RJ 10244 (A0205-022) May 23, 2002
Computer Science

IBM Research Report

Workload Characterization and Resource Usage Patterns for a Linux SAMBA Server

T. Paul Lee
IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Workload Characterization and Resource Usage Patterns for a Linux SAMBA Server

T. Paul Lee
IBM Almaden Research Center
San Jose, CA 95120
tpl@almaden.ibm.com

Abstract

Samba is a suite of Unix applications that speak the SMB/CIFS protocol to serve files to a plethora of heterogeneous operating systems and hardware platforms. In this study, we analyze two workloads to gain insight on message exchanges and resource usage patterns. One is that of a simple interactive "rename" operation; its trace allows us to see the typical message exchanges between a Linux Samba server and a Windows NT client. The other is that of the well-known NetBench benchmark which measures throughput and response time with a simulated set of typical Win32 application scripts. We use Linux system call tracing facilities and Samba logging features to characterize the workload in terms of file system pathnames, system calls, and traffic patterns. This technique and insight allows the proper planning and monitoring of Samba servers which are becoming more popular in modern-day corporate computing environments.

1. Introduction

Samba [1] is a suite of Unix applications that speak the SMB (Server Message Block) protocol. SMB is a client-server, request-response protocol for sharing files, printers, serial ports, and communication abstractions such as named pipes and mail slots between computer systems. It dates back to the mid-80s as an early IBM PC network protocol and was subsequently developed further by Microsoft and others [2]. Many of the documents for SMB are available at Microsoft FTP site [3]. Microsoft has also contributed materially by putting forward its definition of SMB and the Common Internet File System (CIFS/1.0), as a public Request for Comments (RFC), a standards document. This preliminary draft remained as working documents for the Internet Engineering Task Force (IETF) after its expiration in 1998. Nevertheless, the de facto standard status of this important protocol prevails in spite of many unfinished pieces.

Linux-based Samba servers are becoming more popular and visible in modern-day corporate computing environments [4,5]. In the data-centric heterogeneous client-server model, file servers like SAMBA make data available to clients of all sorts in a seamless fashion. In this study, we analyze two workloads to gain insight on message exchanges and resource usage patterns. One is that of a simple interactive "rename" operation; its trace allows us to see the typical message exchanges between a Linux Samba server and a Windows NT client. This is presented first as a simple example for understanding the SMB/CIFS protocol.

The other is that of the well-known NetBench [6] benchmark which measures throughput and response time with a simulated set of typical Win32 application scripts. This benchmark is often used to compare SMB/CIFS file servers for their serving capacities. We use Linux system call tracing facilities and Samba logging features to characterize the workload in terms of file system pathnames, system calls, and traffic patterns. This technique can be also applicable to on going monitoring of Linux-based SAMBA servers.

2. Workload Characteristics of a Simple "Rename" Operation

To gain insight on the SMB/CIFS message exchanges and the resource usage pattern, we start with a simple "rename" operation of a file object. In a typical command language, we run "rename *oldname newname*" to change the name of a file object from *oldname* to *newname*. Most operating systems and protocols have direct support for this common operation either as standard system call or as a transaction operation code. This experiment uses a Windows NT 4.0 Explorer as client shown in Figure 1. You single click twice to allow the GUI to highlight and open up a box to enter a new name for the file object called "*oldname*". We type in "*newname*" with "Enter" key to complete the operation.

