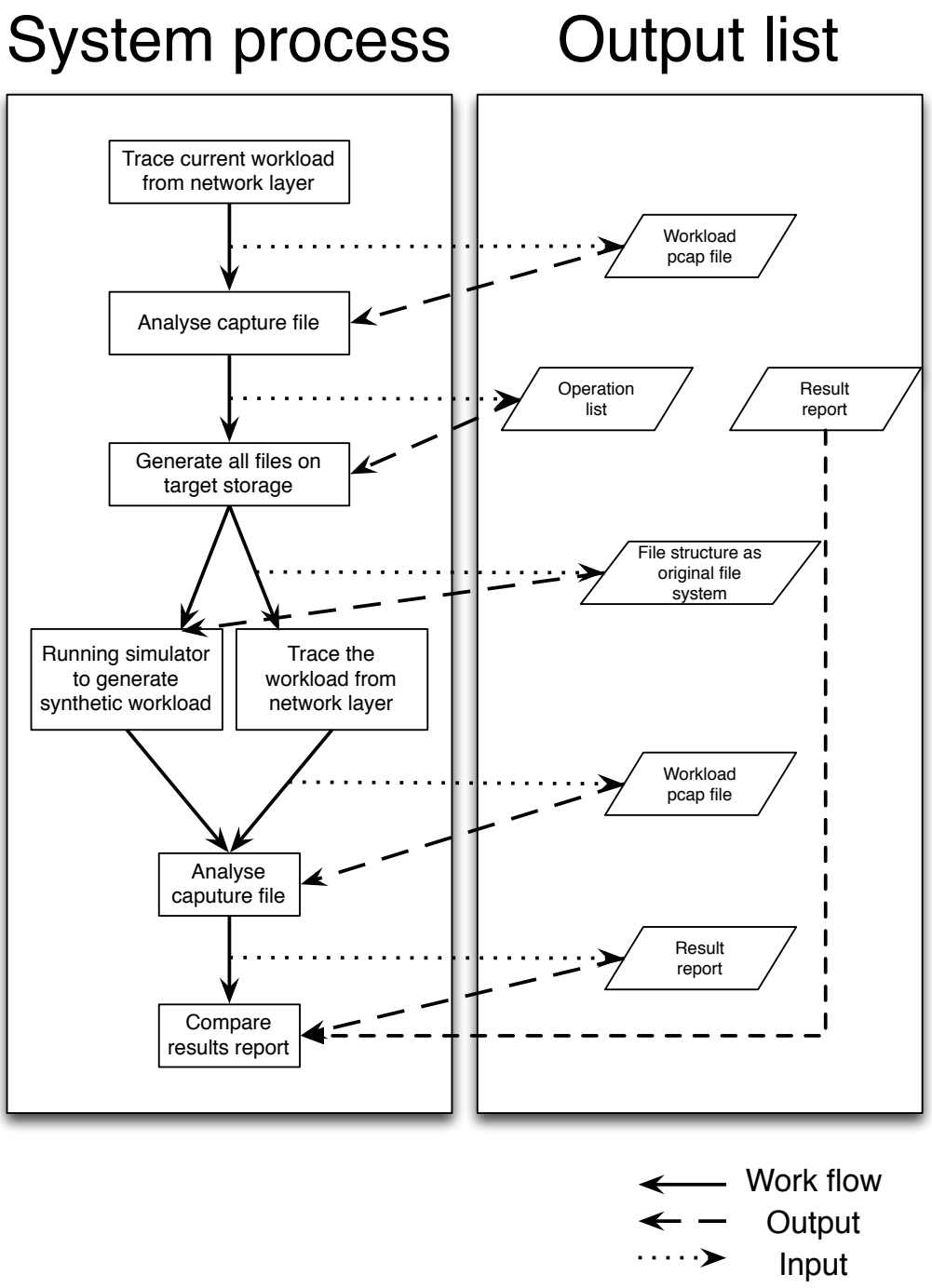

Figure 4.1: Design work flow for evaluation system[8]

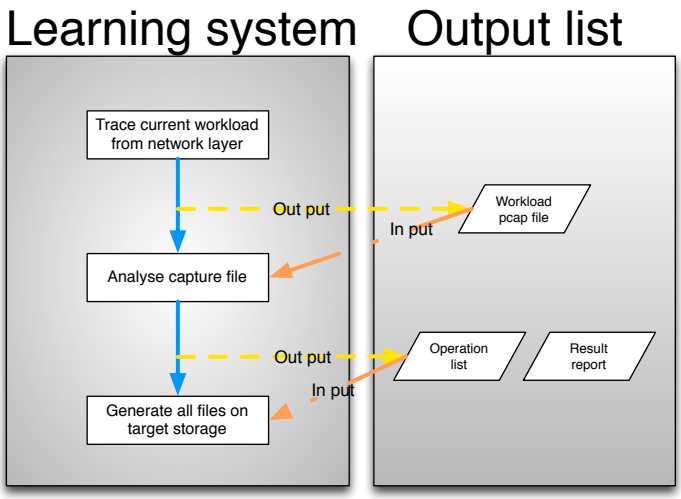

Figure 4.2: Learning system

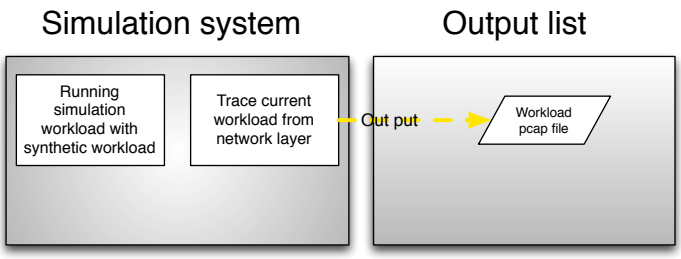

Figure 4.3: simulation system

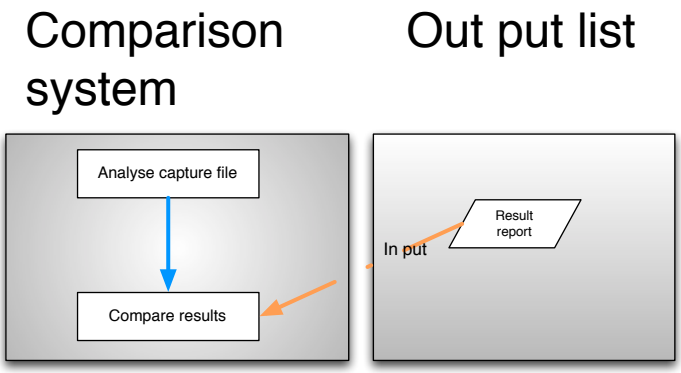

Figure 4.4: Comparison system

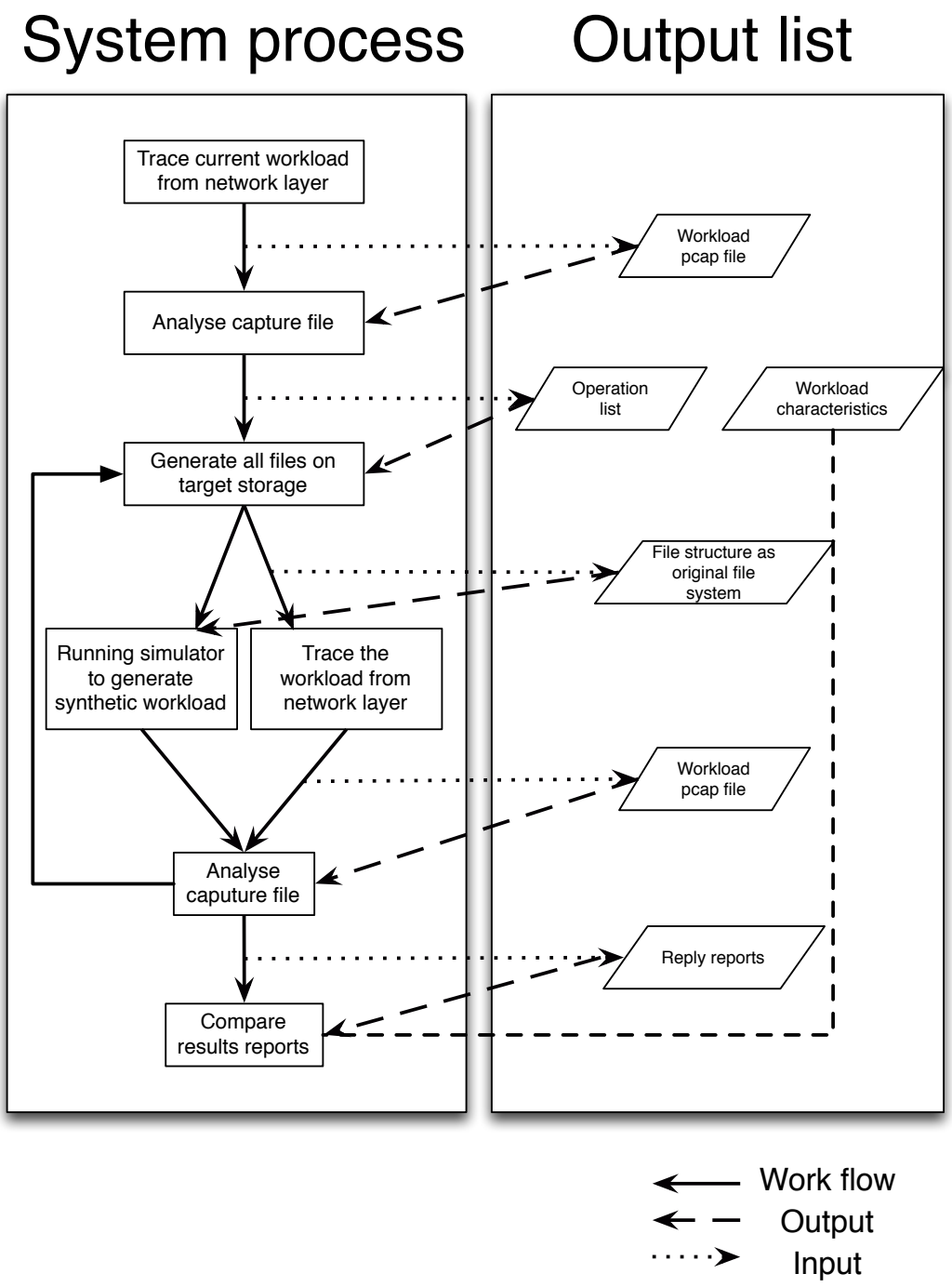

Figure 4.5: Design work flow for benchmark system[8]

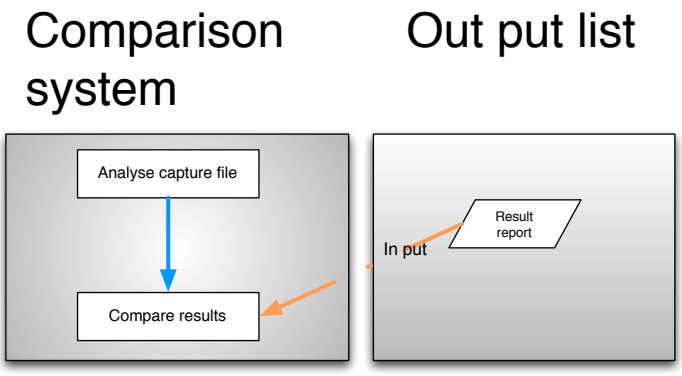

Figure 4.6: Comparison system

ones, to do that it requires us to decode all operations and log them down for the next step. Without it one is not able to prove whether the synthetic workload is identical compare with the original workload. In the first part we need to do the following tasks by developing the learning system.

4.2.1 Capture traffic information: The learning system will collect current product SMB (Server Message Block) traffic information.

In this step, the implement is straight forward. One needs to dump the SAMBA traffic to a pcap file.

4.2.2 Analyze log file which is dumped in previous step

Analyzer is response for two outputs, the first one is a result report, the second one is a operation list.

We designed the analyzer scripts as shown in Figure 4.54.7. After collected the traffic dump file, we open this file and process it twice as showed. First the script filter the pcap file with only read command traffic, which include both requests replies. The script break down each packet information, and save all required elements to a hash variable– %fid. The structure of %fid is a nested structure as shown in Figure 4.6. At the first level we use file name to index all entries.

