File-System Traces Analysis

Sudhir Kasanavesi
Department of Computer Science
Stony Brook University
SBU ID: 108492541

Nihar Konireddy
Department of Computer Science
Stony Brook University
SBU ID: 108395318

Kalyan Chandra Chintalapati
Department of Computer Science
Stony Brook University
SBU ID: 108080090

I. ABSTRACT

Traces are good sources of information in understanding the workload characteristics of a system [1]. A good trace analysis helps in understanding the behavior of the system under a real-time workload and these results would help in optimization of the system resources. File-system traces, in particular have been used for file system evaluation, user behavior analysis, file-system debugging and intrusion detection [3]. But traces tend to be large, hard to use and share. Often, however, researchers are not interested in the precise details stored in a bulky trace, but rather in some statistical properties found in the traces - properties that affect their system's behavior under load [1].

Network File System (NFS) is by far the most widely used file-system today. NFS client sends requests using Remote Procedure Call (RPC) to the server. NFS Server translates these RPC requests and accesses the local physical file system. NFS traces can be captured by sniffing the ethernet packets since the request-reply between client and server is achieved using RPC. The packets which are captured consists of NFS calls and replys, they have to be decoded and then recorded in the trace.

This paper aims at providing a framework to analyze the network file system. We have built a system that extracts some of the desired properties required for analyzing Network File System (NFS) Version3 and convert it to the DataSeries[2] format. This is fed to the analyzer built as a part of the system for analyzing several workloads. It does some basic analysis such as determining the number of NFS v3 procedures on a given workload, find the READ and WRITE I/O sizes etc on the traces captured. The network traffic is captured by tshark, network protocol analyzer. The results of our analyzer are evaluated with the "dsstatgroupby" tool which is a part of DataSeries [2] tools. Our system currently extracts few desirable properties from a subset of NFS V3 procedures and can be extended to make the user specify the properties which they intend to analyze.

II. INTRODUCTION

File system trace analysis would give a comprehensive understanding of the system under a specific workload. The richness of the information at file system level gives a much more meaningful support than block level trace information. The goals of the tools and methods described can be of great help to researchers and system administrators to understand the workload of NFS server, identify latent performance issues. They can be further used in the file system replayers to simulate the real-time workload. There might be situations where file system researchers might be content to know the number of reads, writes etc. and also want to know the names of those files and its path to introduce some optimizations. "If the system becomes swamped with requests for those files, the administrators might also want to identify the owner of the files and the users responsible for the requests so that those users can be consulted" [4].

We introduce a system which is targeted to analyze the Network File System Version3. Our system uses "tshark" - network protocol analyzer for capturing the NFS Call and Request packets, then convert it to DataSeries [2] format - which is an efficient structured serial data storage format. We have also built an analyzer module which does some basic analysis on the traces.

Section [III] briefly presents reasons for selection of NFS, related work and motivation for developing this system. Section [IV] discusses the design of the system and decisions involved for choosing specific network protocol analyzer tool, filebench benchmarking tool and DataSeries [2] format. In Section[V] we discuss in detail the implementation of our approach. In Section [VI], we present the results of our experiments in converting a trace generated by filebench workload, we also present the analysis done using our system on the trace collected. Finally, we conclude in section [VII] with an overview of future enhancements to our implementation.

III. BACKGROUND AND MOTIVATION

There are about 30 widely used file systems. Few prominent ones being ext2, ext3 and Network File System (NFS) file systems. The process of obtaining the trace information at file systems like ext2 or ext3 is relatively cumbersome task. Researchers have proposed that there are two different approaches in which a file-system (typically ext2 or ext3) trace could be captured.

- By using system call trace such as strace, but this would be an overkill because there would be lot of other system calls which might not be due to the I/O operations performed on the file system. One of the heuristic approach is to trace only I/O related systems, but we might miss out on tracing system calls like fork(). Such system calls would be important in trace analysis because we would be interested to see which file descriptors a particular process is inheriting from its parent when it is forked. This approach is complicated to filter out irrelavant system calls from the strace but it is possible.

- By intercepting system calls at VFS level. In this approach we can snoop the system calls related to file-system by modifying the kernel code at the level of file operations, inode operations or address-space operations. VFS Interceptor [3] is developed using this approach by modifying the VFS kernel code with a little overhead. Not many tools have been developed to extend this approach because it is not easy to modify the kernel code without much overhead.