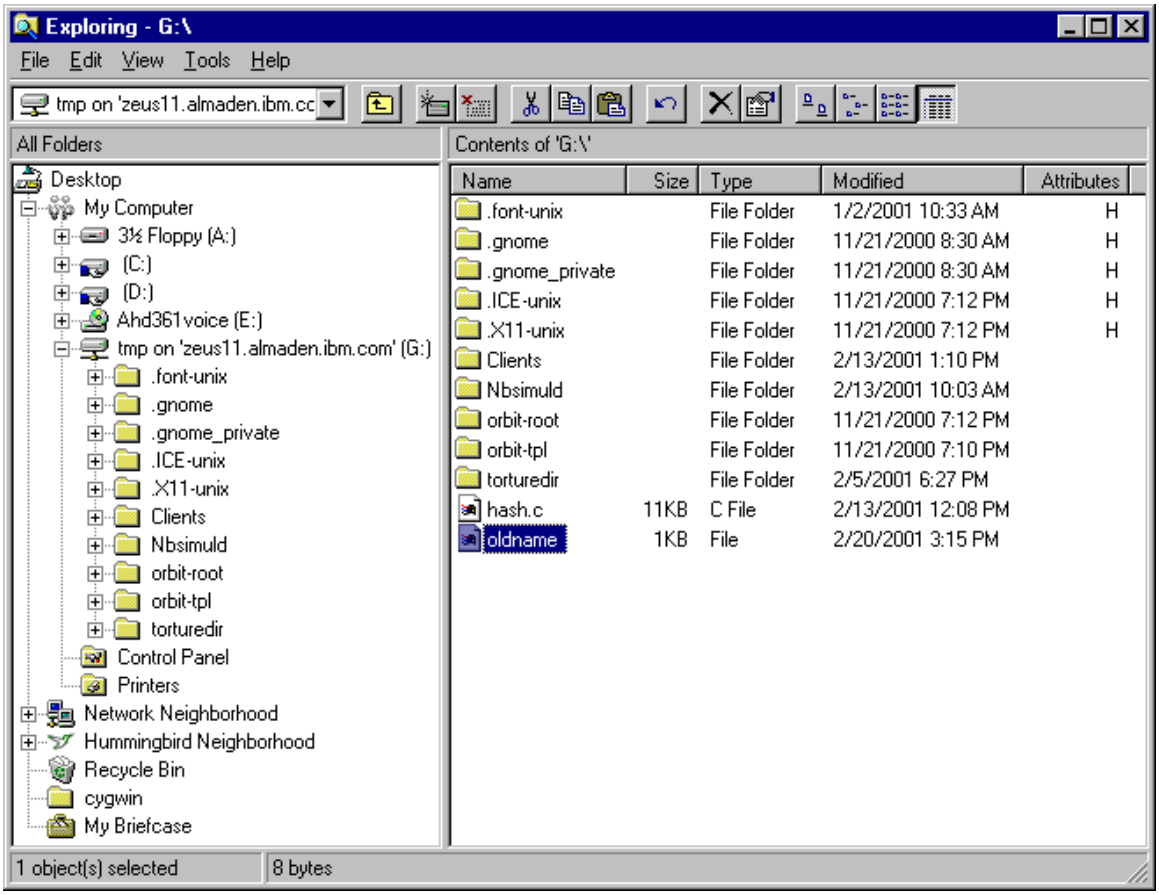

Figure 1: A Simple Rename Operation with Windows NT Explorer

On the server side, the Samba daemon is started as *"smbd –D –d 10*". This starts the Samba as a daemon process that detaches itself from the terminal, and runs in the background with debug level set to 10. This produces lots of debug and tracing information to a log file which we extract transaction information to produce Tables 1 and 2. The extraction processing is straightforward using Linux command "*grep smb_com –C5 logfile*" [7]. In essence, we are extracting all occurrences of "*smb_com*" keyword along with its 5-line context.

We have observed a total of 37 transactions for this "rename" operation in a Windows NT environment as shown in Tables 1 and 2. We do not show the first eight transactions when Windows' Explorer first accesses and opens the top-level directory as shown in Figure 1. These CIFS transactions include initial protocol negotiation, tree connect request, querying file system information, reading the root directory for display purposes, opening the root directory, and asking the server to notify change in the root directory.

The client implementation, that is, the Windows NT Explorer, uses the request/response SMB/CIFS protocol taking appropriate steps to ensure the desired file system properties and integrity constraints are observed and maintained before and after the operation. There are total of 37 transactions and we break this down to two tables of manageable sizes for illustration purposes. At the highest level, a rename operation has to verify first that the original name exists with proper attributes, second that the new name does not exist, and then the "move" operation is issued to the server for the desired change. Nevertheless, lower file-system specific details have to be considered by the client implementation such as case sensitivity (or insensitivity) and name mangling possibility. This usually adds a few exchanges of messages somewhat unexpectedly by the normal users of the system. It is also useful to note that "FindFirst" (a trans2 subcode) subcommand is used to search directory for the desired filename or filename pattern.