The structure of %fid is a nested structure as shown in Figure 4.6. At the first level we use file name to index all entries. We only save two types of commands, read and write, it serves as second level index, Then each file can be opened by different processes concurrently, the SAMBA server assigns a unique fid to each open request for a file. After the first three level index, the script push all detail

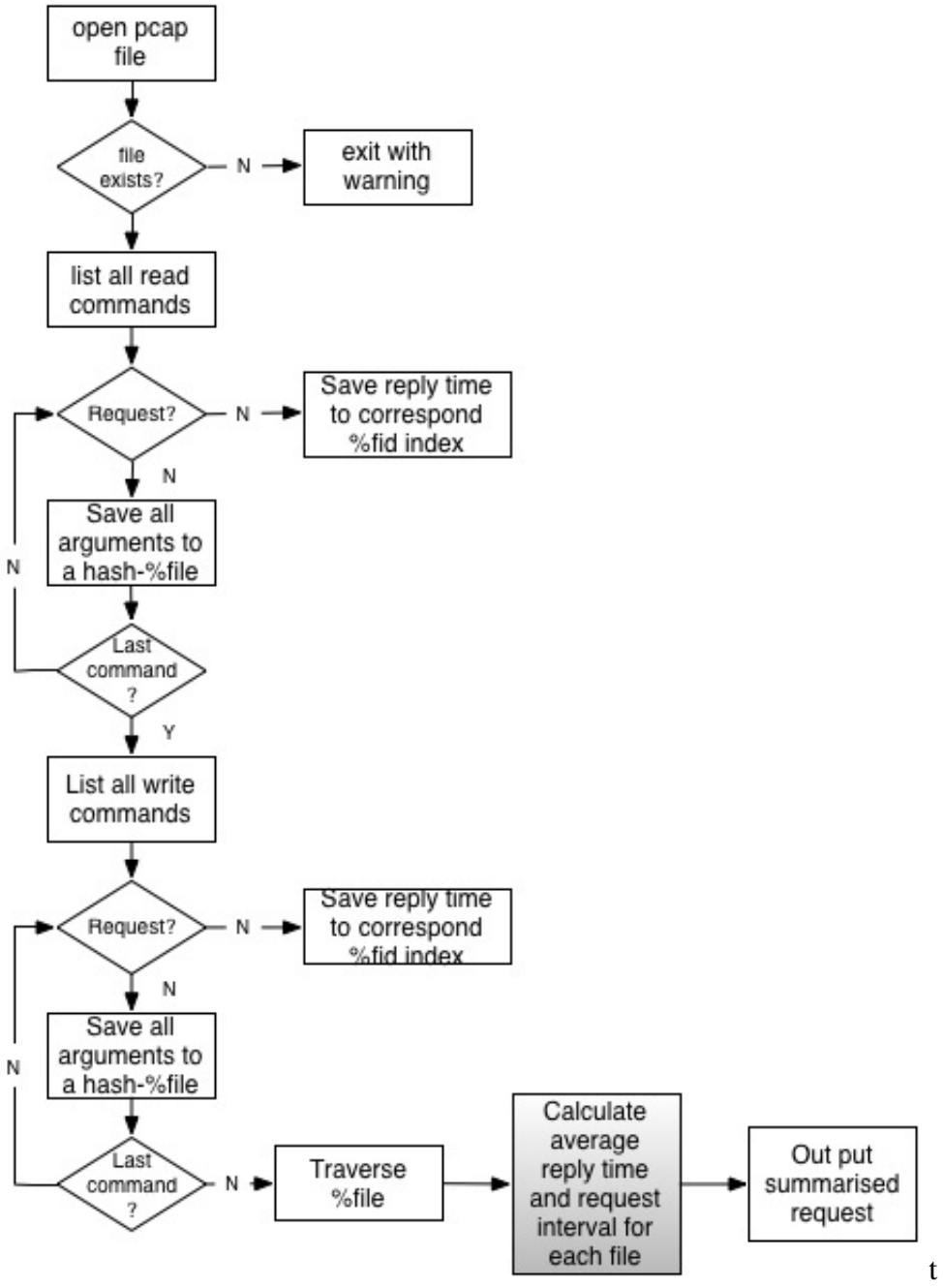

Figure 4.7: Analyzer design

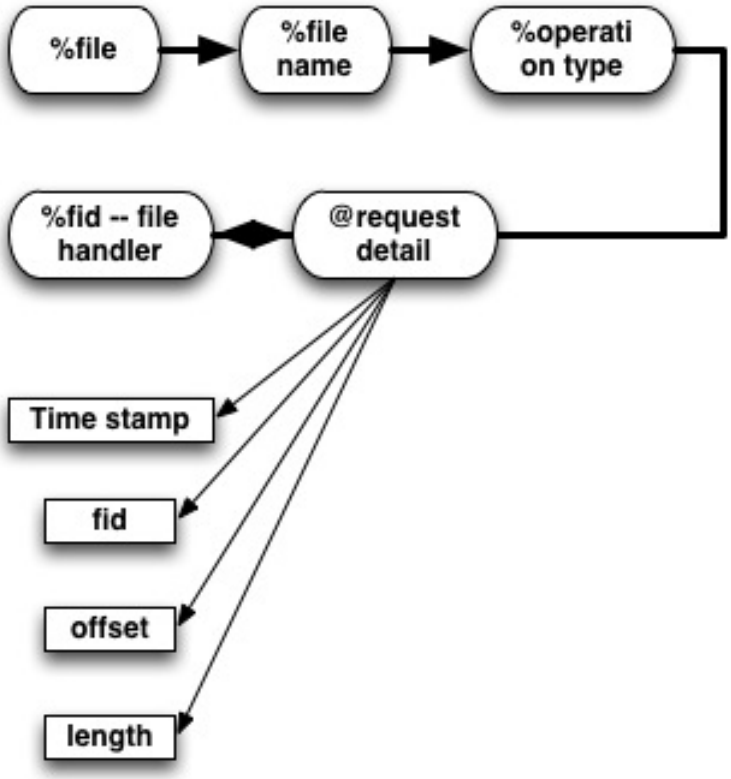

Figure 4.8: Fid structure

arguments of a command in to an array. We employees the array - hash structure to store all the details. We use the time stamp to calculate the time inter-arrival between each requests, and we sum up the correspond sequential requests length and only present the start position and total length as one record in operation list. By integrating all sequential access requests as one record, one can achieve better performance in simulation system, without compromising the workload characteristic. Since we create any random offset access as separate records. According to this structure of fid, one is also able to calculate how many times a file is opened, how many requests for each open fid and total length acquired for this file. The inter-arrival time for each request of one open file handler is also been calculated. Since we only monitor the workload on network layer, so the application layer dependence relationship is transparent for us. Therefor the timestamp is a critical element to approach the similar parallel I/O pattern.

4.2.3 Identify workload

It is critical for us to compare the simulation workload with the original workload. The result can verify the accuracy of our system. In this step, the key properties of a workload are used to identify one. As mention by [12], the following arguments are used to identify a workload:

- Total data length
- Total files
- Average file size
- Read requests
- Write requests
- Mean value of the length of the requests
- Standard deviation of the length of the requests
- inter-arrival time between requests
- offset distance of discrete requests
- Total read request length
- Total write request length

A workload can be defined by these arguments. In the evaluation stage, original workload and synthetic workload are used to test the same storage system twice, then calculate all the arguments for each time. By comparing them, one can tell wether the two workloads are similar or not.

4.2.4 Generate a workload report, which will represent all key characterizations by the previous workload identification

As one of the two outputs of this step, and according to Figure 4.5, the learning system represents a report which lists all key parameters for identifying a workload. All these parameters are used for both evaluating new storage system and validation of current system. The output files are organized by files appeared in network trace.

Name	Example
Request inter-arrival	5.23 e-15
Request length	57890 Bytes
Operation type	W
Reply time	0.0000434 s
Frequency	23

Table 4.1: key parameters

4.2.5 Generating a synthetic workload operation list.

Refer to 4.2.1, an operation list is created based on the request details. Instead of write all request entries to oplist file, those sequential requests are integrated as a single request with their first offset and their total length. In this file, requests are organized as : operation type; time stamp; start offset; length; fid; file name.

4.3 simulation system design

After data collection and analyzation step, we need to replay a synthetic workload. For any storage system, one will experience four kinds of operations, Write/Read operations, and meta operations. We will only replay all read and write operations by this tools, since they will generate the most pressure to storage system. Our simulation system is divided into three parts. The first part is preparation part, it reads oplist file and store all requests parameters to a hash variable- %fid. The thread pool is prepared in the preparation part. Using the thread pool can save our simulator from thread create and reclaim. Since the one could only simulate the parallel workload according to time stamp, we need to shrink the overhead of the simulator. Thread pool creates all worker threads as required, and each time a new thread is called, thread pool assigns a available worker to it. Then after the work finishes its task, the thread pool reclaims it afterward.

The original workload is simulated based file level access pattern. The simulator open the target file at the exactly time as original workload did, then perform all operations for this file handler sequentially. To do that the simulator requires the timestamp for opening a file, then the worker of the simulator is passed all related arguments for replay actual read or write operations.

Each time the worker receives the following arguments:

- File name
- Operation type
- Operation start offset list under an unique file handler

- Operation length list under an unique file handler

A worker open a file according to the file name, and then set the offset one by one according to the offset array. Each time after the worker set up the offset, it start to perform a read or write operation as the arguments specified. All operations for the same file will be performed linearly. The reason for that discussed in Chapter 7 Conclusion. The Figure 4.74.9 shows the simulator will continues to perform the read or write operation according to the operation list, until it traverse all the elements in the operation start offset list, which is passed as an argument for worker. The file handler is closed at the bottom of the worker to avoid exceeding of the limitation of operation system. Because it is a linearly process in a single thread, so we can make sure the access sequential is exactly the same as the original workload.

Simulator begin to generate the synthetic workload as shown. Meanwhile one executes the monitor on network again to capture all information for synthetic workload. The learning tool is used again for two reasons:

- Compare with original workload to calculate accuracy rate.
- Compare with test results of other test target storages to evaluate them.

4.4 Comparison system design

To compare the results between the different storage systems, we need to record their reply time for each response. Tshark is used to collect the reply time from the log file. Tshark can extended its function with a parameter "-z" to display the extra information of each packet. We use the "-z proto,colinfo,smb.time,smb.time" to show the time inter-arrival between a response packet and the related request packet. In Figure 4.10, we listed the work flow of this part. As shown, the reply time is only revealed inside response packet. After we log the reply time into a file named replytime.csv, we can compare this file with others' replytime.csv. The mean value and standard divination can be calculated later.

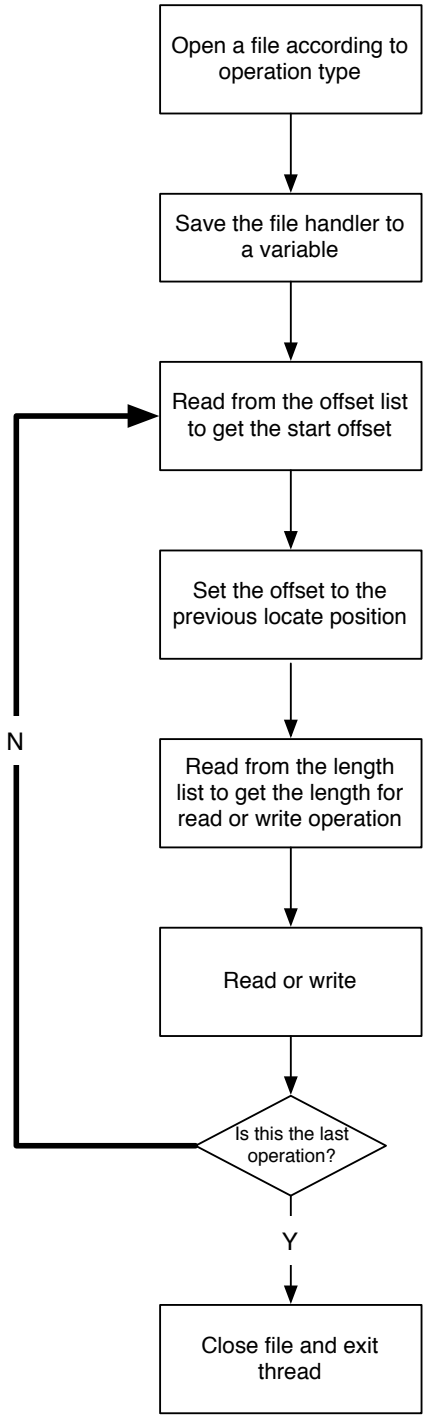

Figure 4.9: Worker process

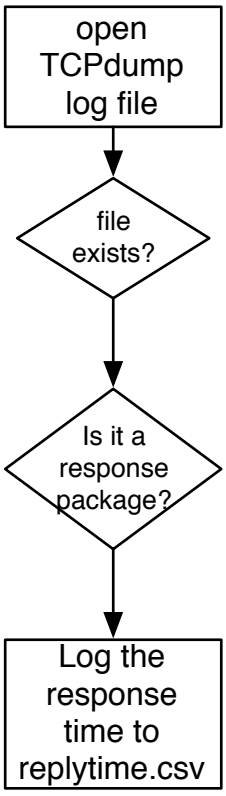

Figure 4.10: Reply time collect

Chapter 5

Results

This chapter describes the experiments and their results. The result and system design is divided into two chapters. The chapter 4 shows all the design process and makes a clear separation between results and the incremental experiment design, the design considerations were put in the separate system design chapter. In this chapter we organized the results in three parts, They are analyze report for original workload, analyze report for synthetic workload, and the compression report for them. The first one and the second one listed the result for evaluation our system in different environments, which is explained in Section 4.1.1. The last part listed the result for benchmarking, one can find the related design in Section 4.1.2.

We use a the4.1 to identify the workload. We have run our tests with different workloads.

5.1 Copy file workload

As we mentioned in previous chapters, the original workload is the workload we dumped in the product environment. The synthetic is the workload we generated by the simulation subsystem.

