CITI Technical Report 93−10

AFS Server Logging

S. Blumson
P. Honeyman
T. E. Ragland
M. T. Stolarchuk

`info@citi.umich.edu`

ABSTRACT

The AFS servers at the Center for Information Technology Integration have been modified to trace and log file server activity. This report discusses the AFS modifications and the structure of the trace files and data. We also describe three large datasets collected from the logging servers, available to other researchers.

November 30, 1993

Center for Information Technology Integration
University of Michigan
519 West William Street
Ann Arbor, MI 48103-4943

AFS Server Logging

S. Blumson
P. Honeyman
T. E. Ragland
M. T. Stolarchuk

`info@citi.umich.edu`

1. Introduction

This report describes our modifications to AFS for generating trace records describing file server activity. The goal of tracing is to record much salient information about each client request presented to AFS servers over an extended period of time. We use these logs for a number of purposes:

- Evaluating pricing models and strategies for deploying AFS to the campus.
- Planning and evaluating needs for AFS server and TCP/IP capacity.
- Developing analytic models of network activity in a distributed filing environment.
- Simulating disconnected operation in mobile AFS clients.
- Developing a set of profiles describing the access patterns of various categories of users.
- Simulating the performance of intermediate servers.
- Diagnosing problems.

The trace records are generated on AFS servers; AFS client activity that is served from the local cache manager does not show up in the logs. Each RPC presented to the server produces a single trace record.

The data are collected from the University of Michigan Institutional File System (IFS). The IFS, which consists of AFS servers and clients running on various platforms scattered across the U-M campus, has been evolving over the past several years with the goal of providing an integrated, location-independent file system for the entire University community.

2. Building a logging server

A logging AFS server is built from `rxgen`, the Rx stub compiler, specially modified at CITI. (Rx is the remote procedure call service layer on which AFS is built.) Our modification causes the server stubs for AFS to bracket each service call with calls to a prologue function and an epilogue function. For example, the modified stub for the `fetchdata` call contains:

```
LOG_SRXAFS_FetchData(SRXAFS_FetchData, &Fid, Pos, Length,
                                &OutStatus, &CallBack, &Sync);
z_result = SRXAFS_FetchData(z_call, &Fid, Pos, Length,
                                &OutStatus, &CallBack, &Sync);
LOG_EPILOG(SRXAFS_FetchData, "SRXAFS_FetchData");
```

(The first and third lines are the ones we added; the second line is the normal output of `rxgen`.)

The prologue and epilogue functions take a ‘‘snapshot’’ of the state of the system at the time they are called. The epilogue function then tallies the resources used to process the request and dumps a trace record to the log file. Additional code saves the resource counters around server thread context switches so that, to the extent possible, resources are in fact charged to the correct RPC.

The log file is usually placed in the working directory of the file server, generally `/usr/afs/logs/AFSlog.XXXXXX`. (The `mktemp(3)` library function is used to create the name of the log file.)

The file `/afs/citi.umich.edu/usr/afs/src/rx/RXAFSLOG_README` describes how to build a logging AFS server. We also describe the steps in the Appendix.

3. Data fields

Here is a sample trace record in text form:

```
fetchdata (0) @ 688145060.320041 client 141.211.128.207
user 0.020000 sys 0.040000 elapsed 0.150000 in 2 out 1
yield 0 syscall 54 retrans 0 user honey fid 200000B3:3EB0:112E0
pos 0 len 5711
```

The fields in this record are interpreted as follows:

`fetchdata (0)`	ASCII and numeric representation of request type
`@ 688145060.320041`	Start time (seconds since January 1, 1970)
`client 141.211.128.207`	IP address of client making the request
`user 0.020000`	CPU time spent while in user mode
`sys 0.040000`	CPU time spent while in kernel mode
`elapsed 0.150000`	Wall clock time spent processing the request
`in 2`	Number of disk reads
`out 1`	Number of disk writes
`yield 0`	Number of thread yields
`syscall 54`	Number of system calls
`retrans 0`	Number of Rx retransmissions
`user honey`	Kerberos identity of client
. . .	