Table 1: CIFS Transactions for Rename Operation (part 1/2)

Transaction Number	CIFS transaction	Sub-commands	Argument	Return	Remark
9	(0xA2)NTcreateX		\oldname	Fnum 4527	(starting point)
10	(0x04)Close		4527		
11	(0xA2)NTcreateX		\oldname	Fnum 4528	(second time)
12	(0x04)Close		4528		
13	(0xA2)NTcreateX		\oldname	Fnum 4529	(third time)
14	(0x2E)readX	4529	Max=min=8	Nread=8	
15	(0x04)Close		4529		
16	(0xA2)NTcreateX		\oldname	Fnum 4530	
17	(0x2E)readX	4530	Max=min=8	Nread=8	(read again)
18	(0x32)trans2	Qfileinfo,0x104	. (dot)		
19	(0x32)trans2	Qfsinfo, 0x105			
20	(0x32)trans2	Findfirst	\oldname		
21	(0x32)trans2	Findfirst	\newname	Error code	
22	(0x04)Close		4530		

In Table 1, "NTCreateX" transaction is used to open the file object with name "*oldname*". At some point, the file length information is obtained so that a "readX" transaction is issued to read the content. It is not clear why this client has to issue three NtcreateX requests before it issues the readX request. Different clients might have different implementations for similar operations.

Transactions 18 through 22 basically query file system tree information and check the existence of "*oldname*" and non-existence of "*newname*" in the current directory. Note that the number of bytes read happens to be 8 bytes long since the content of the file is initialized to be "oldname\r" in our example.

Table 2: CIFS Transactions for Rename Operation (part 2/2)

Transaction Number	CIFS transaction	Sub-commands	Argument	Return	Remark
23	(0xA2)NTcreateX		\oldname	Fnum 4531	(until #43)
24	(0xA2)NTcreateX		\newname	Error code	
25	(0xA2)NTcreateX		\newname	Error code	(second time)
26	(0xA2)NTcreateX		\newname	Error code	(third time)
27	(0x08)getattr		\newname	Error code	
28	(0x32)trans2	Qfilepathinfo,0x101		\oldname	
29	(0xA2)NTcreateX		\oldname	Fnum 4535	(nested open)
30	(0x04)Close	4535			
31	(0xA2)NTcreateX		\oldname	Fnum 4536	(nested open)
32	(0x32)trans2	Qfileinfo,0x102	\oldname		
33	(0x2E)readX	4536	Max=min=256	Nread=8	
34	(0x2E)readX	4536	Max=min=512	Nread=8	(read again)
35	(0x32)trans2	Qfileinfo,0x102	\oldname		(same as #32)
36	(0x2E)readX	4536	Max=min=2	Nread=2	(read third time)
37	(0x32)trans2	Qfileinfo,0x102	\oldname		
38	(0x32)trans2	Qfileinfo,0x102	\oldname		
39	(0x32)trans2	Qfileinfo,0x102	\oldname		
40	(0x2E)readX	4536	Max=min=8	Nread=8	(read exact)
41	(0x2E)readX	4536	Max=min=4096	Nread=8	(read last time)
42	(0x04)Close	4536			(open at #31)
43	(0x04)Close	4531			(open at #23)
44	(0x07)move		.\oldname .\newname		(THE command)
45	(0x32)trans2	Findfirst, 0x104	\newname		(all done)

In Table 2, transactions 23 through 43 encompass a large nested open loop to ensure that the "*newname*" does not exist, the "*oldname*" has certain attributes, and the pathname agrees with access permission before the actual "rename/move" transaction is requested.

Many read attempts with different maximum/minimum combination are definitely very peculiar for this implementation. We observed the same kind of behavior with a DOS client (under command prompt) and other clients in different versions of Windows.

The very last transaction corresponds to the verification of the file object name before the string is displayed on the Explorer's window. Note that this last transaction is understandably specific to the GUI requirements of visual change/update of file attribute and size for the object selected. Analyzing the log file resulted from running a simple "rename *oldname newname*" using DOS client can help us to explain some differences with respect to that of the Explorer counterpart. Nevertheless, multiple re-reads and re-opens are present on both log files in our study.

In addition to preparing for the server to log important details for this simple operation, we use the system call trace utility (strace) to record the system call activities. Two variations were used:

```
strace –f –c –o /tmp/rename.Ctrace –p 1238
strace –f –o /tmp/rename.trace –p 4131
```

The first form traces the process 1238, presumably the samba daemon process for the client in question, follows through fork() system calls, and saves the summary (-c option) to the log file named */tmp/rename.Ctrace*. The second form does no summary, but it provides a history of traced system calls in the log file.