The first experiment is performed using the following setup:

1. Windows 2008 server as client
2. Ubuntu 12.04 64 server as SAMBA server
3. Direct connection between the client and server
4. The original workload is to copy 13 pictures from the SAMBA server to the local disk of the client

Original workload		Synthetic workload	
Total data length	11620313 byte	Total data length	11620313 byte
Total files	13	Total files	13
Average file size	893870.2	Average file size	893870.2
Read requests	238	Read requests	238
Write requests	179	Write requests	179

Table 5.1: key parameters for original and synthetic workload

We observed both workload by our learning system, and the brief result of the both workloads are shown in Table 5.1. Both workloads requests the same amount of data from server, and the same amount of files.

As mentioned in Chapter 4, until we analyzed the workload properties in the following aspects, one can not tell wether the two workload are identical.

- The offset distances of the workload
- inter-arrival time of each requests
- Request length

We use these properties to identify the workload.

5.1.1 Request offset property

The offset is the position in the file, measured s the number of byes counted from the start of the file, at which the requested bytes should be read from or written to.

The offset property has influence on the storage performance. Since the disk performs better when it access the data sequentially, especially for traditional hark disk. The offset indicates whether the file is accessed sequentially or randomly.

All files are linear spread on disks in our case. We must simulate all offset as it revealed in the original workload. In Figure 5.1 we listed the offset histogram comparison of the original workload and simulation workload. On the upper side of the Figure, we isolated the write operations from the both workloads, it shows the simulation can replay the entire offsets "jump" of the write request as original workload. The histogram Figure shows the offsets location of both the both workloads are identical with each other.

For the read operations, it is also very important to replay the exact offset operations. Since the offset distance can impact the read performance. The best scenario of memory control is that the kernel cache pages with data from the disk in an asynchronous mode, also known as read ahead, so that subsequent read or write system calls can find the requested content in memory. But this request the

data is accessed sequentially. As shown in Figure 3.4, it clearly shows that the operations are not sequential. According to Figure 5.1 our system generate all the initial locations as original workload.

5.1.2 Inter-arrival times

The inter-arrival time of requests is the elapse time between a request and the previous one. It indicate the tensity of the server. From the network dump file, all requests are linearly. The Figure 5.2 shows the simulation workload inter-arrival time is larger then the original workload. They have the same shape. The original workload opened a file to operate before the time we could observed the read or write requests on network. But in the simulation system, we did not open the file until the time we should read or write it in simulation workload. The overhead time was consumed by open operation, as shown in Figure 5.2. To shrink the overhead time, the simulator generated all requests those have the same file identifier descriptor (FID) continuously without any sleep time. There for the original workload has longer inter-arrival time then synthetic workload. In the original workload, the inter-arrival time between requests can be caused by application dependency, network latency or other delay cause by SAMBA client and network.

Figure 5.4 shows the probabilities of the inter-arrival times histogram of both workloads. While the most of requests arrive within less than 0.02 second of the immediately previous request for both workloads. Less then 1% of all requests arrive exceed 0.4 second after the previous request. The left requests arrive less then 0.4 second immediately after the previous requests. The log scale on the y-axis indicates that the magnitude of these inter-arrival time modes tails off sharply. The regularity of this behavior suggests the client system generate a constant access request in this time period.

The inter-arrival time of the original workload is shown in Figure 5.4, if one compare it with the inter-arrival time of the synthetic workload, it shows the synthetic workload inter-arrival time mode tails off even more sharply. The reason for that is the simulator will ignore all the inter-arrival times for all the requests under the same FID.

5.1.3 Data length

Data length is how many bytes client requested from the server. The system replay all the previous requests with the same length and offset positions. As shown in Table 5.2, the original workload is identical to the simulation workload. Both workloads requested the same amount data. The SAMBA server input totally 11620313 bytes and output 11620313 in both cases. The server was requested the

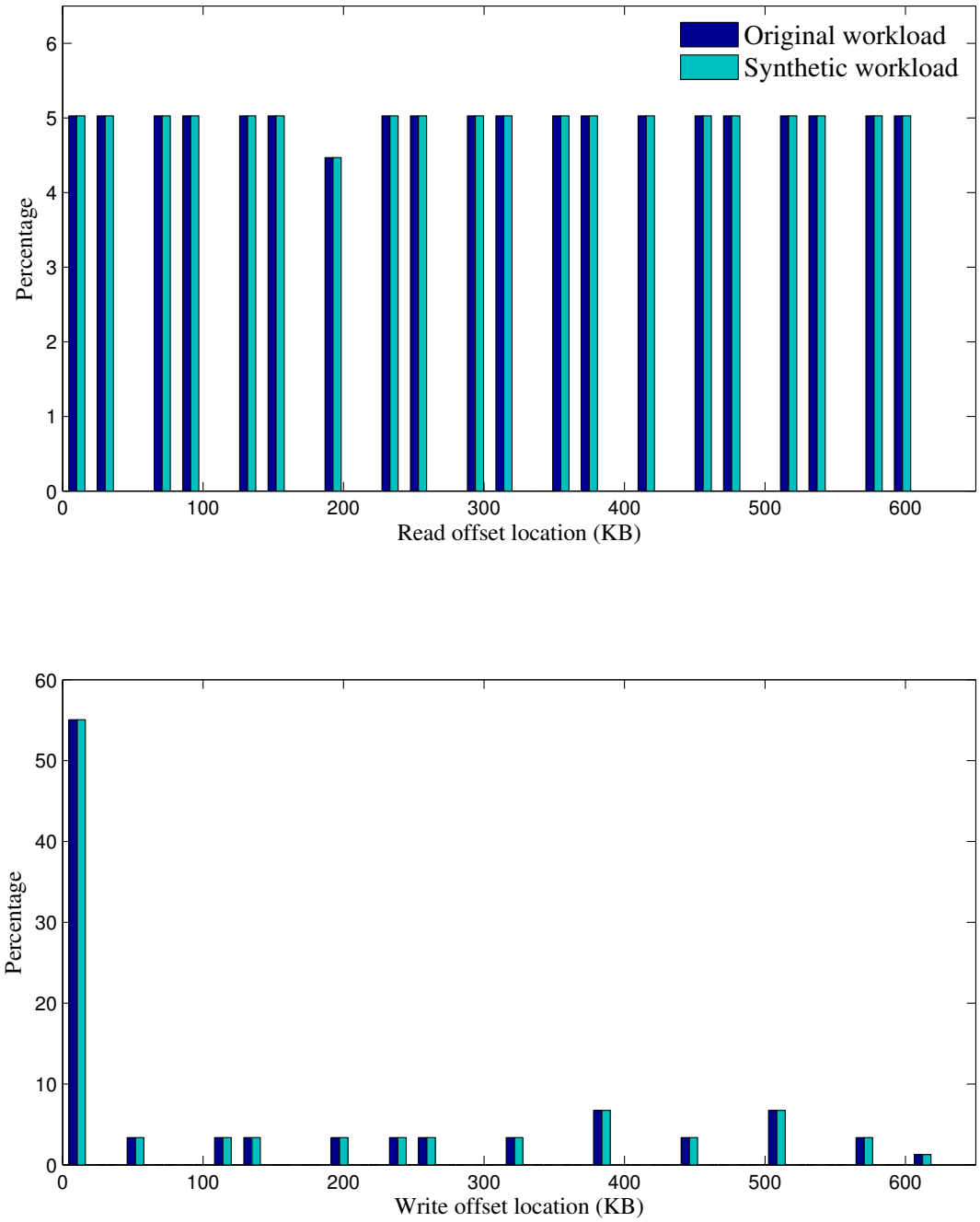

Figure 5.1: Offset initial position histogram

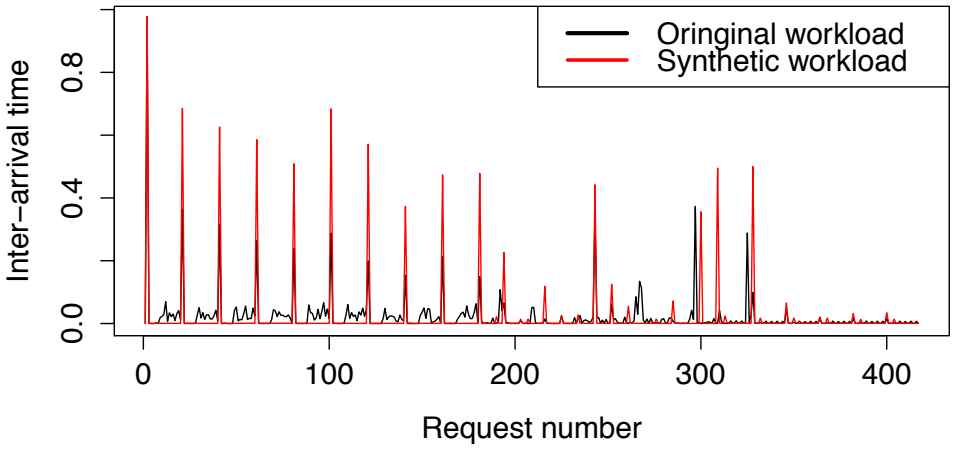

Figure 5.2: Inter-arrival time for copy files

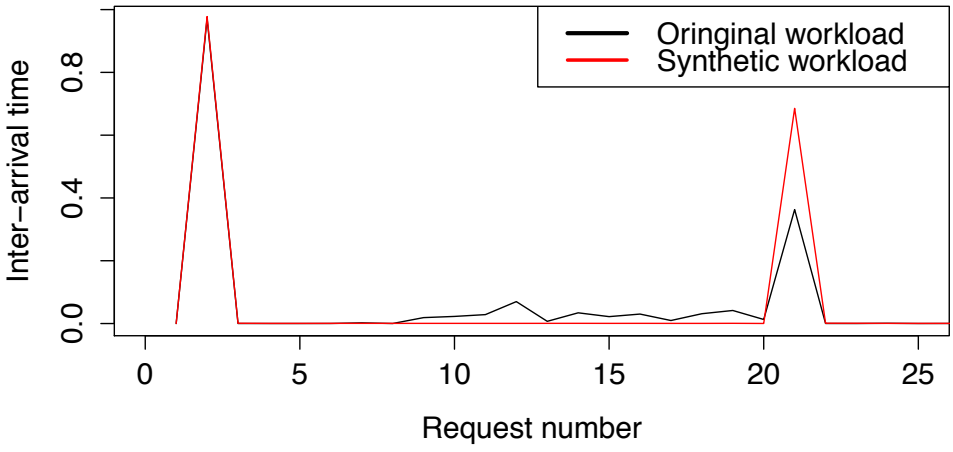

Figure 5.3: Inter-arrival time for copy files zoom in

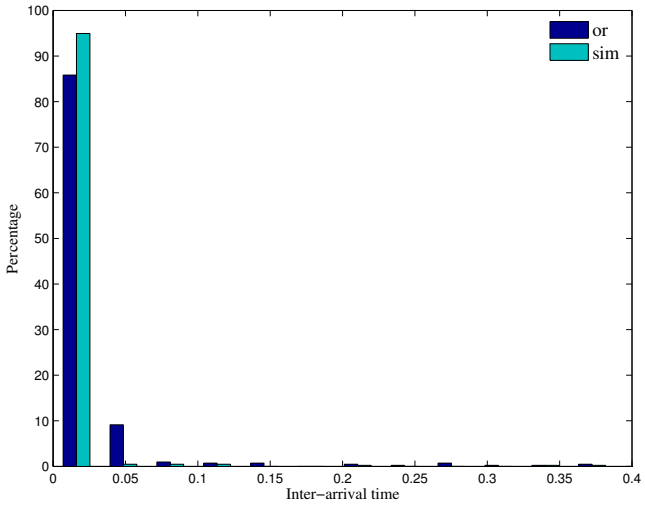

Figure 5.4: Histogram of inter-arrival time

Original workload		Synthetic workload	
Total data length	11620313 byte	Total data length	11620313 byte
Mean value of request	27866.46	Mean value of request	27866.46
Standard deviation	20265.59	Standard deviation	20265.59
Max request length	61440	Max request length	61440
Min request length	4096	Min request length	4096
Total read request length	5915734	Total read request length	5915734
Total write request length	5704579	Total write request length	5704579

Table 5.2: Request length for original and synthetic workload

same amounted data. Considering how similar this request length information, the inter-arrival times and the offset information, one can say the simulation workload is very similar with the original workload.

5.2 Data compression workload, original workload and simulation workload comparison

During the second experiment, we run it on Windows client for a compressing task.

The first experiment is taken under the following setup:

1. Windows 2008 server as client

Original workload		Synthetic workload	
Total data length	798824763byte	Total data length	798824763byte
Total files	5	Total files	5
Average file size	159764953	Average file size	159764953
Read requests	4485	Read requests	4485
Write requests	7876	Write requests	7876

Table 5.3: key parameters for original and synthetic workload

2. Ubuntu 12.04 64 server as SAMBA server

3. Direct connection between the client and server

4. The original workload is to compress 4 files which on the SAMBA server into an archive file on server.

We observed both workload by our learning system, and the brief result of the both workloads are shown in Table 5.3. Both workloads requests the same amount of data from server, and the same amount of files. The task is compress 4 files into an archive file. All the files are on the NAS disk. The final archive file is also on NAS disk. This client should read all files into memory, and then the compression tools should analyze them and write a new file into disk. As shown in 5.3, the client request 780613751 bytes from the SAMBA server. There are 5 files included in this workload, and the average transferred length of files is 156122750 bytes.