NFS is by far the most widely used network file-system today. NFS client sends requests using RPC to the server. NFS Server translates these RPC requests and accesses the local physical file system. NFS trace can provide lot of information about the NFS workload. The system performance could be studied if the traces can be analyzed. Some of the basic interesting analysis that could be done using the traces are finding out the busiest server/client, finding out the hottest files (or types of files), find the most frequently used procedures. Doing such an analysis helps us to understand the behaviour of the system under workload and gives us pointers to optimize the system specific to the particular workload. Capturing traces also can yield in aiding for doing an advanced analysis such as finding the read/write ratio, getattr/read ratio, getattr/write ratio. For example, we can analyze the trace to find out how many getattr procedures are followed by read/write procedure. Depending on the ratio of getattr/read and getattr/write ratio, we can predict and prefetch the attributes to increase the system performance.

IV. Design

We present a system which (1) converts the trace captured by ”tshark” which consists of NFS Call and Reply packets to DataSeries[2] format. (2) performs some basic analysis on the NFS traces. This could further be integrated with T2M [7] to generate workload models from the traces for certain benchmark tools for understanding complex distributions in real-world traces. In this section, we present the workflow of the system along with several crucial design decisions in selecting specific tools and formats.

A. Need for trace collection

SNIA IOTTA Repository provides a repository of traces which were captured on some workloads. These traces could be used directly to analyze, but they do not provide information about the environment and workload under which the trace is captured. This would make it hard to speculate while analyzing the traces. So we captured the traces under a known enviroment using the workloads that are distributed along with the filebench - benchmarking tool. NFS client and NFS server is setup in two different virtual machines and work- load is generated using filebench.

B. Selecting network packet sniffer

NFS file-system traces can be captured by sniffing the ethernet packets since the request-reply between client and server is achieved using RPC. Some of the tools which capture the network traffic are tcpdump, snoop, tethereal and wireshark etc. tcpdump is one among the oldest command line tools available for this pur- pose. But doesnt have a lot of NFS smarts, however, so generally this is a tool that is used to capture network traffic to a file for later analysis by a tool like Wireshark that can dissect RPC and NFS traffic more completely. tcpdump (especially 3.7.2) generates the output in an irregular, difficult to parse or undocumented format. For example, tcpdump 3.7.2 prints file offsets for read and write operations in hexadecimal for NFSv3 requests and decimal for NFSv2 requests, but does not print whether the request is NFSv2 or NFSv3 [4]. nfsdump[4] is built to gather and analyze NFS traces. It sniffs packets, decodes NFS calls and responses, and records them in a text format. This tool was considered to capture the NFS trace, but since it has not been updated for over 5 years, we had to look out for other tools which are stable and continously maintained.

TShark is a network protocol analyzer. It lets you capture packet data from a live network, or read packets from a previously saved capture file, either printing a decoded form of those packets to the standard output or writing the packets to a file. TShark also provides some interesting features which will be of great use in performing an advanced analysis of the traces. TShark 1.65 allows NFS file name and path snooping which could be used in analyzing hottest files/path of particular workload. This along with other features, ease-to-use and flexibility made us choose TShark tool for capturing the traces.

C. System view

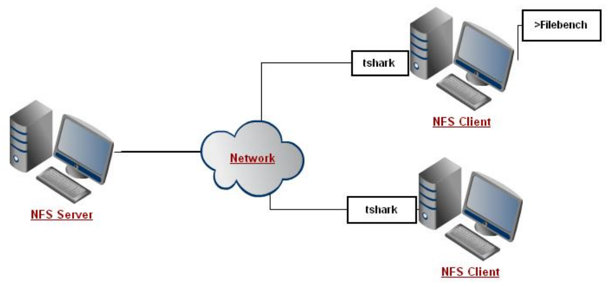

Fig. 1. Basic architecture of the system

Figure 1 shows the basic architecture of our system. We used VMware Player for setting up NFS server and NFS clients on the virtual machine. filebench workload is generated on the NFS clients after mounting the file system. This generate NFS Call and Reply packets which are collected using tshark running on the client.

D. DataSeries [2]

We chose the efficient and flexible DataSeries format [2] - recommended by the Storage Networking Industry Association (SNIA). ”DataSeries data model is conceptually very similar to that used by relational databases. Logically, a DataSeries file is composed of an ordered sequence of records, where each record is composed from a set of fields. Each field has a field-type (e.g., integer, string, double, boolean) and a name. A DataSeries record is analogous to a row in a conventional relational database” [2].