Table 3: System Call Breakdown in terms of CPU Time

2.0.7 samba % time	NT4.0 seconds	Explorer usecs/call	rename calls	oldname errors	newname syscall
80.00	0.041734	1128	37		select
6.37	0.003323	185	18		getdents
4.37	0.002281	62	37		send
2.99	0.001558	19	81		read
1.62	0.000846	11	78		semop
1.59	0.000831	19	43	20	stat
0.79	0.000412	21	20	6	open
0.51	0.000267	6	44		gettimeofday
0.39	0.000201	201	1		rename
0.35	0.000185	11	17		getpid
0.35	0.000182	9	21		fstat
0.28	0.000148	11	14		close
0.18	0.000093	8	12		lseek
0.12	0.000064	7	9		fcntl
0.04	0.000022	22	1		lstat
0.03	0.000018	9	2		kill
100.00	0.052165		435	26	total

The system call trace breakdown is shown in Table 3 for the "rename" operation. Since this trace is gathered when the user is using the GUI to do file name change, most of the

time server spent is on network and socket related select() system call. It is worth noting that there are many getdents() calls and stat() calls which are results of going through the directory many times and checking the existence and non-existence of given file names.

System call traces without the summary mode logs each system call and thus provides a time-sequenced set of events. Among these 435 system calls, 43 of them are stat() calls. Figure 2 gives the flavor of some of these stat() calls with their arguments and return values. These naturally lead to the investigation of the needs of stat() system calls and the overall architecture of implementing a cache for the stat() system call results. That was a different but a related task in this study. It is illustrated further in the next Section.

```
                      strace output and grep "stats" (43 calls)
     2.0.7 samba and NT4.0, Explorer, rename oldname newname
4131 stat("oldname", {st_mode=S_IFREG|0664, st_size=8, ...}) = 0
4131 stat("oldname", {st_mode=S_IFREG|0664, st_size=8, ...}) = 0
4131 stat("oldname", {st_mode=S_IFREG|0664, st_size=8, ...}) = 0
4131 stat("oldname", {st_mode=S_IFREG|0664, st_size=8, ...}) = 0
4131 stat("oldname", {st_mode=S_IFREG|0664, st_size=8, ...}) = 0
4131 stat("oldname", {st_mode=S_IFREG|0664, st_size=8, ...}) = 0
4131 stat("oldname", {st_mode=S_IFREG|0664, st_size=8, ...}) = 0
4131 stat("oldname", {st_mode=S_IFREG|0664, st_size=8, ...}) = 0
4131 stat(".", {st_mode=S_IFDIR|S_ISVTX|0777, st_size=204800, ...}) = 0
4131 stat(".", {st_mode=S_IFDIR|S_ISVTX|0777, st_size=204800, ...}) = 0
4131 stat("oldname", {st_mode=S_IFREG|0664, st_size=8, ...}) = 0
4131 stat("./oldname", {st_mode=S_IFREG|0664, st_size=8, ...}) = 0
4131 stat("newname", 0xbfffe398)      = -1 ENOENT (No such file or directory)
4131 stat("newname", 0xbfffe398)      = -1 ENOENT (No such file or directory)
4131 stat("oldname", {st_mode=S_IFREG|0664, st_size=8, ...}) = 0
4131 stat("oldname", {st_mode=S_IFREG|0664, st_size=8, ...}) = 0
4131 stat("newname", 0xbfffe778)      = -1 ENOENT (No such file or directory)
4131 stat("newname", 0xbfffe778)      = -1 ENOENT (No such file or directory)
4131 ……………….
```

Figure 2: A Sample of stat() System Calls in "rename" Operation

3. Workload Characteristics of NetBench Runs

NetBench is a portable benchmark that measures the performance of file servers as they handle network file requests from clients [6]. Clients in this context are PCs running Windows 95/98, 2000, or Windows NT. The server environment is modeled as data provider for the applications such as word-processing or spreadsheet program that run on the clients. As a result, the server's disk I/O speed and network I/O speed are expected to be the areas that affect the benchmark score. NetBench provides an overall I/O throughput score and average response time for the server and individual scores for the clients.

Figure 3 shows the result of throughput measurement for a Linux Samba Server configured with 2-way SMP 800MHz Pentium III, 2GB memory, and 1Gbps Ethernet card. The Linux server in question runs a stock 2.4.0 kernel with ext2 file system and

Samba version 2.0.7. We use NetBench version 6.0 for this experiment. The client environment consists of 22 Windows NT machines each of which logically simulates up to 3 clients per machine for the experiment. Note that the unit of throughput is Mbits per second (Mbps). This is proportionally correlated to the number of transactions and work done by the clients. This metric represents the number of bits moved divided by the amount of time to do so, and it is an internal measure of work done and is not a direct measure of any throughput of disk or network bandwidth. This represents the total of all participating clients in the experiment. The server performance peaks out around 317 Mbps for this configuration. Without the SMPness, the throughput stops around 240 Mbps in the same configuration.