5.2.1 Request offset property

In Figure 5.5 we listed the offset histogram comparison of the original workload and simulation workload. On the upper side of the Figure, we isolated the write operations from the both workloads, it shows the simulation can replay the entire offsets "jump" of the write request as original workload. The histogram Figure shows the offsets location of both the both workloads are identical with each other.

5.2.2 inter-arrival times

The Figure 5.6 shows the simulation workload inter-arrival time is larger then the original workload. They have the same shape. The original workload opened a file to operate before the time we could observed the read or write requests on network. But in the simulation system, we did not open the file until the time we should read or write it in simulation workload. The overhead time was consumed by open operation.

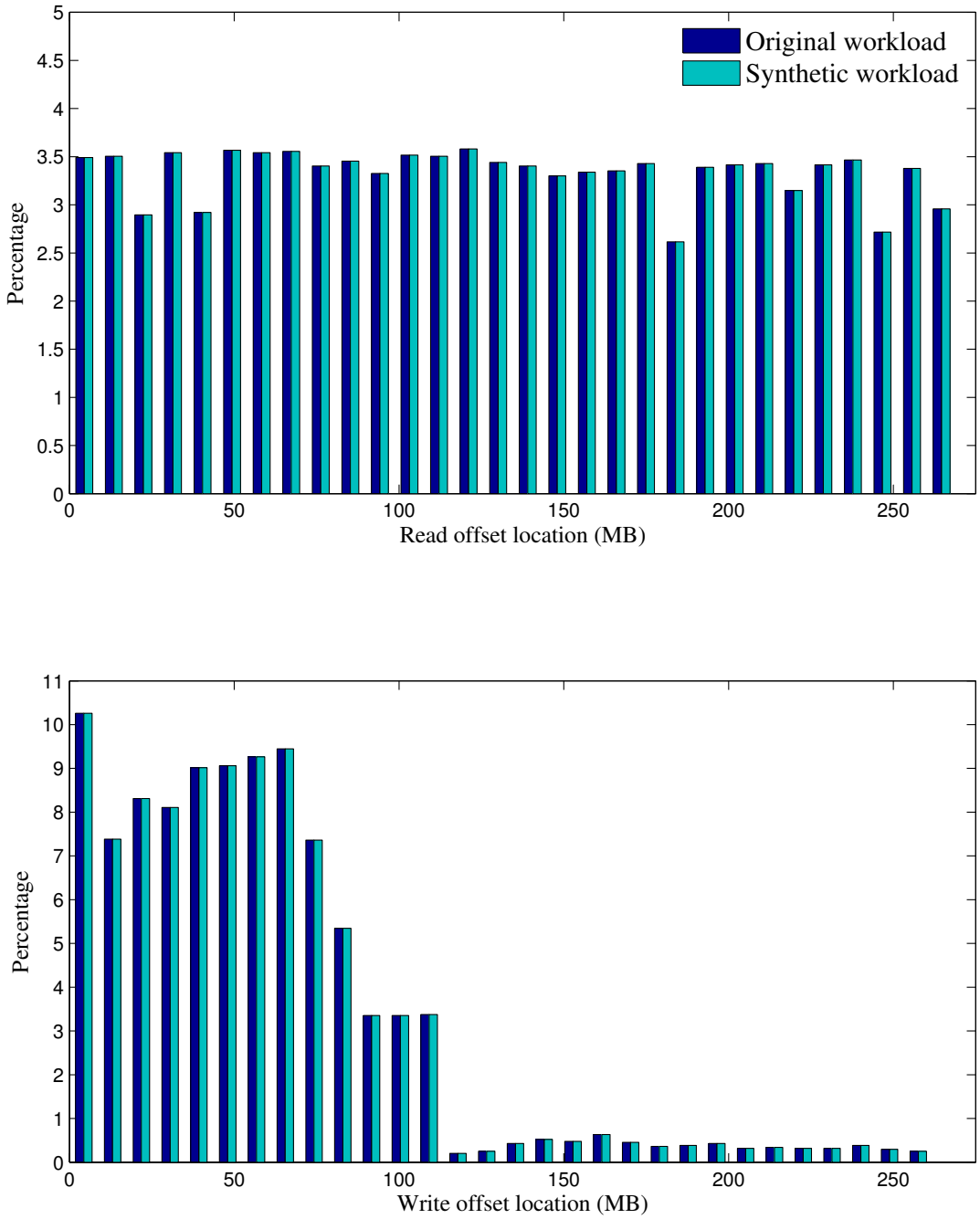

Figure 5.5: Offset initial position histogram

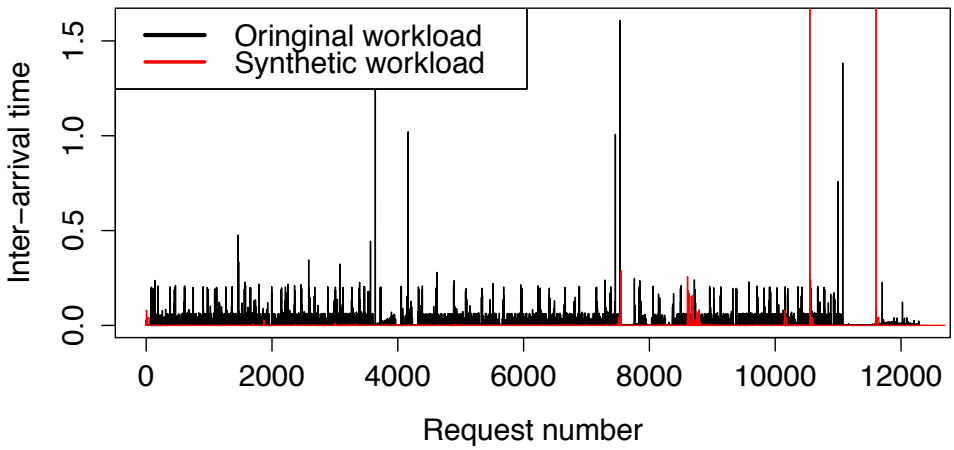

Figure 5.6: Inter-arrival time for copy files, red line stands for synthetic workload,the black line for original workload

As shown in Figure 5.6, the original workload has similar inter-arrival time as the synthetic workload does. The mean value of the inter-arrival time of the original workload is 0.009159702, the mean value of the simulation workload is 0.006249655. The simulator generated all requests those have the same file identifier descriptor (FID) continuously without any sleep time, but in the original workload, the compress tool need to process all the data to generate a archive file, therefor one can see the time of the original workload is longer then the simulator workload.

Figure 5.7 shows the probabilities of the inter-arrival times histogram of both workloads. While the 90% of requests arrive within less than 0.02 second of the immediately previous request for both workloads. The log scale on the y-axis indicates that the magnitude of these inter-arrival time modes tails off sharply as the first experiment. The regularity of this behavior suggests the client system generate a constant access request in this time period. It also shows the synthetic workload inter-arrival time mode tails off even more sharply. The reason for that is the simulator will ignore all the inter-arrival times for all the requests under the same FID.

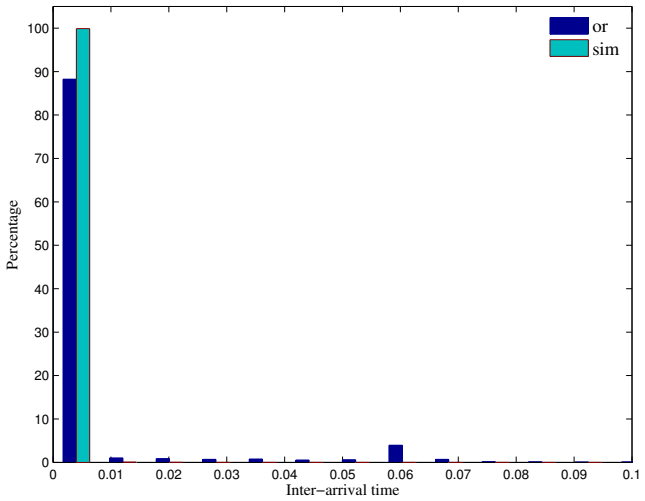

Figure 5.7: Histogram of inter-arrival time

Original workload		Synthetic workload	
Total data length	798824763byte	Total data length	798824763byte
Mean value	5	Mean value	5
Standard deviation	20265.59	Standard deviation	20265.59
Max	61440	Max	61440
Min	4096	Min	4096
Total read request length	5915734	Total read request length	5915734
Total write request length	5704579	Total write request length	5704579

Table 5.4: Request length for original and synthetic workload

5.2.3 Data length

The system synthetic all the previous requests with the same length and offset positions. As shown in Table 5.4, the original workload is identical to the simulation workload. Both workloads requested the same amount data. The SAMBA server input totally 5704579 bytes and outputted 5915734 in both time. The server was tested under the same pressure both times. Combine this request length information with the inter-arrival times and offset information, one can say the simulation workload is similar with the original workload.

	LAPTOP	VM
CPU	2.53GHz i5	2.93GHz E7500
MEMORY	4GB	200MB
HARD DISK TYPE	SSD	5400RPM, shared by VMware
NETWORK	10G	10G

Table 5.5: NAS server hardware information

5.3 System benchmarking

Once after we have proved the synthetic workload is identical with the original workload, we ran the same synthetic workload on two SAMBA servers. They are set up differently. One is a laptop, with SSD and 6GB memory, the other one is a virtual machine, it has a shared 5400RPM disk, and 100MB memory. Under the same network condition, we expected the performance of the laptop would be better then VM. Because the larger memory can lower the chance of having page out and page in operation. The SSD should also response faster then the traditional hard disk.

The same synthetic workload is used to test both servers and the client. The two workloads were generated with the same client.

The reply times are shown in Figure 5.8 , one can tell the reply times from the VM is longer then those from laptop. The laptop reply times are more stable with standard deviation 0.0000411s, the mean value is 6.500e-05 second. The standard deviation of the VM server is 0.0007578231, mean value is 0.000075 second. The laptop performed better as we expected.

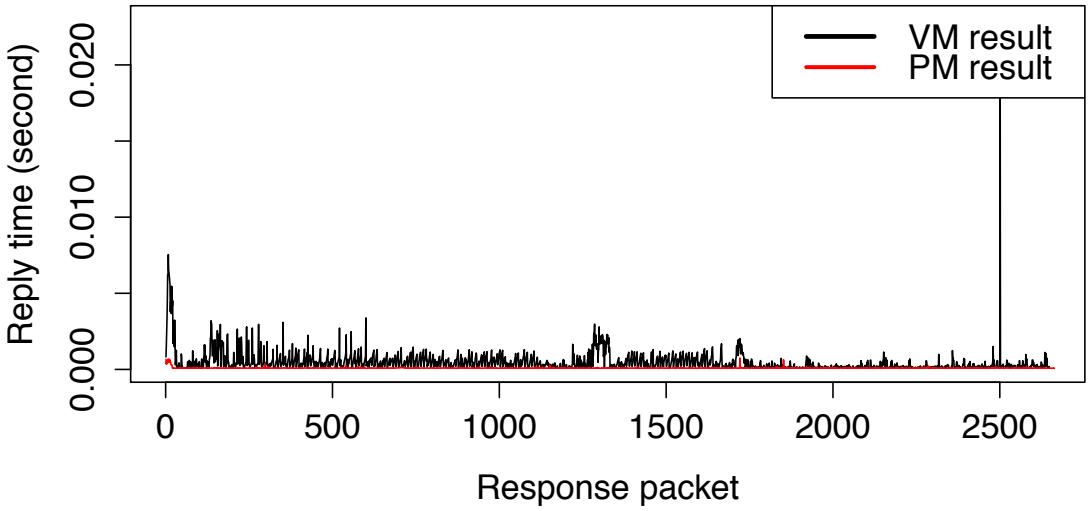

Figure 5.8: Reply time

Chapter 6

Discussion and conclusion

6.1 Review of the approach

Storage administrators try to find a way to benchmark a potential storage system efficiently and accurately, but no such system exists today. The entire project is set up on a trace and replay methodology. Instead of using general workloads to test users' highly customized environment, we first learn from the product environment, and then we analyze the workload we dumped from the network. The last step is to generate a synthetic workload to test the potential storage systems. By doing so, one can benchmark the storage system efficiently and accurately.

In the first step, the goal is to dump all network requests to the storage. To replay them accurately one need to dump as complete as possible. To satisfy this requirement, one need to dump the network traffic by using a mirror port, since this function consumes the least resources on switch. To keep the traffic trace completely, one should make sure the sniffer machine, which is shown in Figure 4.2, is capable to process the traffic fast enough.

SMB/CIFS protocol has seventy five different command category totally, but in our project only the request and reply of read and write commands are recorded. Since the most resources of the storage system are consumed by read and write requests. We record all the necessary variables for workload identifications, the request type of a request, the time of a request, the length and offset of a request, and the file handler and the file name of a request. This information is used to tailor the simulation workload.

In the replay part, we first pre-load all the necessary variables in to memory. This pre-load operation can accelerate the program efficiency for the replay part. All the request are grouped according to the file handler. The simulator replays all the request groups according to the time stamp of every first request of each group. The time gap between the group is filled with sleep time. All the following

requests in the group are replayed one by one. During the simulation, all packets are dumped to sniffer again, as shown in Figure 3.2. After the simulation is implemented on all storages, the traces are used to generate the reply time reports. The performance capacity of each storage system is evaluated by the reply time report.

