Performance analysis of a Linux based FTP server

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Master of Technology

by
Anand Srivastava

to the
Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur
July 1996

Certificate

Certified that the work contained in the thesis entitled "*Performance analysis of a Linux based FTP server*", by Mr.*Anand Srivastava*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

(Dr. Dheeraj Sanghi)
Assistant Professor, Department of Computer Science & Engineering,
Indian Institute of Technology,
Kanpur.

July 1996

Abstract

Linux over the past couple of years has matured to the point where it has been accepted as a viable platform for server applications. This transformation is due to its stability and support provided by a few companies. Currently it is being used by Internet Service Providers. Linux will be accepted for more serious applications only if it can handle heavy loads. This thesis analyzes the performance of Linux in one such application, the FTP server. Several experiments were conducted to determine the performance under different conditions. Conclusions based on these experiments are drawn and given in this thesis. Experiments show that Linux performs quite well under heavy loads.

Acknowledgments

I am grateful to **Dr. Dheeraj Sanghi** for his constant guidance throughout this work and for providing an opportunity to get acquainted with the internal working of a real operating system, Linux. I would like to thank all the people who have contributed their efforts to the free software movement. I would also like to thank the **Class of '94** who made my stay here a beautiful and memorable experience.

Contents

Chapter 1

Introduction

1.1 Motivation

Linux over the past couple of years has matured to the point that it is now perceived as a viable operating system for servers. The transformation is partly due to its free availability, and availability of software for the platform. But the more important reason is that the kernel has proved its stability over the time, and that a few companies like *Caldera* and *Cygnus* have come forward for providing support. It is currently being used by the Internet Service Providers, because they are small businesses and want to keep their investments low. The move to more demanding applications will depend on the ability of Linux to handle heavy loads. One such application is an FTP server. The motivation for this thesis was to analyze the performance of an FTP server, and to find out how it reacts to variations in load.

Linux is a full-featured 32-bit operating system. It is a fully POSIX-compliant operating system.

File **T**ransfer **P**rotocol [PR85] is the standard for transferring files between different hosts.

1.2 The Server

Our server was housed on an *Intel*-486 based personal computer. It is connected to other machines with a 10Mbps ethernet link. The FTP server, is based on the **Ext2fs** (Second Extended file system) [TCT]. This file system has become the de-facto standard for Linux. It allows variable block sizes. We have used 4KB block size. We used the Linux kernel version 1.3.35.

1.3 Organization of Thesis

Rest of the thesis has been organized as follows.

In **Chapter 2** we will discuss the properties of Ext2fs (file system) and buffer cache which have an effect on Linux performance.

In **Chapter 3** we will explain the implementation of Linux's buffer cache, and how it works.

In **Chapter 4** we will explain how the experiments were conducted. We will see what modification were made to the server, and what are the types of experiments.

In **chapter 5** we will discuss the results of experiments conducted and how they relate to the expected results.

Finally, in **chapter 6** we will conclude by summarizing the results that we have obtained from the experiments.

Chapter 2

Introduction to Linux

In this chapter, we explain the design aspects of file system and buffer cache in Linux that help in improving performance.

2.1 The File System (Ext2fs)

The file system manages files on disk, or on other mass storage devices. It allows processes to read, write, and modify files by bringing the required portions of files into main memory, and writing them out as required. It provides a uniform interface for accessing files on different storage devices. Following features of Ext2fs help in improving file system performance.

Block Groups The Ext2fs is divided into block groups. The file system tries to allocate all blocks of a file in one group. This reduces seek time and increases file system performance.

Different block sizes Ext2fs allows creating file systems with different logical block size. It currently supports 1K, 2K, and 4K block sizes. File systems with large block size give better disk performance as more data is read per disk read.

On the other hand using bigger logical blocks wastes disk space. This is because a typical Unix file system is composed mainly of small files [Bac86]. This problem is eliminated by using fragments. Fragment is part of a logical block. A file uses only as many fragments as necessary for the last incomplete block.