The remaining fields depend on the particular AFS request; this is explained further in the section on optional fields.

Log data are recorded in variable length records with fields shown in the following table.

Field	Length	Flag	Field	Length	Flag	Field	Length	Flag
MASK	short	M	ETIME	long	C	FID1	long×3	O
CMD	short	C	IOIN	short	C	FID2	long×3	O
START	long	C	IOOUT	short	C	NUM1	long	O
µSTART	long	M	YIELD	short	C	NUM2	long	O
ADDR	long	C	SYSCALL	short	M	NUM3	long	O
UTIME	long	C	RESEND	short	C	STR1	string	O
STIME	long	C	USER	string	M	STR2	string	O

Using principles similar to TCP header compression, some fields are elided from the output if their values can be discerned from earlier records. Fields marked ‘‘`M`’’ are mandatory; *i.e.*, they appear in every output record. Fields marked ‘‘`C`’’ are also mandatory, but may be compressed out of output stream by means described in the next section. Fields marked ‘‘`O`’’ are optional, depending on the particular AFS command being processed.

4. Mandatory and compressible fields

In this section, we describe the mandatory and compressible fields. Compressible fields are those that can be elided from the raw output in certain circumstances, described next. We typically post-process this raw output into the form shown in the sample trace record above, with compressible fields restored to their values, and then re-compress the resulting text stream with the UNIX `compress` command. The post-processor fills in any elided fields.

4.1. MASK

The MASK field is a 16-bit word that shows which of the compressible and optional fields are present. The START, UTIME, STIME, IOIN, IOOUT, YIELD, and RESEND fields are elided from the raw output if they are equal to zero. The CMD and ADDR fields are elided if the value of the field is identical to that in the previous record.

4.2. CMD

This field gives the numeric AFS command as shown in the following table:

Id	Name	Id	Name	Id	Name
0	`fetchdata`	11	`makedir`	22	`oldsetlock`
1	`fetchacl`	12	`removedir`	23	`oldextendlock`
2	`fetchstatus`	13	`setlock`	24	`oldreleaselock`
3	`storedata`	14	`extendlock`	25	`getstatistics`
4	`storeacl`	15	`releaselock`	26	`giveupcallbacks`
5	`storestatus`	16	`getvolumestatus`	27	`getvolumeinfo`
6	`removefile`	17	`setvolumestatus`	28	`bulkstatus`
7	`createfile`	18	`getrootvolume`	29	`xstatsversion`
8	`rename`	19	`checktoken`	30	`getxstats`
9	`symlink`	20	`gettime`		
10	`link`	21	`ngetvolumeinfo`		

4.3. START and µSTART

The START field is a long integer giving the elapsed time in seconds since the previous AFS request. For all reasonable values, the elapsed time seconds are multiplied by one million and added to the µSTART field, in which case START is elided.

The µSTART field is a long integer giving the elapsed time in microseconds and is always present. µSTART may be larger than one million. If the START field is present, START and µSTART are combined to give the total elapsed time. The resulting value is then added to the starting time of the previous AFS request to give the actual starting time.

4.4. ADDR

The ADDR field gives the IP address of the client that made the AFS request. If present, the ADDR field is in network order. If omitted, the client address is the same as that of the previous AFS request.

4.5. UTIME and STIME

For file servers running on the UNIX operating system, the UTIME field shows the amount of CPU time spent servicing the request while in user mode; the STIME field gives the kernel mode CPU time. On MVS systems the UTIME and STIME fields show CPU times in the IFS/Rx and TCP/IP address spaces, respectively.[†] Both fields are four-byte integers in units of microseconds. The UTIME and STIME fields are elided if equal to zero.

4.6. ETIME

The ETIME field shows the total elapsed time from the start of the request to its completion. It is a four-byte integer, in microsecond units, and is always present.