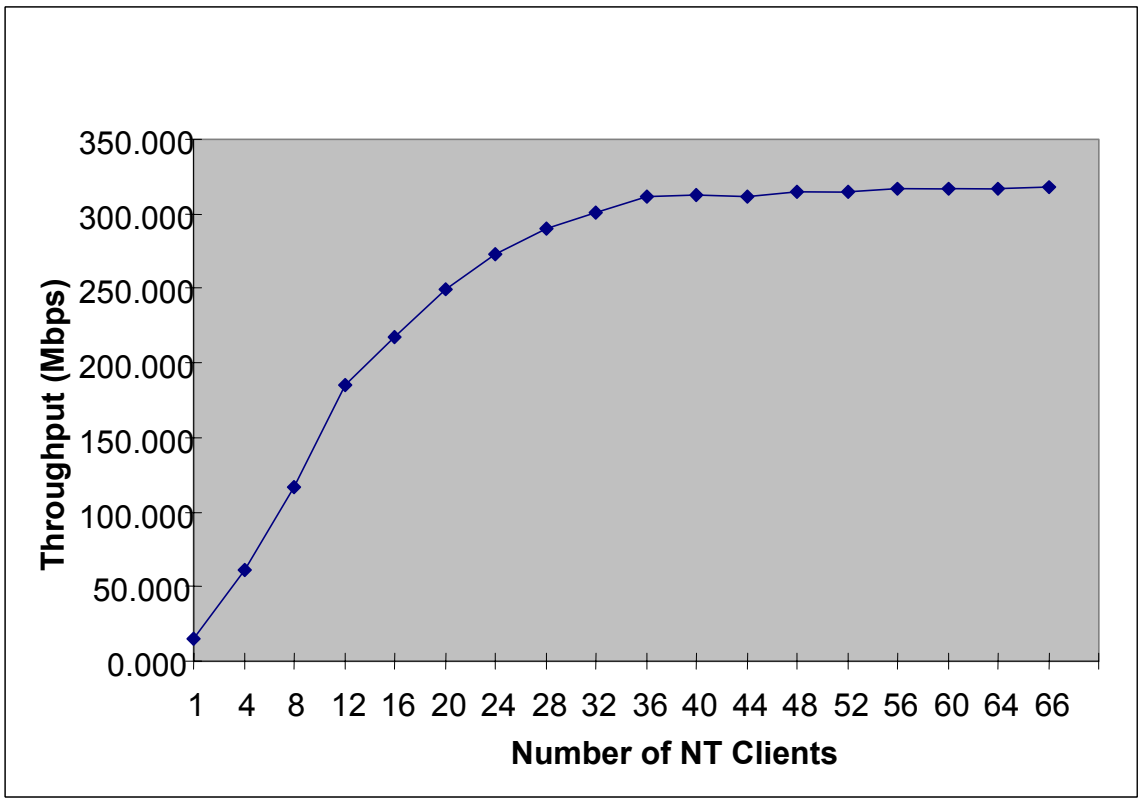

Figure 3: NetBench Throughput as a function of Number of Windows NT Clients

NetBench also tracks the response times of the majority of the I/O calls in milliseconds. Figure 4 depicts the average response time across all calls and all clients participated in the experiment. This metric is typically used as a sanity check to detect any undesirable server behavior while the throughput measure is used to compare one SMB/CIFS server against another.