Read ahead It has been observed that most file reads are sequential. This allows Ext2fs to read a few consecutive blocks, along with the requested blocks. These blocks, which are read before they are actually requested, are called read ahead blocks. The number of read ahead blocks is different for different devices. Read ahead blocks are read only when an unread block is requested.

2.2 The Buffer Cache

The buffer cache manages portions of files that are in memory. Its objective is to improve performance during file access. In this section we explain the features that affect system performance.

Buffers of different size Linux buffer cache supports buffers of sizes 512 bytes, 1KB, 2KB, and 4KB. This is necessary because the supported file systems use these block sizes. For example Ext2fs allows 1KB, 2KB, 4KB block sizes, while msdos file system, allows 512 bytes or 1KB block sizes. Supporting different block sizes increases the complexity of the buffer cache design and has a detrimental effect on the file system performance.

Dynamic allocation Memory for the buffer cache is allocated dynamically and can occupy almost all of the available memory. On a machine with 8MB memory, it was observed, that it usually occupied 4MB, when the system was not working on memory intensive applications. In some cases, when file intensive applications were running, it was observed that the memory usage went above 5MB. The dynamic allocation effects the performance in many ways as we see in the next section.

2.3 Effects of Dynamic Buffer Cache

We discuss in this section the ways in which the presence of a dynamic buffer cache affects the performance of the system.

When the system is working under nearly constant load, buffer cache stabilizes to an optimal memory usage level.

Memory manager can free pages as soon as the processes using them have exited. However buffer cache cannot do so, as it does not have information about which buffers are being used by which processes. It therefore tries to free buffers which have the least chances of reuse. Thus memory used by the buffer cache is reused slowly.

Buffer cache is helpful only if the files are reused, otherwise it results in inefficient memory utilization.

Chapter 3

Design and conduction of Experiments

In this chapter we detail how and why various experiments were conducted. In the first section we discuss the design of various experiments conducted. In the next section we outline the way in which the experiments were conducted.

3.1 Design of Experiments

We need to control the conditions so that we can isolate factors affecting the performance of an FTP server. The FTP server, being based on file input output, uses the buffer cache quite intensively and buffer cache of Linux is dynamic in nature. This would prevent us from performing experiments one after another, because the first one would affect the second one. There may also be some random errors introduced, because our ethernet link is being shared by other systems as well. So, we need to repeat our experiments and then average the results. We also need to discard some initial repetitions, so that the influence of previous experiments is nullified.

We decided to conduct 15 experiments discarding the first five. This would provide us with constant load conditions.

We created files before conducting each experiments, to ensure that a file was never in the cache before an experiment was conducted. The files would be created afresh even for repetitions of an experiment.

We also had a 1 minute gap between two experiments. The Linux kernel 1.3.35 used in this experiment is not a very stable kernel and had the problem that a couple of processes would go into an infinite sleep unpredictably. This problem would occur very frequently when there was no time gap in between two experiments and we had started a large number of connections.

We conducted several experiments to find out the data transfer rate achieved by the server, under various load conditions. The transfer rate depends on the following parameters.

File size: A bigger file size would reduce the transfer rate, because more buffers will be required. The buffer cache needs to free previously used buffers. This will have very small effect on transfer rate. File size of 1MB was used for most of the experiments because we had observed that buffer cache occupied around 4MB usually. So, 1MB would be an ideal size to bring out the affects caused by buffer cache.

Number of connections: When more than one processes are trying to transfer files, they will be competing against each other for memory, and CPU. This would reduce the transfer rate. The system transfer rate, in case of multiple simultaneous connections, is taken as the overall data transfer rate for all connections. Overall data transfer rate gives the correct system performance only if all connections transfer files simultaneously.

Type of experiment: Files transferred over different connections may be same or different. When transferring same file, due to buffer cache, the file will have to be read from disk only once. Also the memory requirement will be less than the case of transferring different files. This will give a better transfer rate.

The following experiments test the effect of the above parameters on system performance. The experiments have been repeated as explained above to keep the load constant.