”A single DataSeries file comprises one or more extents (potentially with different extent-types), a header and extent-type extent at the beginning of the file, and an index extent and trailer at the end of the file. The extent-type extent contains records with a single string-valued field, each of which contains an XML specification that defines the extent-types of all the other extents in the file.” [2]

E. Workflow

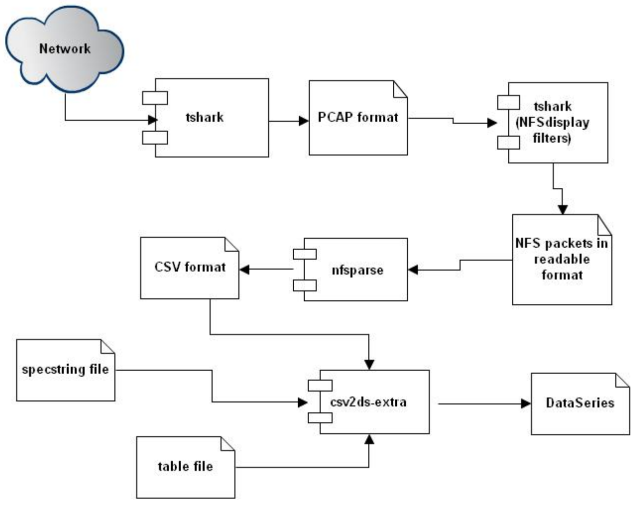

Fig. 2. Workflow of the system

Figure 2 shows the basic workflow of the system. tshark collects the raw trace information for the workload mentioned above and stores it in the pcap format. The collected trace is again filtered (using tshark nfs display filters) to preserve two important fields in the trace, they are RPC xid and file full paths (optional). RPC xid is a unique id accompanied with each RPC procedure call. NFS v3 supports asynchronous writes and many other operations can be performed asynchronously, so in order to correlate the call and reply of a specific NFS procedure, RPC xid is necessary. This is the reason for preserving RPC xid in the trace information. We have made it mandatory to apply NFS display filter to capture the RPC xid. However, file path snooping is an optional feature which when enabled in tshark will allow our system to capture it in the trace.

As shown in the Figure 2, the PCAP format is converted into a new readable format containing the fields of our interest. This is fed to ”nfsparse” tool which converts it into a csv file. This csv file along with table file of extent types [2] and specification strings [2] file is fed to to the ”csv2ds-extra” tool for generating the trace in DataSeries format. Additionally, ”nfsstat” (not shown in the figure) will do some basic analysis on the DataSeries file.

V. Implementation

We now give the implementation details of our design. We used two virtual machines using VMware Player, one of the virtual machine is configured as NFS server and the other virtual machine is configured as NFS client. Both NFS Server and Client are configured to support NFS V3 protocol.

A. System setup

We have installed CentOS 6.2 kernel on the virtual machine which is configured as NFS Server and Linux Mint 12 Lisa kernel on the virtual machine which is configured as NFS Client. Using ”vmnetcfg” tool which is packaged along with the installation of VMware Player, we have configured both virtual machines to be included in a private ”Host-only” network. We have also configured DHCP settings of the ”Host-only” network so that it assigns static IP addresses to both server and client. Using the static IP address of the client virtual machine, we have added an entry to the /etc/exports on the server virtual machine for the NFS server to accept and allow NFS requests.

TShark is a network protocol analyzer. It is a terminal oriented version of Wireshark designed for capturing and displaying packets when an interactive user interface isn’t necessary or available. It allows us to capture packet data from a live network, or read packets from a previously saved capture file, either printing a decoded form of those packets to the standard output or writing the packets to a file. TShark’s native capture file format is libpcap format. We have installed tshark version 1.65 on the NFS Client virtual machine.

Filebench is a file system and storage benchmark that allows to generate a large variety of workloads. We have installed filebench on the client machine to generate workloads and thus capture the ethernet packets which consists of NFS Call and NFS Reply packets using ”tshark”. Filebench, along with its installation provides with us with a set of predefined scripts which generate the workloads. For our experiments, we have use ”fileserver.f” script for generating the workload. This workload is selected because it could generate 11 out of 23 NFS v3 procedures which we thought was reasonable to support in building this prototype system.