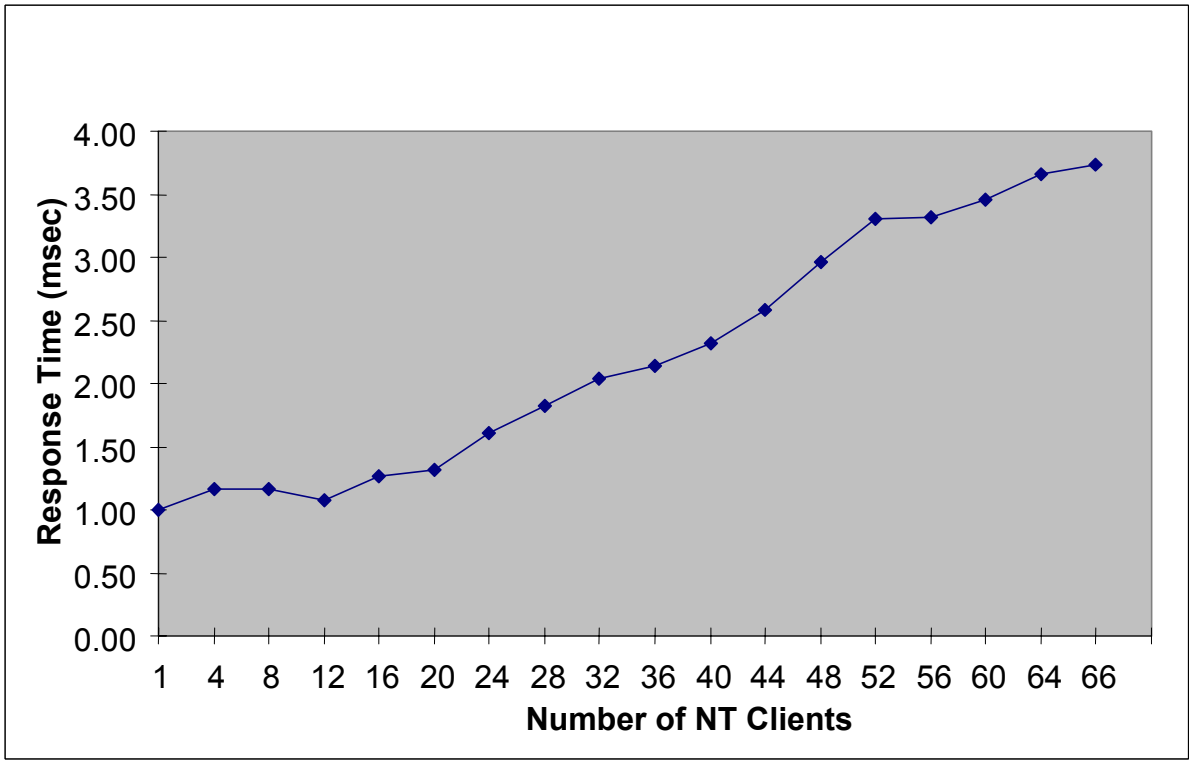

Figure 4: NetBench Average Response Time as a function of Number of Windows NT Clients

From a microscopic viewpoint, we are also interested in the detailed resource usage pattern and file system access behavior of NetBench runs. For this, we look into the details of one client run, record and examine its pathname access pattern.

In a typical 5-minute run for NetBench, each client generates about 80,000 pathname resolution requests, that is, number of calls to *unix_convert*() function, and issues 370,000 stat() system calls with 120,000 bad return code. This heavy usage of stat() calls and *unix_convert*() function leads to serious investigation on the current implementation of *unix_convert*() function. This function transforms the Windows pathname to canonical Unix pathname plus case sensitivity and name mangling processing and using stat() system call to verify the existence or non-existence of the translated pathname. For example, this involves the translation of pathname "\clients\client1\~dmtmp*" to "clients/client1/~dmtmp/*". Heuristics to cache partially matched prefixes of pathnames are used to speed up the translation and search. This part has been traced and analyzed for a more efficient implementation. Since the NetBench runs are basically CPU and network bound, and a sheer reduction in the number of stat() calls does not provide significant improvement in throughput measures. Nevertheless, one of the internal tools used to stress various aspects of samba server functionality, that is, the *smbtorture* program, was used to compare "trans2" transaction for the change of the stat cache implementation. This test happens to measure the efficiency of the alternate

implementation. The overall CPU time decreases by about 23% with significant reduction of getdents() and stat() calls in the new implementation.

We show the "before" and "after" system call distribution in Tables 4 and 5 to compare the new implementation using the insights of data analysis and review of access pattern. The number of stat() calls drop significantly with the new implementation, and the correct and more efficient handling stat() calls resulted in eliminating unnecessary re-reading of the directory content. The elimination of most of the getdents() calls is the reason for most of the 23% CPU time reduction.