Loop Back experiment: This experiment was conducted keeping the client and the server on the same machine without going over the ethernet. It was used to show that the ethernet would be a bottleneck, and it would also show the load condition at which the system memory becomes insufficient. We transferred 1MB files over 1, 2, 4, and 6 simultaneous connections.

Varying number of connections: This experiment was conducted over the 10Mbps ethernet. It was done to find out the way in which FTP performance varies with number of connections. In this experiment 1MB files were transferred over 1, 2, 4, 8, 16, and 24 connections.

Varying sizes of files: This experiment was done to find out the way in which FTP performance is affected with file sizes. It was also conducted over the ethernet. Here we transferred 256KB, 1MB, 4MB, and 16MB files over 4 simultaneous connections.

We also wanted to find out how the performance varies with variations in load. This was done by comparing the transfer rates achieved with constant load and the transfer rates achieved with changing load. While the experiments under constant load where conducted by repeating experiments consecutively, experiments under changing load were repeated only after a full set of experiments was completed. We also wanted to highlight the effects of buffer cache, so we conducted these experiments in the order of decreasing load. In these experiments 1MB files were transferred over 6, 4, 2, and 1 simultaneous connections. The experiments were conducted in the order as detailed below.

The experiment for 6 connections with same file was done first, then experiment for 6 connections with different files was done. After that experiments were repeated with 4 connections, then with 2 connections and finally with 1 connection. After experiment with 1 connection, all experiments are repeated starting from experiments with 6 connections. We conducted 5 repetitions of the experiments in this way.

3.2 Conducting the Experiments

3.2.1 Working of Server Side

We used the **wu.ftpd** server daemon available with Linux. The source for the server is available at various web sites.

To test the server under heavy load conditions, we do multiple simultaneous transfers. These transfers should start around the same time, so that overall data transfer rate would give a correct measure of system performance. Each transfer is served by a separate server process created by the FTP daemon. We needed some mechanism to synchronize the server processes. We did this by stopping all the server processes before they started transferring data. When all the server processes were ready to transfer data, they were allowed to continue.

The `RETR` command [PR85] is used by an FTP client to retrieve a copy of the required file from an FTP server. The `STOP` signal can be used to stop a process, which can be later continued by the `CONT` signal [Ste92]. The server was modified so that on receiving the `RETR` command the server process would send a `STOP` signal to itself.

The server was also modified to log the file name, start time and the end time for a transfer. The data was logged to the log file `/var/adm/xferlog`, which is the default log file for **wu.ftpd**. This data was used for calculating the data transfer rate. Time was obtained using `gettimeofday` system call. This call returns the time in microseconds.

3.2.2 Working of Client Side

An experiment consists of creating the files to be transferred, then forking FTP processes which would transfer these files, and then deleting those files. The transfer rate is calculated according to the values stored at the server. A number of experiments are conducted in succession. The logged data should contain enough information to distinguish between the various experiments. The following information is embedded in the file name as period separated fields.

- file size
- different or same file was transferred
- experiment number for differentiating between repetitions of one experiment
- number of connections
- file number used to differentiate between different files in one experiment

We have a main program which is used to conduct the experiments. The main program runs on the client system. It uses the `rsh` (remote shell) program for executing commands on the server system, e.g. to create files on the server system. First it creates the files that are transferred during the experiment. Then it forks FTP clients which make connections to the server. Main program then remotely executes a shell script on the server. This script waits for the server processes to reach the stopped state. At this point the script sends `CONT` signal to all server processes. Then it waits for the server processes to complete the transfers. The script returns when the transfers are complete. The main program, which is waiting for the script to end, then removes the files that were created at the beginning of the experiment.

3.2.3 Calculation of Results

To calculate the overall transfer rate achieved in an experiment, we first find out the time when first transfer was started and the time when last transfer was completed. Using these two values we calculate total time taken for the experiment. Total amount of data transferred is calculated from the file size field in the file name. The overall transfer rate is calculated by dividing the total data transferred by total time taken. Since there are several repetitions of an experiments we find the average overall transfer rate. To confirm that all the experiments do start at around the same time we also calculate the average transfer rate over individual connections. If the average transfer rate multiplied by the number of connections is not much greater than the overall transfer rate, then the experiments must have started at nearly the same time. If we are suspicious that transfers of some experiment did not start at the same time then the actual data can be seen.