6.2 Tools used

At the first and the last stage, TCPdump is used for capturing the workload we need to analyze. We chose TCPdump to capture the network traffic because TCPdump is known as a stable and fast trace recording tool. It is embedded with Ubuntu server. We take advantage from a filter function of TCPdump. Since the SMB/CIFS protocol is only running on TCP 445 port normally, one can minimize the size of the log file by only dumping the traffic on port 445. This lowers the risk of losing packet from the network traffic.

Tshark is used for analyzing the network capture file. Since we do not replay the workload at the same time we record it in our system, the analyzer is not required to be efficient. Tshark can extract all the variable we need from every TCP packet. Meanwhile it can calculate the time elapse between the request packets and its response packets. This function is very useful in our project. Since the reply time is what we used to evaluate the performance of all the storage systems.

SmbClient is employed to generate the SMB requests. This perl module is a useful tool set. It helps us to establish the SMB/CIFS connection between client and server, it also encapsulates all the contents according to protocol requirement. But since this tool set is designed to test the functionality of the SMB/CIFS environment, it initializes a new connection every time we open a file, and disconnected it once the file is closed. This costs extra time on the server. The extra time is revealed as inter-arrival time difference in Figure5.2. Along with this weakness, it still satisfies our requirement to show that the trace and replay methodology can be used in real product environment.

6.3 Replay decision

When replaying the commands used for opening a file for reading or writing, we replay them ahead of the read or write requests. This makes it easier to replay the workload, but the trade off of that is that it also lowers the accuracy of the inter-arrival time. Since in the original workload, the file is opened ahead of the read or write request. That is the reason we observed overhead in Figure 5.2 and Figure 5.3. We also replayed all the following request in the same group as a linear

flow without any waiting time, even when there is waiting time in original workload. We are implement it in this way because we assume the it generates larger pressure for the storage systems, but it is still similar with the original workload. The waiting time in the original workload can be caused by multi requests from different clients, different threads or user behavior. User behavior caused the time gap inside a group in three ways, first is the user's behavior, for example reading a large document, the client only loads a part of the file at once. The second is that application dependency, the application will not require further data until other application has done with the current data. The third reason is that client is overloaded, it will cause the delay between requests. Our choice is to assume that the time inter-arrival inside each group is caused by different threads. Since this can increase the pressure of each group requests for server, and the inter-arrival time between each group is still maintain the same.

Another decision we made to replay the workload is that we only simulate all the requests from a single client. In the original workload, the requests generally come from several clients and not just a single one. But the time and resource are limited, so we put this improvement into future work.

During the analyzing part, the default assumption is that the user network is stable and match the bandwidth requirement. Otherwise the test result may be compromised, the reason of performance enhancement is hard to define.

6.4 System benchmarking

In our project, the reply time is the only variable we used to score a target storage system. Normally one also need to consider more variables to get a conclusion for performance evaluation of a storage system, for example IOPS. Normally response time and throughput is matters for storage performance. We have already use the similar workload for every test, the throughput is depends on how fast the server can response. So that is the reason one can only focus on the response time in our project. Our system has a fixed and highly customized synthetic workload to generate the pressure for storage systems.

6.5 Problem statement discussion

The goal of this thesis was to get an accurate benchmark result for every unique user environment. To accomplish that we have performed the following tasks:

- Learned the original workload without compromising users' storage performance

- Generated a synthetic workload based on the first step
- Shown that the synthetic workload is identical to the original workload
- Run the simulation against different storage systems and verify that the results are useful when evaluating the capacity of the storage system

We consider the third task the most important one. In the learning sub-system of our project, we have fulfilled the first requirement by extracting all the necessary data from the raw network dump log. We also managed to generate a simulation workload based on the first step results. To do the last step, we have employed a method designed by Kavalanekar[12] to identify a workload. The key variables for a workload are request length, request offset, request type, and inter-arrival time between requests. By analyzing all the variables, we were able to define a synthetic workload which was similar compared with the original workload, but the simulation is not identical with the original workload. We have discussed the reasons for that in Section 6.3.

6.6 File system aging

As mentioned in Subsection 3.2.3.1 System setup and configuration, our system generated all necessary files on the target system. We used the linux command dd to implement that operation. But since all of our target storage systems are newly installed or conFigured, all data will be perfectly striped on the storage. This generally is different from the product storage system. Fragmentation is unavoidable for any storage system. After they have been used for a while. The discrete file layout lower the performance of all kinds of storages. Therefor the performance of a simulation which runs on a newly set up storage can take advantage of the continuous distributed file layout.

But it is impossible to observed a fragmentation status of a file system from the network trace. Because of this, we are unable to age our target storage systems according to the file layout of the product storage system.

However, we can still use our current system to evaluate all target storage systems. Because all the test targets are newly installed or configured, there all have the identical continuous layout. Under such situation we can compare the performance results of them.

6.7 Future work and suggested improvements

For the learning system, one can collect more types of operations, such as remove, create, get attribute and operations for directories. By inserting all of those vari-

ables into our simulation system, one can get a more accurate synthetic workload.

But some of those operations may fail due to lack of necessary contents. For example, when the simulator attempt to execute a create file request on the server, but the file already exists. Then the operation will fail. To avoid such situations, we need manually insert some correction operations into the replay workload. For the previous example, a remove operation is required before the create operation.

For the simulation system, we can improve it in the following aspects, and these aspects will be discussed in the next subsections.

- Customize the SMBclient
- Implementing a distributed-simulator
- Add capacity tuning option

6.7.1 Customize the SMBclient

The current perl module we used in our simulation system is designed for a functionality test. Instead of maintaining a long connection between server and client, it establishes and releases the connection every time a file is opened or closed. One can observe many connect/disconnect requests from the simulation workload, they are nonexistent in the original workload. The other shortcoming of this module is that it is not thread safe. The module is calling the libsmbclient library, and this library is not thread safe. One can experience some unexpected error during the simulation process. Since the file handler is a shared data structure of all the threads, and it can be accessed by more then one thread in some case.

To solve this problem one can use multi-process instead of a multi-thread implementation. With the multi-thread implementation, one can create all threads ahead. Therefor the system requires no extra CPU time and other resources to handle the creation or termination work for all threads, which is not avoidable in multi-process implementation. Our simulator is a time sensitive system, the inter-arrival times between requests are usually smaller than 0.001 second or of the same order of magnitude. Therefor multi-thread implementation has advantages for that it consumes no extra time when all the SMB/CIFS requests are generated.

6.7.2 Implementing a distributed-simulator

In the current system, we simulate all workload by a single client. But in the original workload, the requests are generally generated by several clients. So the distribute-simulator can offer more accurate impact to the server. One need to consider about the synchronized method between clients. As we mention in Subsection 6.7.1, the simulation system is a time sensitive system, hence the time

synchronize must be implemented. It also requires a central control system to direct all the systems.

6.7.3 Add capacity tuning option

Most of the benchmarking tools or systems can provide a tuning function, users can increase or decrease the workload pressure for the server. By doing this, one can estimate how much pressure can be taken for a target storage, and where the bottleneck of the current system is. This option works in the similar way as a micro-benchmarking tool. These are useful if you are measuring a very small change, to better understand the results of a workload, to isolate the effects of specific parts of the system, or to show worst-case behavior[1]. The challenge of this part is that one should able to show the adjusted workload is still similar or identical with the original one.

6.8 Conclusion

Along with the data explosion era, storage capacity requirement is growing rapidly. It leads the cost of storage systems has a significant percentage of the totoal IT costs. A benchmarking system can secure one's investment on storage can satisfy their requirements both on capacity and performance. An easy implement and accurate benchmarking system is naturally required.

Our system significantly lower the complexity in the implementation stage compare with data maceration, and it could provide the almost same accuracy. As we mentioned in the problem statement, the system can analyze all the SAMBA traffic, and generate a synthetic workload based on previous analyzation. It can also be evaluated for a new environment, to verify is this system can generate a similar workload as it should be.

But as the Section 5.2, the system can not work perfectly under some scenario. If the client needs to process each packet by some application, the inter-arrival time of the synthetic workload will be smaller then the original workload. Which also means the simulator will generate larger pressure for the NAS storage compare with the original workload. To get a more accurate synthetic workload, we need to develop a more efficient simulator, so we can replay the original workload inter-arrival time for each request, instead of only focusing on request groups. Currently, we group multiple requests by their FIDs. But current version of the system still can be a useful tool even under the worse scenario. In that case, the benchmarking result of our system shows the system performance under larger pressure then current usage.

This project opens several potential path for future research and business approach methods.

Chapter 7

Appendixes

7.1 Learning system

7.1.1 Analyze.pl

```
1  #!/usr/bin/perl
2
3  use strict "vars";
4  use Getopt::Std;
5  use Statistics::Basic qw(:all);
6
7  my %opts;
8  my %file;
9  my $totaltime;
10 my $globalcnt;
11 my $f;
12 my $filecount=0;
13 my @filesize;
14
15 my $big=0;
16 system "rm -f oplist";
17 getopts('f:h',\%opts);
18 if ($opts{f}){
19         $f=$opts{f};
20 }else{
21         die ("Please supply pcap file name for
              analyzing!\n");
22 }
```

```
23 system "rm␣-f␣oplist";
24
25 open (reply,">replytime.csv");
26
27 &read;
28 &write;
29
30 &summary;
31
32 sub read{
33         open (w,">>oplist");#this is operation list,
              we will use this to generate replay code
              based on this
34         open (r,"tshark␣-R␣'smb.cmd==0x2e'␣-z␣proto,
              colinfo,smb.file,smb.file␣-z␣proto,colinfo,
              smb.time,smb.time␣-r␣$f|");
35
36         my $i==0;
37
38         while (my $line = <r>){
39                         my $off;
40                 my $length;
41                 my $fid;
42                 my $time;
43                 my $fn;
44                 if ($line=~ /.*Request.*/){
45                         $line =~ /\d+\s+(\d+\.\d+).*
                              FID:\s(.*),\s(\d+)\sbytes.*
                              offset\s(\d+).*"(.*)"/;
46                         $time=$1;
47                         $fid=$2;
48                         $length=$3;if ($3>$big){$big=
                              $3;}
49                         $off=$4;
50                         $fn=$5;
51                         $fn=~ s/[^\d|\w|\\|\.|\-|\%|\$
                              ]//g;
52                         $file{$fn}{r}{count}++;
53                         push (@{$file{$fn}{r}{$fid}},{
                              time=>$time,length=>$length
                              ,fid=>$fid,off=>$off,fn=>
```

```
                                $fn});
54
55                      }
56                      if ($line=~ /.*Response.*/){
57                              $line =~ /smb.time == (\S+)\s+
                                   smb.file == "(.*)"/;
58                              $fn=$2;
59                              $fn=~ s/[^\d|\w|\\|\.|\ -|\%|\$
                                   ]//g;
60                              push (@{$file{$fn}{r}{
                                   replytime}},$1);
61                              $line =~ /smb.time == (\S+)/;
62                              print reply "$1\n";
63
64                      }
65              }
66
67
68
69
70  }
71
72
73  sub write{
74              open (w,">>oplist");#this is operation list,
                   we will use this to generate replay code
                   based on this
75              open (r,"tshark␣-R␣'smb.cmd==0x2f'␣-z␣proto,
                   colinfo,smb.file,smb.file␣-z␣proto,colinfo,
                   smb.time,smb.time␣-r␣$f␣|");
76
77              my $i==0;
78
79              while (my $line = <r>){
80
81                      my $off;
82                      my $length;
83                      my $fid;
84                      my $time;
85                      my $fn;
86                      if ($line=~ /.*Request.*/){
```

```
87                              $line =~ /\d+\s+(\d+\.\d+).*
                                   FID:\s(.*),\s(\d+)\sbytes.*
                                   offset\s(\d+).*"(.*)"/;
88                              $time=$1;
89                              $fid=$2;
90                              $length=$3;
91                              $off=$4;
92                              $fn=$5;
93                              $fn=~ s/[^\d|\w|\\|\.|\-|\%|\$
                                   ]//g;
94                              if (exists $file{$fn}{r}{$fid
                                   }){
95                                      $fid="w$fid";
96                                      ##incase file is
                                           opened as read, and
                                            also accessed as
                                           append or creat
97                              }
98                              $file{$fn}{w}{count}++;
99                              push (@{$file{$fn}{w}{$fid}},{
                                   time=>$time,length=>$length
                                   ,fid=>$fid,off=>$off,fn=>
                                   $fn});
100
101
102                     }
103                     if ($line=~ /.*Response.*/){
104                             $line =~ /smb.time == (\S+)\s+
                                   smb.file == "(.*)"/;
105                             $fn=$2;
106                             $fn=~ s/[^\d|\w|\\|\.|\-|\%|\$
                                   ]//g;
107                             push (@{$file{$fn}{w}{
                                   replytime}},$1);
108
109                     }
110             }
111
112 }
113 sub avgsub{
114             my @input=@_;
```

```
115         my $avg;
116         foreach (@input){
117                 $avg=+$_;
118         }
119         my $num=scalar (@input);
120         if ($num!=0){
121                 $avg=$avg/$num;
122                 return $avg;
123         }else{
124                 return 0;
125         }
126
127 }
128
129 sub summary{
130         foreach my $k1 (keys %file){
131                 my $tmpfilelength;
132                   if ($k1){
133                         foreach my $o (keys %{$file{
                              $k1}}){
134                         print "opert: $o\n";
135                         my $avg=&avgsub(@{$file{$k1}{
                              $o}{replytime}});
136                         if ($avg==0){
137                                 print "check this file
                                      : $k1\n";
138                         }
139                         my $std=stddev(@{$file{$k1}{$o
                              }{replytime}});
140                         my $totoallength;
141
142         foreach my $k2 (keys %{$file{$k1}{$o}}){
143                 if ($k2 eq "replytime"){next;}
144                 my @a1=@{$file{$k1}{$o}{$k2}};
145                 foreach(sort {$$a{time} <=> $$b{time}
                      } @a1){
146                         my $end= $$_{off} +$$_{length}
                              ;
147                         $totoallength+=$$_{length};
148                         $tmpfilelength+=$$_{length};
```

```
149                             print w "$o\t$$_{time}\t$$_{
                                    off}\t$$_{length}\t$$_{fid
                                    }\t$$_{fn}\n";
150                             }
151         }
152                                                         }
153                 }
154                 $filecount++;
155                 push (@filesize,$tmpfilelength);
156         }
157
158
159 }
160 my $averagefilelength=mean(@filesize);
161
162 print "Total␣file␣accessed␣is␣$filecount,␣average␣file
    ␣length␣is␣$averagefilelength\n";
163
164 system "cp␣oplist␣$f.txt";
```