Table 4: System Call Distribution BEFORE the Change

strace -c output (smbtorture trans2 test)

2.0.7 samba

getdents and stat calls are high

% time	seconds	usecs/call	calls	errors	syscall
34.96	4.343811	207	21002		send
17.01	2.114332	96	22004		getdents
13.30	1.653141	30	56012		semop
8.48	1.053835	26	41001	26001	stat
5.09	0.632101	15	42007		read
4.58	0.568644	27	21004		select
3.80	0.471903	118	4000		unlink
3.74	0.464828	46	10004	1	open
1.42	0.176387	176	1000		mkdir
1.35	0.167648	7	25003		gettimeofday
1.13	0.140352	9	16004		fstat
1.12	0.139495	139	1003		write
1.00	0.124036	12	10004		close
0.86	0.107101	107	1000		rmdir
0.79	0.098323	20	5000	1000	lstat
0.62	0.077106	6	12014		getpid
0.44	0.054617	9	6003		fcntl
0.18	0.022139	22	1000		chmod
0.13	0.015683	8	2001		lseek
0.01	0.000784	7	116		time
0.00	0.000075	38	2		chdir
0.00	0.000044	22	2		brk
0.00	0.000034	7	5		geteuid
0.00	0.000022	11	2		setresuid
0.00	0.00002	10	2		setresgid
0.00	0.000018	6	3		getegid
0.00	0.000007	7	1		setgroups
100.00	12.42649		297199	27002	total

Table 5: System Call Distribution AFTER the Change

strace -c output (smbtorture trans2 test)

2.0.7 samba with new unix_convert() implementation

getdents and stat calls drop significantly

% time	seconds	usecs/call	calls	errors	syscall
45.31	4.389434	209	21002		send
16.84	1.631563	29	56012		semop
8.76	0.848463	25	34003	15003	stat
6.52	0.631489	15	42007		read
5.71	0.553045	26	21004		select
3.82	0.370403	93	4000		unlink
3.41	0.330606	83	4006	1	open
1.86	0.180265	7	25003		gettimeofday
1.71	0.165212	165	1000		mkdir
1.30	0.126113	126	1003		write
0.99	0.096295	96	1000		rmdir
0.97	0.09366	19	5000	1000	lstat
0.91	0.088597	9	10005		fstat
0.80	0.077281	6	12018		getpid
0.65	0.062744	16	4006		close
0.23	0.02251	23	1000		chmod
0.18	0.017473	9	2001		lseek
0.01	0.000899	90	10		getdents
0.01	0.000748	6	120		time
0.00	0.00008	16	5		fcntl
0.00	0.000073	37	2		chdir
0.00	0.000048	24	2		brk
0.00	0.000031	6	5		geteuid
0.00	0.000028	28	1		ftruncate
0.00	0.00002	10	2		setresuid
0.00	0.000019	10	2		setresgid
0.00	0.000018	6	3		getegid
0.00	0.000007	7	1		setgroups
100.00	9.687124		244223	16004	total

Since NetBench is a script-based benchmark, the set of pathname used is somewhat a well characterized closed set of 260 unique names. Their skewed distribution is partially shown in Table 6 sorted by reference count. In this table, we can gain some insight on the pattern of the pathnames used.

Table 6: Part of Cumulative Distribution of Pathname References

For file log1a/log.dendro.UCCdout,
Total references = 54350, unique references = 267
(true hit 25952, 47%, and miss 28378)

index	cumulative(%)	raw count	pathname
1	11.74%	6378	\clients\client1\~dmtmp*
2	18.19%	3506	\clients\client1\~dmtmp\PARADOX
3	23.80%	3052	\clients\client1\~dmtmp
4	26.94%	1708	\clients\client1\~dmtmp\WORDPRO
5	29.97%	1646	\clients\client1\~dmtmp\PM
6	32.60%	1429	\clients\client1\~dmtmp\WORD
7	35.12%	1367	\clients\client1\~dmtmp\ACCESS
8	37.57%	1336	\clients\client1\~dmtmp\COREL
9	39.67%	1139	\clients\client1\~dmtmp\EXCEL
10	41.67%	1088	\clients\client1\~dmtmp\PWRPNT
11	43.21%	837	\clients\client1\~dmtmp\PWRPNT\NEWTIPS.PPT
12	44.75%	837	\clients\client1\~dmtmp\PWRPNT\NEWPCB.PPT
13	46.18%	775	\clients\client1\~dmtmp\COREL\@@@CDRW.TMP
14	47.32%	620	\clients\client1\~dmtmp\WORDPRO\LWPSAV0.TMP
15	48.45%	616	\clients\client1\~dmtmp\EXCEL\RESULTS.XLS

In Figure 5, we depict the cumulative distribution of most to leastly used pathnames. Because of this narrow set and limited file operations to this set, the footprint of NetBench runs is mostly captured in main memory's cache structures and leaves the NetBench runs CPU and network bound. In a reasonably configured Linux Samba server, the typical resource limitations are in network bandwidth, CPU speed and the efficiency of Samba server implementation and network protocol stack implementation.