A Perl (Larry Wall's interpreted systems language [WS91]) script is used for parsing the log file, calculating the overall transfer rate for each experiment and preparing tables.

Chapter 4

Experiments and Results

In this chapter we discuss the experiments that we have conducted and analyze the results. We will use the notation (n,*type*) for an experiment which uses n simultaneous connections for transferring *type* files. For example (4,same) would mean transferring same file over 4 simultaneous connections.

4.1 Loop Back Experiments

4.1.1 Constant Load Experiments

In this section we discuss the results of the loop back experiments under constant load. We conducted these experiments to confirm if the 10Mbps ethernet will be a bottleneck. These experiments are not conducted over the ethernet and can have higher transfer rates than the bandwidth limit of ethernet. We transferred 1MB files over 1, 2, 4, and 6 simultaneous connections. The results of the experiments are given in the following table.

Type of transfer	Number of connections			
	1	2	4	6
Same file	1388.82	1393.44	1350.22	1257.11
Different files		1200.11	500.23	328.27
Overall transfer rates in KBps, 1MB files				

Table 1: Performance at constant load in loop back experiments

▮ *Analysis of Results*

For the case of transferring same files we observe that transfer rate is almost same from 1 to 2 connection and then there is a decrease in transfer rate. The first slight increase could be attributed to the slight time saving, because both processes would wait simultaneous for a disk read. The gradual decrease afterwards is due to CPU becoming a bottleneck, due to process switching overheads.

For the case of transferring different files we observe that transfer rate drops from 1 to 2 connections only moderately because memory is enough for two connections. The drop is because some buffers will have to be made free for reuse. From 2 to 4 connections there is a huge drop because the buffer cache is not sufficient to take in the 4MB of 4 files at one time. Working in the limited memory would require freeing and reusing buffers at a larger scale, along with processor switching overheads. Due to the same reasons we get a large drop again when going from 4 to 6 connections.

4.1.2 Changing Load Experiments

In this section we discuss the experiments that we conducted to find out how change in load affected the transfer rate. The experiments were conducted in the order of decreasing load. 1MB files were used with 1, 2, 4, and 6 simultaneous connections. The following table shows the results of the experiments.

Type of transfer	Number of connections			
	6	4	2	1
Same file	1062.94	1083.88	983.27	947.52
Different files	350.50	476.70	1067.44	
Overall transfer rates in KBps, 1MB files				

Table 2: Performance with changing load in loop back experiments

Analysis of Results

If we compare these results with the results in table 1, we observe that there is a general reduction in the transfer rates except for the case of experiment (6, different). The reason for the better performance in this case is that these experiments use substantial amounts of buffer cache. In the case of constant load these experiments were conducted consecutively, requiring the next repetition to free buffers. While in this case the experiment has been conducted after the experiment (6, same). These experiments use substantial amount of memory pages through virtual address space. These pages are free when the processes exit allowing the next process to find memory easily.

The opposite is true for the experiments transferring same files. In addition the effects of buffer cache are also carried over to the next few experiment. This is how the buffer usage of (6, different) affects (4,different) and also how (4,different) affects (2,different). This also explains why (1,same) is lower than (6,same).

We also note a counter intuitive result in that (2,same) is lower than (2,different). Ordinarily it should not happen because in the first case only one file has to be read from disk, while in the second case two files have to be read. The transfer rate of (2,same) is low because of high buffer usage by (4,different), while the transfer rate of (2,different) is high because of low buffer usage by (2,same).

4.2 Non-Loop Back Experiments

4.2.1 Varying Number of Connections

In this section we will discuss how the performance varies with increasing number of connections. The experiments comprised of transferring 1MB files on 1, 2, 4, 8, 16, 24 simultaneous connections. The results of the experiments are as follows.