7.1.2 filegenerator.pl

```
1  #!/usr/bin/perl
2
3  use strict "vars";
4  use Getopt::Std;
5
6  my $path;
7  my %fn;
8  my %opts;
9  getopts('f:h',\%opts);
10 if ($opts{f}){
11         if (-d $opts{f}){
12                 $path=$opts{f};
13         } else{
14                 warn "folder␣not␣exists,␣use␣/srv/
                       samba/share2␣instead.\n";
15                 $path="/srv/samba/share2";
16
17         }
```

```
18 }
19
20
21
22
23 open (r,"oplist");
24
25 while (my $line =<r>){
26
27         my @l= split(/\t/,$line);
28         #r       3017.755829      13373440
               13393920          0x1d0e   \\GruppeC7-Prosjekt
               .rvt
29         my $name= $l[5];
30         $name =~ s/\\\\/\//g;
31         chomp ($name);
32         my $len= $l[3];
33         my $start=$l[2];
34         my $end=$len+$start;
35         if ($fn{$name}){
36                 if ($fn{$name}<=$end){
37                         $fn{$name}=$end;
38                 }
39
40         }else{
41                 $fn{$name}=$end;
42         }
43
44 }
45
46 foreach my $key (keys %fn){
47         $key =~ /(.*)\/.*/;
48         my $fp=$path.$1;
49         system "sudo␣mkdir␣-p␣$fp";print "$fp\n";
50         system "sudo␣dd␣if=/dev/urandom␣of=$path$key␣
               bs=$fn{$key}␣count=1";
51         print "sudo␣dd␣if=/dev/urandom␣of=$path$key␣bs
               =$fn{$key}␣count=1";
52         system "sudo␣chmod␣777␣-R␣$path";
53
54 }
```

7.1.3 Sort.pl

```
1  #!/usr/bin/perl
2  use strict "vars";
3  use Getopt::Std;
4
5  my %opts;
6  my $sortonly=0;
7  my $f;
8  my $minus=0;
9  my $insert=0;
10
11 getopts('sf:im',\%opts);
12 if ($opts{m}){
13         $minus=1;
14 }
15 if ($opts{s}){
16         $sortonly=1;
17 }
18 if ($opts{i}){
19         $insert=1;
20 }
21 if ($opts{f}){
22         $f=$opts{f};
23 }
24
25
26 my %time;
27 my %fid;
28
29 my $min=11111111111111111111111111111111111111;
30
31 if (defined $opts{f}){
32         #print "file name is $f\n";
33         open (r,$f) or die "Please run analyze.pl
               first, and check oplist file or supply file
               name with -f\n";
34 }else{
```

```
35          open (r,"oplist") or die "Please␣run␣analyze.
               pl␣first ,␣and␣check␣oplist␣file.\n";
36  }
37  while (my $line = <r>){
38
39          my @a= split (/\t/, $line);
40          $time{$a[1]}{line}=$line;
41          #r       17997.074110    0       4096    0x47ba
                \\www\\netbeans\\oblig1\\._oblig_1_oppg_1
               .php
42          $time{$a[1]}{o}=$a[0];
43          $time{$a[1]}{s}=$a[2];
44          $time{$a[1]}{l}=$a[3];
45          $time{$a[1]}{f}=$a[4];
46          $time{$a[1]}{fn}=$a[5];
47
48          if (not exists $fid{$a[4]}{first} or $fid{$a
               [4]}{first}>= $a[1]){
49                  $fid{$a[4]}{first}=$a[1];
50                  $fid{$a[4]}{type}=$a[0];
51          }
52
53          chomp ($time{$a[1]}{fn});
54          $time{$a[1]}{fn} =~ s/\\\\/\//g;
55
56          if ($min>=$a[1]){$min=$a[1];}
57  }
58
59  close (r);
60
61  if (defined $opts{f}){
62          open (w,">$f");
63  }else{
64          open (w,">oplist");
65  }
66
67  if ($sortonly){
68          print "running␣sort␣only\n";
69          &onlysort;
70  }
71  elsif ($minus){
```

```
72          print "running substract only\n";
73          &substract;
74  }elsif($insert){
75          &insert;
76          &onlysort;
77  }else{
78          &insert;
79          &substract;
80  }
81
82
83
84  sub substract{
85          my $last=0;
86          foreach my $key (sort {$a<=>$b} keys %time){
87
88                  if ($last==0){
89                          print w "$time{$key}{o}\t0\
                                t$time{$key}{s}\t$time{$key
                                }{l}\t$time{$key}{f}\t$time
                                {$key}{fn}\n";
90                  }else{
91
92                          my $t= $key-$last;
93                          print w "$time{$key}{o}\t$t\
                                t$time{$key}{s}\t$time{$key
                                }{l}\t$time{$key}{f}\t$time
                                {$key}{fn}\n";
94                  }
95                  $last=$key;
96
97          }
98
99          #print $min."\n";
100
101 }
102
103
104 sub onlysort{
105         foreach my $key (sort {$a<=>$b} keys %time){
106
```

```
107                     print w "$time{$key}{line}";
108
109             }
110 }
111
112 sub insert{
113             foreach my $key (keys %fid){
114                     my $opentime= $fid{$key}{first}-2;#
                            open 2 seconds before first request
115                     $time{$opentime}{o}="o$fid{$key}{type}
                            ";
116 #print "open time $opentime\n";
117                     $time{$opentime}{fn}=$time{$fid{$key}{
                            first}}{fn};
118                     $time{$opentime}{f}=$time{$fid{$key}{
                            first}}{f};
119                     $time{$opentime}{line}="$time{
                            $opentime}{o}\t$opentime\t$time{
                            $opentime}{f}\t$time{$opentime}{fn
                            }\n";
120             }
121
122 }
```

7.2 Simulation system

```
1  #!/usr/bin/perl
2
3  use strict;
4  use Getopt::Std;
5  use Filesys::SmbClient;
6  use Time::HiRes qw(usleep nanosleep);
7  use threads;
8  use Thread::Pool;
9
10
11 my $ip;
12 my $user;
13 my $pass;
```

```
14 my $domain;
15 my %opts;
16 my %fid;
17 my $path;
18
19 getopts('i:u:p:d:', \%opts);
20
21 if ($opts{i}){
22         $ip=$opts{i};
23 }else{
24         die ("Please␣supply␣specify␣ip␣address␣for␣smb
             ␣server\n");
25 }
26
27 if ($opts{p}){
28         $path=$opts{p};
29 }else{
30         die ("Please␣supply␣specify␣share␣name␣for␣smb
             ␣server\n");
31 }
32
33 system "sudo␣chmod␣777␣oplist";
34 system "./sort.pl␣-f␣oplist␣-s";
35 ########################part 1 grab arguments from
     oplist, we will only simulate access pattern for
     file level instead of package level.
36 my %fid;###structured as fid->$time fid->$op fid->
     @startoffset fid->@length
37 my %maintime;###to store file level even time,
     structured as maintime->time->fn maintime->time->
     $op maintime->time->fid
38
39
40
41 open (r,"oplist");
42
43 while (my $line =<r>){
44         my @al= split (/\t+/,$line);
45         my $packageop=$al[0];
46
47
```

```
48          if ($packageop eq "w" || $packageop eq "r"){
49                  my $packagetime=$al[1];
50                  my $packagestart=$al[2];
51                  my $packagelength=$al[3];
52                  my $packagefid=$al[4];
53                  my $packagefn=$al[5];
54                  $packagefn=~ s/\\\\/\//g;
55
56                  if (defined $fid{$packagefid}){
57                          if ($fid{$packagefid}{time}>=
                               $packagetime){
58                                  $fid{$packagefid}{time
                                       }=$packagetime;##
                                       only log file first
                                        access time, other
                                        access will be
                                       sequencial
59                          }
60
61                  }else{
62                          $fid{$packagefid}{time}=
                               $packagetime;
63                          $fid{$packagefid}{op}=
                               $packageop;
64                          $fid{$packagefid}{fn}=
                               $packagefn;
65                  }
66                  push (@{$fid{$packagefid}{start}},
                       $packagestart);
67                  push (@{$fid{$packagefid}{length}},
                       $packagelength);
68          }
69  }
70
71
72
73          #insert first access time to maintime
74  foreach my $key (keys %fid){
75
76          $maintime{$fid{$key}{time}}{fid}=$key;
```

```
77          $maintime{$fid{$key}{time}}{op}=$fid{$key}{op
                };
78
79  }
80
81          ####test area
82  foreach my $time (sort {$a<=>$b} keys %maintime){
83  #       print "maintime time: $time op:$maintime{$time
        }{op} fid:$maintime{$time}{fid} fn:$maintime{$time
        }{fn}\n";
84  }
85  ########################part 2 preparation part
        ####################################
86  open (r2,"string"); ##prepare for write operation, it
        will read this random file and write its content to
         target file location
87
88
89          #prepare thread pool for later usage
90  my $pool = Thread::Pool->new(
91  {
92  workers => 1,
93  do => \&do,
94  }
95  );
96
97
98  system "ulimit␣-s␣16384"; #this operation will
        increase the stack value to 16M for thread in linux
        . By doing this one could read more at once from
        smb server without "Segmentation fault (core dumped
        )"
99
100 my $string10m :shared= &genstring(10485760); #prepared
         the fix content for write operation
101
102 my @sleep=&sleeptime;
103
104
105 ########################part 3 simulation part
        ####################################
```

```
106 foreach my $time (sort {$a<=>$b} keys %maintime){
107         if ($maintime{$time}{op} eq "r"){
108                 $pool->job(\$fid{$maintime{$time}{fid
                        }}{fn},"r",\@{$fid{$maintime{$time
                        }{fid}}{start}},\@{$fid{$maintime{
                        $time}{fid}}{length}});
109         }
110         if ($maintime{$time}{op} eq "w"){
111                 $pool->job(\$fid{$maintime{$time}{fid
                        }}{fn},"w",\@{$fid{$maintime{$time
                        }{fid}}{start}},\@{$fid{$maintime{
                        $time}{fid}}{length}});
112         }
113
114         my $sleep= shift @sleep;
115         $sleep=$sleep*1000000000;
116         nanosleep ($sleep);
117 }
118
119 $pool->autoshutdown( 1 ); # shutdown when object is
      destroyed
120 $pool->shutdown;            # wait until all jobs done
121 $pool->abort;
122
123 ########################part 4 subrutine part
124
125 sub do{
126 my $smb = new Filesys::SmbClient(username  => "",
127                                  password  => "",
128                                  workgroup => "
                                       WORKGROUP");
129         my ($f,$op,$s,$l)=@_;
130         my $fn=$$f;
131         chomp ($fn);
132         my $fid=0;
133         while (1){
134
135                 $fid=$smb->open("smb://10.0.0.2/
                        share2$fn") or print "Can't␣read␣
                        file:", $!, "$fn\n";
136                 nanosleep (20);
```

```
137                     if ($fid)
138                     {
139                             print "open successed\n";
140                             last;
141                     }
142             }
143             my @start=@$s;
144             my @length=@$l;
145             foreach my $key (keys @start){ ##run each
                    access request for one open file with one
                    thread
146                     $smb->seek($fid,$start[$key]);
147             #       while ($length[$key]>104857){###cut
                    the request to max length for stack size,
                    this will increase the throughput
148             #               if ($op eq "r"){
149             #                       my $a=$smb->read ($fid
                    ,104857);
150             #               }
151             #               if ($op eq "w"){
152             #                       $smb->write ($fid,
                    $string10m);
153             #               }
154             #               $length[$key]-=104857;
155             ##      }
156
157                     if ($op eq "r"){
158                             my $a=$smb->read ($fid,$length
                                [$key]);
159                     }
160                     if ($op eq "w"){
161                             my $string =&genstring($length
                                [$key]);
162                             $smb->write ($fid,$string);
163                     }
164
165
166             }
167             $smb->close($fid);
168             print "job $fn end\n";
169     }
```

```
170
171 sub genstring{
172         my $a=shift @_;
173         my $b;
174         read (r2,$b,$a);
175         seek (r2,0,0);
176         return $b;
177 }
178
179 sub sleeptime{
180         my @tmp;
181         my $last=0;
182         foreach my $time (sort {$a<=>$b} keys %
              maintime){
183
184                 if ($last==0){
185                         $last=$time;
186                         next;
187                 }else{
188
189                         my $t= $time-$last;
190                         push (@tmp,$t);
191                 }
192                 $last=$time;
193
194         }
195
196         return @tmp;
197
198 }
```