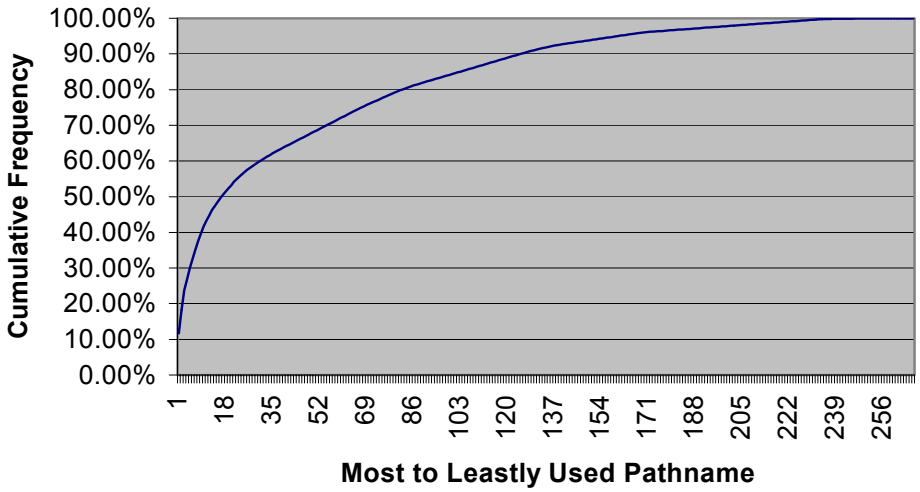

Figure 5: Cumulative Distribution of Pathname Frequencies in NetBench Runs

4. Conclusions

File servers are becoming an important part of enterprise computing. They serve files to a plethora of heterogeneous operating systems and hardware platforms. Linux-based SAMBA servers speak SMB/CIFS protocol allowing simultaneous file accesses to both Windows-based file systems and non-Windows-based, most notably, UNIX file systems, possible and convenient.

In this study, we have analyzed two workloads to gain insight on the message exchanges and resource usages between Samba client and server. One is that of a simple interactive "rename" operation; its trace allows us to see the typical message exchanges between a Linux Samba server and a Windows NT client. The large number of transactions required to do a simple "rename" calls for a careful and efficient implementation of client and file serving daemon. The other is that of the well-known NetBench benchmark which measures throughput and response time with a simulated set of scripts that use typical Win32 applications. The planning of such server requires attention for efficient network protocol implementation and bandwidth as well as raw CPU power with sufficient memory for file system caching. We have used Linux system call tracing facilities and Samba logging features to characterize the workloads in terms of file system pathnames, system calls, and traffic patterns. This technique and insight allows the proper planning and monitoring of Samba servers which are becoming more popular in modern-day corporate computing environments.

Acknowledgement

The author would like to express his gratitude to Ying Chen and Manoj Naik for their time and patience in discussing the data gathered for this work and their expertise in Samba implementation details. The author would also thank Nhan Tran for her help in the laborious setup of the lab environment for the experiment of NetBench runs.

References

[1] R. Eckstein, D. Collier-Brown and P. Kelly, *Using Samba*. O'Reilly & Associates, Sebastopol, CA, 2000.

[2] R. Sharpe, "Just What is SMB?" *http://samba.anu.edu.au/cifs/docs/what-is-smb.html*, September 1999.

[3] FTP site for SMB and CIFS documents: *ftp://ftp.microsoft.com/developr/drg/CIFS/*

[4] J. Allison, "Samba 2.0 Released," *http://linuxtoday.com/stories/2298.html*, January 1999.

[5] Linux Magazine, "The Story of Samba: Linux's Stealth Weapon," *http://www.linux-mag.com/1999-08/samba_02.html*, September 1999.

[6] "Understanding and Using NetBench 6.0*,"* *http://www.zdnet.com/*, Ziff Davis Inc., 1999.

[7] M. Welsh, M. Dalheimer and L. Kaufman, *Running Linux*. Third Edition, O'Reilly & Associates, Sebastopol, CA, 1999.