[†] The MVS service times are accurate only if the address spaces are single threaded, and if there is no other usage of TCP/IP on the machines. Both conditions were true at this writing.

4.7. IOIN and IOOUT

On the UNIX operating system, the IOIN and IOOUT fields show the number of disk read and write operations performed on behalf of the request. On MVS systems, they show the number of Start IOs in the IFS/Rx address space and the TCP/IP address space, respectively, subject to the same accuracy conditions as the UTIME and STIME fields. IOIN and IOOUT are 16-bit integers, elided if zero.

4.8. YIELD

The file server is multi-threaded. The YIELD field is a 16-bit integer that shows the number of times a thread servicing this request yielded control to the lightweight process scheduler. If zero, this field is elided.

4.9. SYSCALL

The SYSCALL field is not meaningful on UNIX and MVS systems.

4.10. RESEND

The RESEND field is intended to show how many packets were retransmitted by the RPC communications layer while the request was being serviced. We are highly skeptical of its accuracy. It is a 16-bit field, elided if zero.

4.11. USER

The USER field shows the Kerberos identity of the user that led to the service request. Many requests are unauthenticated, *e.g.*, requests issued by a background daemon. In these cases the USER field shows a question mark.

5. Optional fields

The remaining optional fields are controlled by the MASK field. These fields are `FID1`, `FID2`, `NUM1`, `NUM2`, `NUM3`, `STR1`, and `STR2`. The interpretation of these fields varies from command to command.

A FID is a File IDentifier, a fundamental AFS data structure. It consists of three long integers: volume number, vnode number, and vnode version number. If the vnode number is even, a FID corresponds to a regular file; otherwise it represents a directory. The NUM fields are long integers, used for integer valued output. The STR fields are used for character string output. The STR fields are null-terminated in the raw output file.

The FID, NUM, and STR fields are present only if they are meaningful for a particular command. The interpretation of these fields varies depending on the AFS command.

The following table shows the optional fields used by AFS commands; commands not shown use no optional fields.

Command	FID1	FID2	NUM1	NUM2	NUM3	STR1	STR2
fetchdata	×		×	×			
fetchacl	×						
fetchstatus	×						
storedata	×		×	×	×		
storeacl	×						
storestatus	×						
removefile	×					×	
createfile	×	×				×	
rename	×	×				×	×
symlink	×	×				×	×
link	×	×				×	
makedir	×	×				×	
removedir	×					×	
setlock	×		×				
extendlock	×						
releaselock	×						
getvolumestatus			×				
setvolumestatus			×				

The remainder of this section describes the interpretation of these fields for the AFS commands that use them.

5.1. fetchdata

FID1 is the FID of the file or directory being read. NUM1 is the offset in the file. NUM2 is the size of the request.

5.2. fetchacl

FID1 is the FID of the file.

5.3. fetchstatus

FID1 is the FID of the file.

5.4. storedata

FID1 is the FID of the file being stored. NUM1 is the offset in the file. NUM2 is the size of the request. NUM3 is the new file size.

5.5. storeacl

FID1 is the FID of the file.

5.6. storestatus

FID1 is the FID of the file.

5.7. removefile

FID1 is the FID of the parent directory. STR1 is the name of the file being removed.

5.8. `createfile`

`FID1` is the FID of the directory in which the file is being created. `STR1` is the name of the new file. `FID2`, the FID for the new file, is a result parameter returned to the client.

5.9. `rename`

`FID1` and `STR1` are the old directory and name. `FID2` and `STR2` are the new directory and name.

5.10. `symlink`

`FID1` and `STR1` identify the directory and name of the new link. `STR2` is the contents of the symbolic link. `FID2`, the FID for the new symbolic link, is a result parameter returned to the client.

5.11. `link`

`FID1` and `STR1` identify the directory and name of the new link. `FID2` is the FID of the object being linked.

5.12. `makedir`

`FID1` and `STR1` are the parent directory and name of the new directory. `FID2`, the FID for the new directory, is a result parameter returned to the client.