B. Capturing and filtering ethernet packets

tshark is used to capture ethernet packets. These packets contain NFS Call’s and Reply’s and need to be filtered. In our experiments, we have captured these packets on the NFS Client, but this can be done on the NFS Server side as well. We have used the below command to capture packets using tshark and to store it in PCAP format.

```
# tshark -i <interface >-F libpcap -w <output_file >
```

But these packets need to be filtered for capturing only NFS Call's and Reply's. The above captured PCAP file can be again read using tshark and converted into a readable raw text format using the below command

```
# tshark -r <pcap_file >-R nfs -t e -z proto, colinfo, rpc.xid, rpc.xid -z proto, colinfo, nfs.full_name, nfs.full_name
```

This file which is filtered for NFS packets is used in further steps and finally converted into DataSeries format.

C. Defining extent types, table fields, specstrings

We have defined each NFS Call (and Reply) as a seperate extent-type. The argument for each procedure defines the fields of that particular extent. Some of the fields are obtained only after applying specific NFS display filters on tshark. For such fields, we have set opt_nullable as "yes" so if they are not present in the trace, the trace can still be converted to DataSeries format.

The extent-type, table fields and spec strings for one such procedure is illustrated below. After converting the raw trace to csv format, the read_request line is shown as below

```
read_request,5736614403894349824,0xf1667b09,0xda524a8d, 208896, 2872, "192.168.224.129:/home/bigfileset"
```

It is not clear what 5736614403894349824, 0xf1667b09, 0xda524a8d, 208896 etc mean. One needs to define a spec string that describes the format of the CSV file. The spec string for the same request is defined as

```
read_request(time_called, xid, file_handle, offset, length, full_pathname);
```

The table fields describes fields and their types that will constitute target DataSeries file. The table fields for the same request is defined as

```
<extent type ><field name ><is nullable? ><field type >
read_request time_called 0 int64
read_request xid 0 variable32
read_request file_handle 0 variable32
read_request offset 0 int64
read_request length 0 int64
read_request full_pathname 1 variable32
```

Using the table fields, we can generate xml schema. The correspoding xml schema for the extent type is

```
<ExtentType name="
    IOTTAFSL:Trace:Syscall:read_request"
    namespace="iotta.snia.org.fsl.cs.
    sunysb.edu" version="1.00">
  <field name="time_called" type="int64"
      opt_nullable="no"/>
  <field name="xid" type="variable32"
      opt_nullable="no"/>
  <field name="file_handle" type="
      variable32" opt_nullable="no"/>
  <field name="offset" type="int64"
      opt_nullable="no"/>
  <field name="length" type="int64"
      opt_nullable="no"/>
  <field name="full_pathname" type="
      variable32" opt_nullable="yes"/>
</ExtentType>
```

D. nfsparse

This is a tool developed in C++ which does unit conversion to a CSV file with the fields corresponding to the fields of a target DataSeries file. It iterates over every line in the raw trace file which is filtered for the NFS packets and generates the CSV file. This CSV file is fed to "csv2ds-extra" tool which converts the CSV to DataSeries format.

This tool needs a table file which explains the type of the fields in its target DataSeries file. It also need specstrings file which explains the semantics about various fields present in the trace. Below figure illustrates the control-flow of this tool.

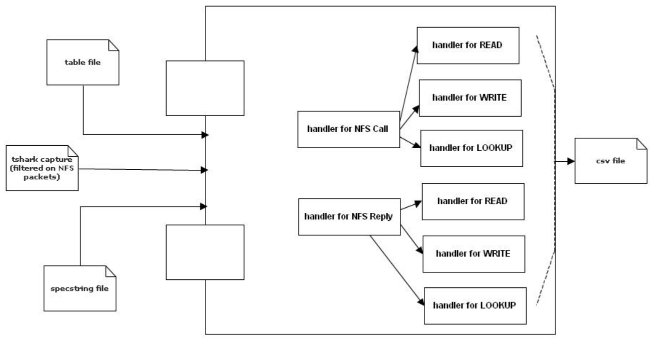

Fig. 3. Functionality of nfsparse

E. nfstrace2ds

This is a shell script which aggregates all the above operations. The input to this shell script are:

- trace captured by "tshark" in the PCAP format.
- table file which explains the semantics of the useful information needed to extract from the trace.
- specstring file which explains the order of the parameters of every procedure in the trace.