Type of transfer	Number of connections					
	1	2	4	8	16	24
Same file	457.53	517.69	523.80	530.01	516.49	503.47
Different files		480.36	421.32	302.75	229.08	199.36
Overall transfer rates in KBps, 1MB files						

Table 3: Performance with varying number of connections over ethernet

▮ *Analysis of Results*

For transferring same files we observe that the ethernet is a bottleneck. The transfer rate is nearly same for almost all the values. The difference between (1,same) and (2,same) is large. This is because one out of two processes can start transferring files faster than one process. This further helps in the case of (4,same) and (8,same) because substantial amount of memory pages are used which allows the processes to start transferring faster. For (16,same) and (24,same) process switch times become substantial and start affecting the transfer rate.

For transferring different files we observe that (2,different) is being affected by its use of buffer cache. Being two processes is helping the transfer rate by allowing it to start faster but having to free buffers is reducing the gain that can be obtained. For 4 and more connections the memory becomes a bigger bottleneck than the ethernet and the performance reduces with increase of connections. The performance drop is less than 30% every time number of connections is doubled.

4.2.2 Varying File Sizes

In this section we will discuss our experiments used to determine the transfer rate as a function of file sizes. We transferred files of 256KB, 1MB, 4MB, and 16MB over 4 connections. The results are given in the following table.

Type of transfer	File size			
	256KB	1MB	4MB	16MB
Same file	520.05	510.75	483.45	494.88
Different files	531.44	405.01	321.57	306.10
Overall transfer rates in KBps, 4 connections				

Table 4: Performance with varying file sizes over ethernet

Analysis of Results

We observe from the above table that in the case of transferring same file there is very small difference between the different transfer rates. We observe that transfer rate in experiment (256K,same) is lower than (256K,different). The four processes in (256K,same) wait in a queue to access the first block. While the four processes in (256K,different) can proceed independently. Providing the opportunity to start transferring data sooner.

For the case of transferring same files we observe that the transfer rates are nearly constant. The transfer rate reduces from 256KB to 4MB, because the processes need to free an increasing number of buffers. After 4MB a limit is approached and then the transfer rates should be nearly constant.

We also observe that transferring different files larger than 1MB, reusing buffers poses a performance bottleneck. The transfer rate would become nearly constant a little above 4MB, because we observe very little difference between (4M,different) and (16M,different).

Chapter 5

Conclusions

We have examined the performance of Linux under various load conditions and have also analyzed how it reacts to changes in load. According to our experiments we have obtained the following results.

- Under moderate loads a 10Mbps ethernet poses a performance bottleneck.
- For the case of transferring different files we find that the 8MB RAM becomes a bigger bottleneck than the ethernet with just 4 connections. The rate of transfer reduces by less than 30% when number of connections are doubled. This means that Linux scales quite well with load.
- For the case of transferring same files we find that the memory is sufficient for around 16 connections. Memory would pose a bottleneck for more number of connections.
- Transferring different files reduces the performance but the fall in performance reduces as the files become very large.
- In the loop back experiment we discovered that there was a very large fall in transfer rate from 2 to 4 connections when transferring different files.

From these observations we can conclude that Linux does scale quite well, and will be able to handle heavy loads.

5.1 Future Work

We have observed that the ethernet is the major bottleneck. It would be useful to find out how Linux would perform given a 100Mbps ethernet, because then the link would not be a bottleneck.

References

[Bac86] Maurice J. Bach. *The Design of the UNIX Operating System.* Prentice-Hall, Englewood Cliffs, NJ 07632, USA, 1986.

[PR85] J. Postel and J. Reynolds. File transfer protocol (ftp). Technical Report RFC-959, Network Working Group, 1985.

[Ste92] Richard Stevens. *Advanced Programming in the UNIX Environment.* Addison-Wesley, Reading, MA, USA, 1992.

[TCT] Theodore Tsó, Remy Card, and Stephen Tweedie. Design and implementation of the second extended filesystem. In *Proceedings of the First Dutch International Symposium on Linux.*

[WS91] Larry Wall and Randal L. Schwartz. *Programming Perl.* Nutshell Handbooks. O'Reilly and Associates, Inc., 632 Petuluma Avenue, Sebastopol, CA 95472, first edition, January 1991.