5.13. `removedir`

`FID1` is the FID for the parent directory. `STR1` is the name of the directory being removed.

5.14. `setlock`

`FID1` is the FID of the object being locked. `NUM1` is the type of lock.

5.15. `extendlock`

`FID1` is the FID of the object being relocked.

5.16. `releaselock`

`FID1` is the FID of the object being unlocked.

5.17. `getvolumestatus`

`NUM1` is the volume ID.

5.18. `setvolumestatus`

`NUM1` is the volume ID.

6. CITI AFS datasets

In October, 1990 and again in April, 1992, we enabled AFS server logging in all of the servers under our control; in April 1993 this was repeated on a subset of the servers constituting the fast majority of the workload. This provides us with three extensive datasets, which we are using for our own purposes, and which we are making available to other researchers. To obtain a copy of the CITI AFS datasets, contact: `info@citi.umich.edu`.

In an attempt to simplify life for everyone, for the first two datasets we converted the compact output files generated by the servers into a printable text format, filling in elided fields and accumulating start times into actual values. These files were then compressed using the UNIX `compress` command, substantially shrinking their size. The larger volume of data made this unworkable for the 1993 datasets; instead a subroutine library is provided for expanding the data on the fly.

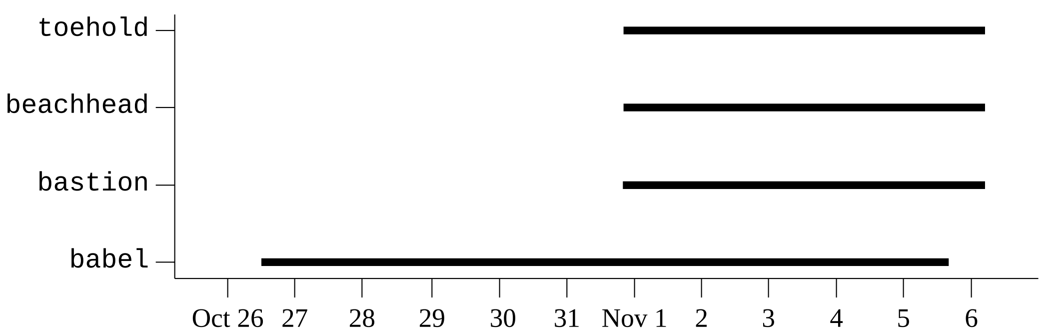
6.1. October, 1990 dataset

These logs, generated in the early days of CITI's Institutional File System project, have a slightly different format: the `yield`, `syscall`, `retrans`, and Kerberos `user` fields are missing in these logs. In all other respects, the format is identical to the earlier description.

Here is a sample trace record from the October, 1990 dataset:

```
fetchdata (0) @ 656956630.043153 client 141.211.168.42
user 0.040000 sys 0.320000 elapsed 0.573903 in: 10 out: 1
fid 20000399:1A:E pos 0 len 65536
```

Logs were collected on all of the servers in the `ifs.umich.edu` cell in late October and early November, 1990. All servers were IBM RT computers with IBM 9331 SCSI disks. The following chart shows the periods during which server logs were collected.



The logs overlap for 4.8 days, from 8:13:16 P.M. on Wednesday, October 31, 1990 to 5:00:00 A.M. on Monday, November 5, 1990.