nfstrace2ds extracts the useful information specified from the trace and generates its equivalent file in DataSeries format. This file could be used to analyze the behavior of the system. It applies several NFS display filters on the PCAP trace file and extracts the NFS Call's and Reply's. We have used "-R nfs -t e -z proto,colinfo,rpc.xid,rpc.xid -z proto,colinfo,nfs.full_name,nfs.full_name" tshark options for filtering the NFS packets. "tshark" allows file path to full name snooping if it learns the file names before the file system is mounted by the client. The option "-z proto,colinfo,nfs.full_name,nfs.full_name" is made optional but rpc.xid is needed for analyzing the traces where we need to correlate between NFS Call and NFS Reply. The result of this filtered NFS trace is then stored in a temporary file which is used to invoke "nfsparse" which generates csv file for the given trace. This along with the table file and specstring file is fed to "csv2ds-extra" tool which is a part of DataSeries to convert the trace file to its corresponding DataSeries format.

F. nfsstat

This is a analyzer module which processes the trace in the DataSeries format. It uses the DataSeries API to perform some basic analysis on the traces. It tabulates the procedure counts (READ, WRITE, GETATTR, LOOKUP etc) for the workload which generated the traces. It also captures I/O sizes of the READ and WRITE procedures over the trace. This data can be used for analyzing mean I/O size for READ and WRITE procedures and also determining the behaviour of the system such as read-heavy or write-heavy etc.

VI. Analysis and Evaluation

As a part of our system, we have developed a small analyzer module which performs some basic analysis on the trace which we have converted into DataSeries format. We have used DataSeries RowAnalysis API for reading the DataSeries file. "nfsstat" is the module which reads the trace and displays the statistics about NFS V3 procedures. The results of our converter and analyzer are evaluated with "dsstatgroupby" tool which is a part of DataSeries installation tools.
In our experiments, we ran "fileserver" workload for 5 seconds and collected the trace using tshark in PCAP format using the command

```
tshark -i <interface >-F libpcap -w <output_file >
```

On the captured PCAP format,we have applied NFS display filters to extract desirable properties using the following command

```
tshark -r <pcap_file >-R nfs -t e -z proto, colinfo, rpc.xid, rpc.xid -z proto, colinfo, nfs.full_name, nfs.full_name
```

This is fed to "nfsparse" followed by "csv2ds-extra" to convert it into DataSeries format. The DataSeries file is fed to "nfsstat". "nfsstat" iterates through each and every row in the DataSeries file. fileserver workload was able to generate 11 different types of NFS V3 procedures. During 5 seconds of the workload, we analyzed number of times each procedure is requested by the NFS Client. Figure 4 displays the number of times specific NFS V3 procedure is requested.

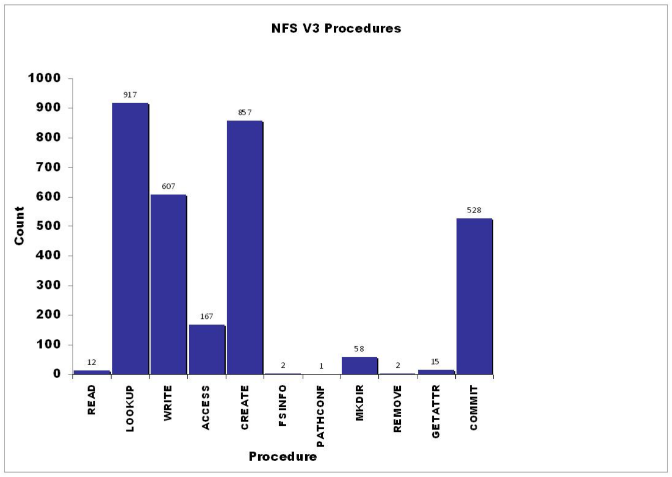

Fig. 4. NFS Procedure vs Count

The table below displays the NFS V3 procedure counts observer during our experiment.

PROCEDURE	COUNT
READ	12
WRITE	607
LOOKUP	917
ACCESS	167
CREATE	857
FSINFO	2
PATHCONF	1
MKDIR	58
REMOVE	2
COMMIT	528
GETATTR	15

"nfsstat" also calculates the mean, standard deviation of I/O sizes for the READ and WRITE procedures. The results of the analyzer are evaluated with DataSeries tools. The table below displays the results of "nfsstat" for the fileserver workload.