7.3 Oplist example

```
r       0       0       32768   0x4c17  /eclipse/
    artifacts.xml
r       1.79999999971869e-05    32768   65536   0x4c17
    /eclipse/artifacts.xml
r       0.00020200000000159     65536   98304   0x4c17
    /eclipse/artifacts.xml
```

```
r       5.99999999906231e-06    98304   131072  0x4c17
    /eclipse/artifacts.xml
r       1.00000000102796e-06    131072  147612  0x4c17
    /eclipse/artifacts.xml
r       0.0558109999999985      0       17920   0x4c18
    /eclipse/eclipsec.exe
r       0.00697900000000118     0       16536   0x4c19
    /eclipse/epl-v10.html
r       0.0918069999999993      0       32768   0x4c1a
    /eclipse/eclipse.exe
r       1.80000000007396e-05    32768   65536   0x4c1a
    /eclipse/eclipse.exe
r       2.99999999953116e-06    65536   98304   0x4c1a
    /eclipse/eclipse.exe
r       2.00000000205591e-06    98304   131072  0x4c1a
    /eclipse/eclipse.exe
r       2.99999999953116e-06    131072  163840  0x4c1a
    /eclipse/eclipse.exe
r       1.9999999985032e-06     163840  196608  0x4c1a
    /eclipse/eclipse.exe
r       0.011375000000001       196608  229376  0x4c1a
    /eclipse/eclipse.exe
r       0.0510629999999992      229376  262144  0x4c1a
    /eclipse/eclipse.exe
r       0.207333999999999       262144  294912  0x4c1a
    /eclipse/eclipse.exe
r       2.3999999999802e-05     294912  312320  0x4c1a
    /eclipse/eclipse.exe
r       0.00414300000000267     0       514     0x4c1b
    /eclipse/eclipse.ini
r       0.0352969999999999      0       32768   0x4c1c
    /eclipse/.eclipseproduct
r       2.09999999967181e-05    32768   65536   0x4c1c
    /eclipse/.eclipseproduct
r       3.00000000308387e-06    65536   98304   0x4c1c
    /eclipse/.eclipseproduct
r       2.99999999953116e-06    98304   131072  0x4c1c
    /eclipse/.eclipseproduct
r       1.9999999985032e-06     131072  163840  0x4c1c
    /eclipse/.eclipseproduct
```

```
r       2.00000000205591e-06    163840  196608  0x4c1c
    /eclipse/.eclipseproduct
r       3.99999999970064e-06    196608  229376  0x4c1c
    /eclipse/.eclipseproduct
r       0.007149000000000185    229376  262144  0x4c1c
    /eclipse/.eclipseproduct
r       0.0334219999999981      262144  294912  0x4c1c
    /eclipse/.eclipseproduct
r       0.0005040000000002947   294912  327680  0x4c1c
    /eclipse/.eclipseproduct
r       0.04300899999999979     327680  360448  0x4c1c
    /eclipse/.eclipseproduct
r       0.0001500000000001427   360448  393216  0x4c1c
    /eclipse/.eclipseproduct
r       0.0007969999999998604   393216  425984  0x4c1c
    /eclipse/.eclipseproduct
r       0.0003050000000000888   425984  458752  0x4c1c
    /eclipse/.eclipseproduct
r       0.00125699999999895     458752  491520  0x4c1c
    /eclipse/.eclipseproduct
r       6.00000000261502e-06    491520  524288  0x4c1c
    /eclipse/.eclipseproduct
r       0.0009199999999997145   524288  557056  0x4c1c
    /eclipse/.eclipseproduct
r       0.0006850000000000713   557056  589824  0x4c1c
    /eclipse/.eclipseproduct
r       0.0008559999999998858   589824  622592  0x4c1c
    /eclipse/.eclipseproduct
r       0.0008270000000001021   622592  655360  0x4c1c
    /eclipse/.eclipseproduct
r       0.00106699999999904     655360  688128  0x4c1c
    /eclipse/.eclipseproduct
r       3.00000000308387e-06    688128  720896  0x4c1c
    /eclipse/.eclipseproduct
r       0.00884799999999686     720896  753664  0x4c1c
    /eclipse/.eclipseproduct
r       0.0004170000000002332   753664  786432  0x4c1c
    /eclipse/.eclipseproduct
r       0.0006069999999998692   786432  819200  0x4c1c
    /eclipse/.eclipseproduct
```

```
r       0.000295999999998742    819200  851968  0x4c1c
    /eclipse/.eclipseproduct
r       0.0017610000000019      851968  884736  0x4c1c
    /eclipse/.eclipseproduct
r       0.000682000000001182    884736  917504  0x4c1c
    /eclipse/.eclipseproduct
r       2.99999999953116e-06    917504  950272  0x4c1c
    /eclipse/.eclipseproduct
r       0.00079200000000057     950272  983040  0x4c1c
    /eclipse/.eclipseproduct
r       0.000367999999998148    983040  1015808 0x4c1c
    /eclipse/.eclipseproduct
r       0.000517000000002099    1015808 1048576 0x4c1c
    /eclipse/.eclipseproduct
r       0.00052599999999714     1048576 1081344 0x4c1c
    /eclipse/.eclipseproduct
r       5.00000000158707e-06    1081344 1114112 0x4c1c
    /eclipse/.eclipseproduct
r       0.000807999999999254    1114112 1146880 0x4c1c
    /eclipse/.eclipseproduct
r       0.000662999999999414    1146880 1179648 0x4c1c
    /eclipse/.eclipseproduct
r       0.00022700000000242     1179648 1212416 0x4c1c
    /eclipse/.eclipseproduct
r       0.000276999999996974    1212416 1245184 0x4c1c
    /eclipse/.eclipseproduct
r       0.00582100000000096     1245184 1277952 0x4c1c
    /eclipse/.eclipseproduct
r       0.00105999999999895     1277952 1310720 0x4c1c
    /eclipse/.eclipseproduct
r       5.00000000158707e-06    1310720 1343488 0x4c1c
    /eclipse/.eclipseproduct
r       0.000523000000001161    1343488 1376256 0x4c1c
    /eclipse/.eclipseproduct
r       0.000385999999998887    1376256 1409024 0x4c1c
    /eclipse/.eclipseproduct
r       0.000564000000000675    1409024 1441792 0x4c1c
    /eclipse/.eclipseproduct
r       0.000384000000000384    1441792 1474560 0x4c1c
    /eclipse/.eclipseproduct
```

```
r       0.00054399999999788     1474560 1507328 0x4c1c
    /eclipse/.eclipseproduct
r       0.000265999999999877    1507328 1540096 0x4c1c
    /eclipse/.eclipseproduct
r       0.000934000000000879    1540096 1572864 0x4c1c
    /eclipse/.eclipseproduct
r       0.000282999999999589    1572864 1605632 0x4c1c
    /eclipse/.eclipseproduct
r       0.000209999999999155    1605632 1638400 0x4c1c
    /eclipse/.eclipseproduct
r       0.000615000000003363    1638400 1671168 0x4c1c
    /eclipse/.eclipseproduct
r       0.00139999999999674     1671168 1703936 0x4c1c
    /eclipse/.eclipseproduct
r       0.000262000000002871    1703936 1736704 0x4c1c
    /eclipse/.eclipseproduct
r       0.000237999999999516    1736704 1769472 0x4c1c
    /eclipse/.eclipseproduct
r       0.000610999999999251    1769472 1802240 0x4c1c
    /eclipse/.eclipseproduct
r       0.000852999999999327    1802240 1835008 0x4c1c
    /eclipse/.eclipseproduct
r       2.99999999953116e-06    1835008 1867776 0x4c1c
    /eclipse/.eclipseproduct
r       0.000348000000002457    1867776 1900544 0x4c1c
    /eclipse/.eclipseproduct
r       0.000753999999997035    1900544 1933312 0x4c1c
    /eclipse/.eclipseproduct
r       0.000505000000000422    1933312 1966080 0x4c1c
    /eclipse/.eclipseproduct
r       5.00000000158707e-06    1966080 1998848 0x4c1c
    /eclipse/.eclipseproduct
r       0.00075999999999965     1998848 2031616 0x4c1c
    /eclipse/.eclipseproduct
r       0.000236999999998488    2031616 2064384 0x4c1c
    /eclipse/.eclipseproduct
r       0.000720000000001164    2064384 2097152 0x4c1c
    /eclipse/.eclipseproduct
r       0.000375999999999266    2097152 2129920 0x4c1c
    /eclipse/.eclipseproduct
```

```
r       0.0003250000000000131    2129920 2162688 0x4c1c
    /eclipse/.eclipseproduct
r       0.0007110000000002571    2162688 2195456 0x4c1c
    /eclipse/.eclipseproduct
r       0.0005529999999996473    2195456 2228224 0x4c1c
    /eclipse/.eclipseproduct
r       7.000000000009027e-06    2228224 2260992 0x4c1c
    /eclipse/.eclipseproduct
r       0.0002880000000001176    2260992 2293760 0x4c1c
    /eclipse/.eclipseproduct
r       0.0008529999999999327    2293760 2326528 0x4c1c
    /eclipse/.eclipseproduct
r       0.0007520000000002084    2326528 2359296 0x4c1c
    /eclipse/.eclipseproduct
r       0.0004899999999999213    2359296 2392064 0x4c1c
    /eclipse/.eclipseproduct
r       4.999999998034 35e-06    2392064 2424832 0x4c1c
    /eclipse/.eclipseproduct
r       0.0004670000000000439    2424832 2457600 0x4c1c
    /eclipse/.eclipseproduct
r       5.000000001587 07e-06    2457600 2490368 0x4c1c
    /eclipse/.eclipseproduct
r       0.0005419999999999376    2490368 2523136 0x4c1c
    /eclipse/.eclipseproduct
r       7.000000000090 27e-06    2523136 2555904 0x4c1c
    /eclipse/.eclipseproduct
r       0.0007669999999999974    2555904 2588672 0x4c1c
    /eclipse/.eclipseproduct
r       0.0004109999999999717    2588672 2621440 0x4c1c
    /eclipse/.eclipseproduct
r       0.0005170000000002099    2621440 2654208 0x4c1c
    /eclipse/.eclipseproduct
r       0.0003229999999998075    2654208 2686976 0x4c1c
    /eclipse/.eclipseproduct
r       0.0007079999999999487    2686976 2719744 0x4c1c
    /eclipse/.eclipseproduct
r       0.0006360000000000081    2719744 2752512 0x4c1c
    /eclipse/.eclipseproduct
r       6.000000002615 02e-06    2752512 2785280 0x4c1c
    /eclipse/.eclipseproduct
```

```
r       0.0002519999999999697    2785280  2818048  0x4c1c
    /eclipse/.eclipseproduct
r       0.00123299999999915      2818048  2850816  0x4c1c
    /eclipse/.eclipseproduct
r       0.0005559999999999557    2850816  2883584  0x4c1c
    /eclipse/.eclipseproduct
r       5.99999999906231e-06     2883584  2916352  0x4c1c
    /eclipse/.eclipseproduct
r       0.0002120000000001211    2916352  2949120  0x4c1c
    /eclipse/.eclipseproduct
r       0.00179299999999927      2949120  2981888  0x4c1c
    /eclipse/.eclipseproduct
r       8.40000000010832e-05     2981888  3014656  0x4c1c
    /eclipse/.eclipseproduct
r       0.0002980000000000798    3014656  3047424  0x4c1c
    /eclipse/.eclipseproduct
r       0.0001639999999998055    3047424  3080192  0x4c1c
    /eclipse/.eclipseproduct
r       0.00169800000000109      3080192  3112960  0x4c1c
    /eclipse/.eclipseproduct
r       0.0005379999999998817    3112960  3145728  0x4c1c
    /eclipse/.eclipseproduct
r       0.0004090000000001214    3145728  3178496  0x4c1c
    /eclipse/.eclipseproduct
r       0.0002099999999999155    3178496  3211264  0x4c1c
    /eclipse/.eclipseproduct
r       0.00102500000000205      3211264  3244032  0x4c1c
    /eclipse/.eclipseproduct
r       0.0001509999999998902    3244032  3276800  0x4c1c
    /eclipse/.eclipseproduct
r       0.0002700000000000437    3276800  3309568  0x4c1c
    /eclipse/.eclipseproduct
r       0.0002390000000000544    3309568  3342336  0x4c1c
    /eclipse/.eclipseproduct
r       0.00137999999999749      3342336  3375104  0x4c1c
    /eclipse/.eclipseproduct
r       0.0007179999999999108    3375104  3407872  0x4c1c
    /eclipse/.eclipseproduct
r       0.0004420000000003162    3407872  3440640  0x4c1c
    /eclipse/.eclipseproduct
```

```
r       0.000276999999996974    3440640 3473408 0x4c1c
    /eclipse/.eclipseproduct
r       0.000220000000002329    3473408 3506176 0x4c1c
    /eclipse/.eclipseproduct
r       0.000467999999997915    3506176 3538944 0x4c1c
    /eclipse/.eclipseproduct
r       0.000239000000000544    3538944 3571712 0x4c1c
    /eclipse/.eclipseproduct
r       0.00105100000000036     3571712 3604480 0x4c1c
    /eclipse/.eclipseproduct
r       0.000830000000000553    3604480 3637248 0x4c1c
    /eclipse/.eclipseproduct
r       2.99999999953116e-06    3637248 3670016 0x4c1c
    /eclipse/.eclipseproduct
r       0.000358000000002079    3670016 3702784 0x4c1c
    /eclipse/.eclipseproduct
r       0.000167999999998614    3702784 3735552 0x4c1c
    /eclipse/.eclipseproduct
r       0.00201900000000066     3735552 3768320 0x4c1c
    /eclipse/.eclipseproduct
r       0.000394999999997481    3768320 3801088 0x4c1c
    /eclipse/.eclipseproduct
r       0.000499000000001359    3801088 3833856 0x4c1c
    /eclipse/.eclipseproduct
r       5.99999999906231e-06    3833856 3866624 0x4c1c
    /eclipse/.eclipseproduct
r       0.00130700000000061     3866624 3899392 0x4c1c
    /eclipse/.eclipseproduct
r       0.000299000000001826    3899392 3932160 0x4c1c
    /eclipse/.eclipseproduct
r       0.000153999999998433    3932160 3964928 0x4c1c
    /eclipse/.eclipseproduct
r       0.000272999999999968    3964928 3997696 0x4c1c
    /eclipse/.eclipseproduct
r       0.00153099999999995     3997696 4030464 0x4c1c
    /eclipse/.eclipseproduct
r       0.000398000000000565    4030464 4063232 0x4c1c
    /eclipse/.eclipseproduct
r       4.00000000055911e-06    4063232 4096000 0x4c1c
    /eclipse/.eclipseproduct
```

```
r       1.9999999985032e-06     4096000 4128768 0x4c1c
    /eclipse/.eclipseproduct
r       0.0009090000000001455   4128768 4161536 0x4c1c
    /eclipse/.eclipseproduct
r       0.0004640000000000908   4161536 4194304 0x4c1c
    /eclipse/.eclipseproduct
r       0.0004179999999999807   4194304 4227072 0x4c1c
    /eclipse/.eclipseproduct
r       4.99999999803435e-06    4227072 4259840 0x4c1c
    /eclipse/.eclipseproduct
r       0.003095999999999932    4259840 4292608 0x4c1c
    /eclipse/.eclipseproduct
r       0.00023400000000251     4292608 4325376 0x4c1c
    /eclipse/.eclipseproduct
r       0.0004409999999998581   4325376 4358144 0x4c1c
    /eclipse/.eclipseproduct
r       0.000455999999999979    4358144 4390912 0x4c1c
    /eclipse/.eclipseproduct
r       0.0004610000000001377   4390912 4423680 0x4c1c
    /eclipse/.eclipseproduct
r       0.0004789999999998564   4423680 4456448 0x4c1c
    /eclipse/.eclipseproduct
r       0.0004270000000001954   4456448 4489216 0x4c1c
    /eclipse/.eclipseproduct
r       2.99999999953116e-06    4489216 4521984 0x4c1c
    /eclipse/.eclipseproduct
r       0.001153999999999966    4521984 4554752 0x4c1c
    /eclipse/.eclipseproduct
r       2.99999999953116e-06    4554752 4587520 0x4c1c
    /eclipse/.eclipseproduct
r       0.0004080000000000186   4587520 4620288 0x4c1c
    /eclipse/.eclipseproduct
r       5.99999999906231e-06    4620288 4653056 0x4c1c
    /eclipse/.eclipseproduct
r       0.001121999999999874    4653056 4685824 0x4c1c
    /eclipse/.eclipseproduct
r       0.0006790000000001651   4685824 4718592 0x4c1c
    /eclipse/.eclipseproduct
r       0.0002239999999999336   4718592 4751360 0x4c1c
    /eclipse/.eclipseproduct
```

```
r        0.000154000000001986    4751360  4784128  0x4c1c
    /eclipse/.eclipseproduct
r        0.0027659999999976      4784128  4816896  0x4c1c
    /eclipse/.eclipseproduct
r        0.0004100000000002242   4816896  4849664  0x4c1c
    /eclipse/.eclipseproduct
r        4.99999999803435e-06    4849664  4882432  0x4c1c
    /eclipse/.eclipseproduct
r        0.000206999999999624    4882432  4915200  0x4c1c
    /eclipse/.eclipseproduct
r        0.0016820000000024      4915200  4947968  0x4c1c
    /eclipse/.eclipseproduct
r        2.99999999953116e-06    4947968  4980736  0x4c1c
    /eclipse/.eclipseproduct
r        1.00000000102796e-06    4980736  5013504  0x4c1c
    /eclipse/.eclipseproduct
r        0.0002420000000000075   5013504  5046272  0x4c1c
    /eclipse/.eclipseproduct
r        0.00131499999999818     5046272  5079040  0x4c1c
    /eclipse/.eclipseproduct
r        0.0002120000000001211   5079040  5111808  0x4c1c
    /eclipse/.eclipseproduct
r        0.0004809999999997067   5111808  5144576  0x4c1c
    /eclipse/.eclipseproduct
r        4.00000000055911e-06    5144576  5177344  0x4c1c
    /eclipse/.eclipseproduct
r        0.0007280000000002282   5177344  5210112  0x4c1c
    /eclipse/.eclipseproduct
r        0.0003630000000000113   5210112  5242880  0x4c1c
    /eclipse/.eclipseproduct
r        0.0005009999999999863   5242880  5275648  0x4c1c
    /eclipse/.eclipseproduct
r        5.90000000002533e-05    5275648  5308416  0x4c1c
    /eclipse/.eclipseproduct
r        0.0030179999999973      5308416  5341184  0x4c1c
    /eclipse/.eclipseproduct
r        0.0004489999999997      5341184  5373952  0x4c1c
    /eclipse/.eclipseproduct
r        6.00000000261502e-06    5373952  5406720  0x4c1c
    /eclipse/.eclipseproduct
```

```
r        0.000379999999999825     5406720  5439488  0x4c1c
     /eclipse/.eclipseproduct
r        0.000824999999998965     5439488  5472256  0x4c1c
     /eclipse/.eclipseproduct
r        0.000617999999999341     5472256  5505024  0x4c1c
     /eclipse/.eclipseproduct
r        0.000372000000002259     5505024  5537792  0x4c1c
     /eclipse/.eclipseproduct
r        0.000593999999999539     5537792  5570560  0x4c1c
     /eclipse/.eclipseproduct
r        0.000498000000000332     5570560  5603328  0x4c1c
     /eclipse/.eclipseproduct
r        0.000258999999999787     5603328  5636096  0x4c1c
     /eclipse/.eclipseproduct
r        0.000308000000000419     5636096  5668864  0x4c1c
     /eclipse/.eclipseproduct
r        0.000418999999997283     5668864  5701632  0x4c1c
     /eclipse/.eclipseproduct
r        0.000845000000001761     5701632  5734400  0x4c1c
     /eclipse/.eclipseproduct
r        0.000378999999998797     5734400  5767168  0x4c1c
     /eclipse/.eclipseproduct
r        0.000226000000001392     5767168  5799936  0x4c1c
     /eclipse/.eclipseproduct
r        0.000272999999999968     5799936  5832704  0x4c1c
     /eclipse/.eclipseproduct
r        0.000554999999998529     5832704  5865472  0x4c1c
     /eclipse/.eclipseproduct
r        0.000680000000002678     5865472  5898240  0x4c1c
     /eclipse/.eclipseproduct
r        0.000490999999996689     5898240  5931008  0x4c1c
     /eclipse/.eclipseproduct
```