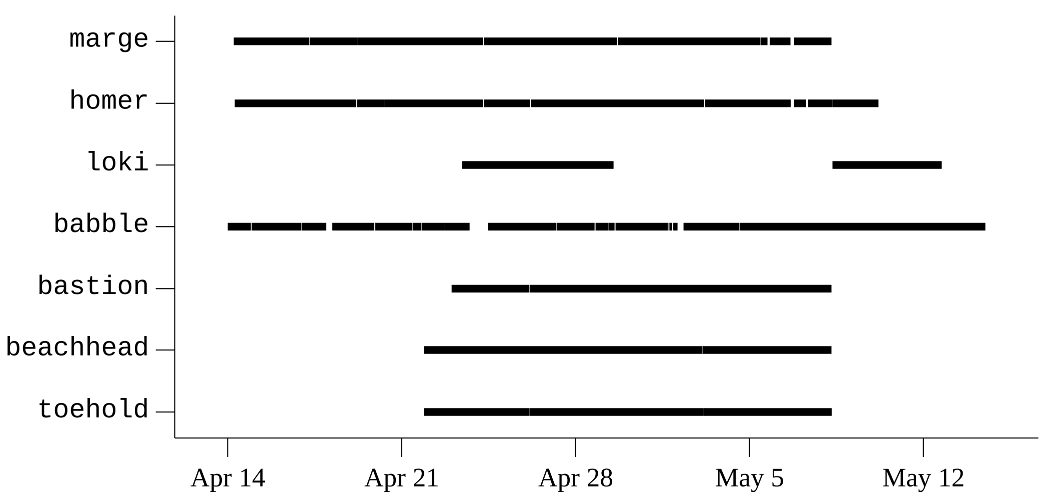
‡ We show a `fetchdir` command, which is actually a `fetchdata` for a FID that happens to be a directory.

The following table shows some of the gross characteristics of the logs.‡

	bastion	beachhead	toehold	babel	TOTAL
fetchstatus	27,222	120,126	189,369	120,831	457,548
gettime	15,216	9,947	23,645	10,717	59,525
storedata	6,746	34,474	6,765	7,219	55,204
fetchdata	3,764	10,171	10,973	20,869	45,777
getvolumeinfo	11,110	2,611	14,054	9,626	37,401
storestatus	3,470	15,747	4,612	3,769	27,598
getvolumestatus	32	15,521	6,148	11	21,712
createfile	2,566	6,685	3,470	3,940	16,661
giveupcallbacks	1,178	5,584	1,896	6,708	15,366
fetchdir	2,528	5,059	5,052	2,587	15,226
removefile	1,735	7,543	3,401	2,508	15,187
rename	1,423	2,426	625	1,372	5,846
link	40	1,171	1,135	28	2,374
makedir	86	998	413	124	1,621
setlock	40	587	935	0	1,562
symlink	9	475	177	28	689
releaselock	40	407	203	0	650
fetchacl	5	154	105	26	290
storeacl	0	113	29	21	163
oldsetlock	0	2	115	0	117
extendlock	4	45	12	0	61
removedir	10	15	22	2	49
oldreleaselock	0	2	45	0	47
getstatistics	0	0	0	29	29
bulkstatus	0	9	0	0	9
oldextendlock	0	0	7	0	7
setvolumestatus	1	0	2	1	4
TOTAL	77,225	239,872	273,210	190,416	780,723

6.2. April, 1992 dataset

Our intent was to collect data for an uninterrupted period from April 23, 1992 to May 8, 1992 on all AFS servers in the `umich.edu` and `citi.umich.edu` cells. However, system problems terminated data collection prematurely on `loki`. The following chart shows the periods from mid-April to mid-May 1992 during which server logs were collected in the cells.



The gaps in the `babble`, `marge`, and `homer` traces reflect server down time. The longest interval during which logs were collected on all the servers lasts for 6.1 days, from 10:23:54 A.M. on Thursday, April 23, 1992 to 12:57:48 P.M. on Wednesday April 29, 1992. The file server characteristics are outlined in the following table.

name	CPU	OS	disks
marge	IBM ES/9000 Model 720	AIX/370 V1.1	IBM 3380
homer			
loki	IBM ES/9000 Model 580	MVS/ESA V4.2	IBM 3380, 3390
babble	IBM RT Model 125	AOS V4.3 BSD	IBM 9332 SCSI
bastion	IBM RS/6000 Model 320H	AIX V3.1	IBM 400MB SCSI
beachhead			
toehold			

marge and homer share a single processor with several other VM guests. The file servers have never been observed to consume more than a fraction of a processor, due to network and other I/O constraints. loki operates in a similar environment, on different hardware.