PROCEDURE	COUNT	MEAN	STD DEVIATION
READ	12	36021.4	29949.7
WRITE	607	44562.3	22332.8

We have also analyzed the I/O sizes of the READ and WRITE procedures. The fileserver worload is observed to have more WRITE procedures when compared to the READ procedures. Figure 5 and 6 plots the graphs between I/O size and its corresponding READ and WRITE procedures.

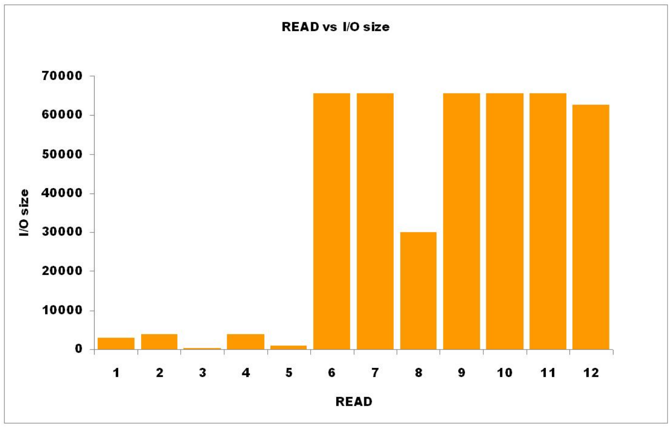

Fig. 5. NFS READ Procedure vs I/O size

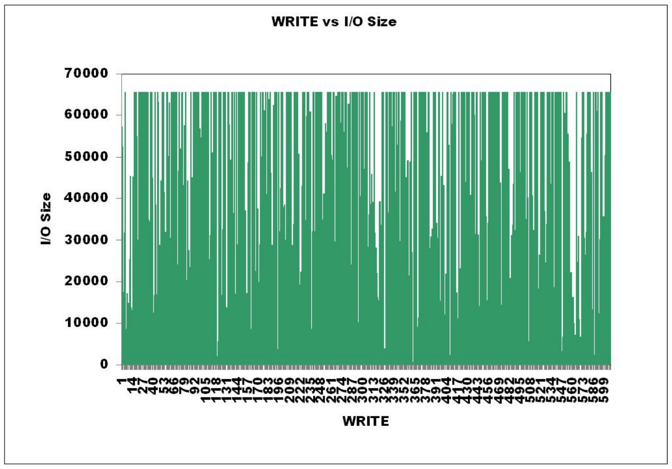

Fig. 6. NFS WRITE Procedure vs I/O size

VII. Conclusion and Future Work

NFS traces provide a wealth of information about an NFS workload. Our system simplifies extracting this information and performs some useful analysis. We have created a system that currently supports only 11 NFS V3 procedures i.e. CREATE, READ, WRITE, LOOKUP, GETATTR, ACCESS, REMOVE, MKDIR, FSINFO, PATHCONF, COMMIT. If the trace captured contains other procedures, our system will ignore the packets and continue further. In future we plan to extend the system to support all other procedures. We have also built a small analyzer module "nfsstat" which does some very basic analysis on the DataSeries file as described in the above sections. We can extend the functionality of this module to perform some advance statistical analysis which would be helpful in understanding the system behaviour.

Also, we can integrate this system with T2M [7] to generate workload models from the traces for certain benchmark tools for understanding complex distributions in real-world traces.

Acknowledgment

The authors would like to thank Professor Erez Zadok and Vasily Tarasov for their valuable help and guidance thoughout the project.

References

[1] Extracting Flexible, Replayable Models from Large Block Traces - V. Tarasov, S. Kumar, J. Ma, D. Hildebrand, A. Povzner, G. Kuenning, and E. Zadok Stony Brook University, Harvey Mudd College, and IBM Almaden Research
[2] DataSeries: An efficient, flexible data format for structured serial data - DataSeries Technical Documentation
[3] VFS Interceptor: Dynamically Tracing File System Operations in real environments - Yang Wang, Jiwu Shu, Wei Xue , Mao Xue
[4] New NFS Tracing Tools and Techniques for System Analysis - Daniel Ellard and Margo Seltzer
[5] Capture, conversion, and analysis of an intense NFS workload - Eric Anderson, HP Labs
[6] RFC 1813 - NFS Version 3 Protocol Specification
[7] T2M: Converting I/O Traces to Workload Models - Vasily Tarasov (student, presenter), Koundinya Santhosh Kumar (student) and Erez Zadok (Stony Brook University) Geoff Kuenning (Harvey Mudd College)