Bibliography

[1] AVISHAY TRAEGER, EREZ ZADOKA, NIKOLAI JOUKOV, CHARLES P. WRIGHT, “Nine Year Study of File System and Storage Benchmarking”. ACM. 2008.

[2] Christina Delimitrou, Sriram Sankar, Kushagra Vaid, Christos Kozyrakis, “Accurate Modeling and Generation of Storage I/O for Datacenter Workloads”. Exascale Evaluation and Research Techniques (EXERT) Workshop.2011.

[3] Ningning Zhu, Jiawu Chen, Tzi-Cker Chiueh, “TBBT: Scalable and Accurate Trace Replay for File Server Evaluation”. FAST. 2005.

[4] Jon Tate, Pall Beck, Hector Hugo Ibarra, Shanmuganathan Kumaravel, Libor Miklas, “Introduction to Storage Area Networks and System Networking”.

[5] Ralph O. Weber, ”SCSI Architecture Model - 3 (SAM-3)”. T10 Technical Committee. 16 March 2002.

[6] Jelmer R. Vernooij, “The Official Samba-4 HOWTO”. http://www.samba.org/samba/docs/man/Samba4-HOWTO/protocol.html

[7] Jose Barreto, “SMB remote file protocol(including SMB 3.0)”. @Storage Networking Industry Association.

[8] Mircrosoft, “Common Internet File System (CIFS) Protocol”.

[9] IDC, http://online.wsj.com/article/BT-CO-20131106-709604.html.

[10] J. C. Mogul, ”Brittle metrics in operating systems research”. Proceedings of 7th Workshop on Hot Topics in Operating Systems. January 1999.

[11] IBM, “Network file systems and Linux”. http://www.ibm.com/developerworks/library/l-network-filesystems/.

[12] Kavalanekar S, Worthington B, Qi Zhang, Sharda V, "Characterization of storage workload traces from production Windows Servers". Workload Characterization, 2008. IISWC 2008.

[13] J. Katcher. "PostMark: A New Filesystem Benchmark". Technical Report TR3022, Network Appliance, 1997. www.netapp.com/tech_library/3022.html.

[14] VERITAS Software. VERITAS File Server Edition Performance Brief: A PostMark 1.11 Benchmark Comparison. Technical report, Veritas Software Corporation, June 1999. http://eval.veritas.com/webfiles/docs/fsedition-postmark.pdf.

[15] Transaction Processing Performance Council. Transaction Processing Performance Council. www.tpc.org, 2005.

[16] SPEC. The SPEC Organization. www.spec.org/, April 2005.

[17] SPC. Storage Performance Council, 2007. www.storageperformance.org.

[18] R. Coker. The Bonnie++ home page. www.coker.com.au/bonnie++, 2001.

[19] W. Akkerman. strace software home page. www.liacs.nl/~wichert/strace/, 2002.

[20] L. Mummert and M. Satyanarayanan. Long term distributed file reference tracing: Implementation and experience. Technical Report CMU-CS-94-213, Carnegie Mellon University, Pittsburgh, PA, 1994.

[21] J. Ousterhout, H. Costa, D. Harrison, J. Kunze, M. Kupfer, and J. Thompson. A Trace-Driven Analysis of the UNIX 4.2 BSD File System. In Proceedings of the 10th ACM Symposium on Operating System Principles, pp. 15-24, Orcas Island, WA, December 1985

[22] 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, San Diego, CA, October 2003.

[23] M. Blaze. NFS Tracing by Passive Network Monitoring. In Proceedings of the USENIX Winter Conference, San Francisco, CA, January 1992.

[24] C. Ruemmler and J. Wilkes. UNIX Disk Access Patterns. In Proceedings of the Winter USENIX Technical Conference, pp. 405-420, San Diego, CA, January 1993.

[25] J. Flinn, S. Sinnamohideen, N. Tolia, and M. Satyanaryanan. Data Staging on Untrusted Surrogates. In Proceedings of the 2nd USENIX Conference on File and Storage Technologies, pp. 15-28, San Francisco, CA, March 2003.

[26] N. Tolia, J. Harkes, M. Kozuch, and M. Satyanarayanan. Integrating Portable and Distributed Storage. In Proceedings of the 3rd USENIX Conference on File and Storage Technologies, pp. 227-238, San Francisco, CA, March/April 2004.

[27] Z. N. J. Peterson, R. Burns, G. Ateniese, and S. Bono. Design and implementation of verifiable audit trails for a versioning file system. In Proceedings of the 5th USENIX Conference on File and Storage Technologies, pp. 93-106, San Jose, CA, February 2007.

[28] V. Prabhakaran, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dussea. Analysis and Evolution of Journaling File Systems. In Proceedings of the Annual USENIX Technical Conference, pp. 105-120, Anaheim, CA, April 2005.

[29] E. Anderson, M. Kallahalla, M. Uysal, and R. Swaminathan. Buttress: A Toolkit for Flexible and High Fidelity I/O Benchmarking. In Proceedings of the 3rd USENIX Conference on File and Storage Technologies, pp. 45-58, San Francisco, CA, March/April 2004.

[30] CodeFX. “CIFS Explained”. 2001.