The following tables shows the number of AFS requests, broken down by server.

	marge	homer	loki	babble
fetchstatus	7,979,974	3,262,562	2,967,709	1,511,093
gettime	349,387	428,775	161,202	45,675
getvolumestatus	1,098,282	197,699	58,853	326
fetchdata	523,069	232,587	329,917	54,709
fetchdir	421,645	230,659	151,461	26,316
giveupcallbacks	420,834	156,203	162,280	38,263
storedata	174,270	346,596	90,353	39,359
storestatus	269,813	253,364	23,015	34,540
extendlock	69,193	3,031	98,150	242,585
createfile	112,169	165,445	46,712	24,182
getvolumeinfo	2	2	0	82,455
getstatistics	66,018	69,818	49,217	111
removefile	52,900	118,106	35,590	14,555
rename	27,177	64,285	6,860	10,362
setlock	65,752	2,051	470	418
releaselock	34,889	1,648	427	283
makedir	7,509	10,097	2,771	948
link	3,764	9,566	407	923
symlink	6,584	5,765	1,128	196
removedir	1,223	4,982	1,596	254
fetchacl	1,338	1,106	360	128
storeacl	751	504	56	28
setvolumestatus	226	218	14	3
getxstats	27	67	0	0
TOTAL	11,686,796	5,565,136	4,188,548	2,127,712

	bastion	beachhead	toehold	TOTAL
fetchstatus	162,132	489,775	1,498,517	17,871,762
gettime	154,077	130,943	150,879	1,420,938
getvolumestatus	236	107	321	1,355,824
fetchdata	5,718	3,641	7,657	1,157,298
fetchdir	6,492	4,865	6,117	847,555
giveupcallbacks	4	1	0	777,585
storedata	2	0	0	650,580
storestatus	0	0	0	580,732
extendlock	0	0	0	412,959
createfile	3	148	36	348,695
getvolumeinfo	74,734	101,181	77,456	335,830
getstatistics	37,650	37,656	37,651	298,121
removefile	2	1	0	221,154
rename	0	0	0	108,684
setlock	0	0	0	68,691
releaselock	0	0	0	37,247
makedir	198	79	193	21,795
link	0	0	0	14,660
symlink	1	0	0	13,674
removedir	0	0	0	8,055
fetchacl	9	10	4	2,955
storeacl	3	1	1	1,344
setvolumestatus	2	2	2	467
getxstats	0	0	0	94
TOTAL	441,263	768,410	1,778,834	26,556,699

6.3. April, 1993 dataset

Our intent was to collect data for the entire month of April 1993 on the mainframe-based servers in the `umich.edu` cell. Various technical hassles were making it difficult for us to build up-to-date instrumented servers for all of our platforms, and the 1992 data convinced us that moving ahead on this subset of servers would capture the vast majority of campus usage.

Data was actually collected from April 8 to April 28. However administrative problems caused data prior to April 21 to be lost on `loki`.

The file server characteristics are outlined in the following table.

name	CPU	OS	disks
blaze	IBM ES/9000 Model 720	AIX/ESA V1.2	IBM 3380
fang			
larch			
spam			
loki	IBM ES/9000 Model 580	MVS/ESA V4.2	IBM 3380, 3390

`blaze`, `fang`, `larch` and `spam` share a single processor with several other VM guests. The file servers have never been observed to consume more than a fraction of a processor, due to network and other I/O constraints. `loki` operates in a similar environment, on different hardware.

The following tables show the number of AFS requests, broken down by server.

	blaze	fang	larch
fetchdata	2,555,478	65,832	2,355,940
fetchacl	874	80	12,294
fetchstatus	8,351,486	131,295	6,751,110
storedata	310,614	40,389	291,017
storeacl	1,353	66	2,731
storestatus	613,291	21,752	223,916
removefile	118,372	10,039	49,758
createfile	291,584	7,155	59,669
rename	68,386	2,335	26,744
symlink	15,380	241	1,487
link	13,863	111	7,823
makedir	6,963	401	3,559
removedir	2,874	734	1,257
setlock	38,263	114	35,502
extendlock	1,538	3	4,226
releaselock	37,540	114	35,364
getvolumestatus	1,020,953	9,947	65,361
setvolumestatus	139	35	138
getrootvolume	0	0	0
checktoken	0	0	0
gettime	1,771,387	1,610,498	2,672,206
ngetvolumeinfo	0	0	0
oldsetlock	0	0	0
oldextendlock	0	0	0
oldreleaselock	0	0	0
getstatistics	25,707	30,955	27,354
giveupcallbacks	261,840	8,145	173,455
getvolumeinfo	0	0	0
bulkstatus	2,852	1,127	2,736
xstatsversion	0	0	0
getxstats	0	0	0
fetch directory	1,301,320	33,885	1,796,039
TOTAL	16,812,057	1,975,253	14,599,686

	loki	spam	TOTAL
fetchdata	289,511	1,587,597	6,854,358
fetchacl	145	4,374	17,767
fetchstatus	1,495,581	5,088,771	21,818,243
storedata	57,165	228,683	927,868
storeacl	57	2,628	6,835
storestatus	17,146	220,723	1,096,828
removefile	11,767	42,639	232,575
createfile	50,525	55,240	464,173
rename	4,286	33,632	135,383
symlink	412	2,176	19,696
link	2,020	6,809	30,626
makedir	942	3,289	15,154
removedir	559	1,425	6,849
setlock	1,689	13,312	88,880
extendlock	1,852	1,600	9,219
releaselock	1,689	13,298	88,005
getvolumestatus	20,901	93,614	1,210,776
setvolumestatus	23	144	479
getrootvolume	0	0	0
checktoken	0	0	0
gettime	1,188,209	1,873,234	9,115,534
ngetvolumeinfo	0	0	0
oldsetlock	0	0	0
oldextendlock	0	0	0
oldreleaselock	0	0	0
getstatistics	36,349	27,126	147,491
giveupcallbacks	89,975	172,189	705,604
getvolumeinfo	0	0	0
bulkstatus	353	394	7,462
xstatsversion	0	0	0
getxstats	0	0	0
fetch directory	93,868	889,898	4,115,010
TOTAL	3,365,024	10,362,795	47,114,815

7. Caveats

Potential users of this data should be conscious of some quirks. The most important is that the collection was done on our servers and so reflects actual user activity only as filtered by the (large) caches on our AFS clients. While this is quite useful for studies of AFS performance, it introduces some significant (and not well understood) biases that might seriously hamper investigators interested in actual user behavior.

Each of the datasets has particular problems relating to the circumstances of its collection. The first (1990) dataset is both old and from a comparatively small and homogeneous user population. The 1992 data is both larger and more diverse, but difficulties in collection on `loki` limit the period for which complete data is available. In addition, the code that assigns resource utilization data to the correct call was broken during this session, limiting the usefulness of those figures.

The 1993 data had similar problems on `loki`. In addition, loss of the beginning of the `loki` data caused us to lose the exact starting time. Because of this, and because the compression algorithm provides only differential time of day for subsequent records, the exact time of each call is uncertain, although we believe we have recovered the starting time within two hours. A similar problem occurred on `blaze`, although there the uncertainty is only a few minutes.

Acknowledgements

Kevin Coffman and Ron Zeilinger of the IFS/MVS staff and Mark Giuffrida and Bob King of the IFS deployment group ported the logging code to the MVS, AIX/370, and RISC System/6000 servers. Charles Antonelli helped debug the AIX/370 server. Bill Doster, manager of the `umich.edu` cell, facilitated access to its servers. Dan Muntz helped with data organization and analysis.

This work was partially supported by IBM.

Appendix

This section describes how to build a logging file server. We assume objects are built in `/usr/afs/obj` and the sources are ‘‘current.’’

These files are new:

```
rx/rx_afslog.h
rx/rx_afslog.c
```

These files have been changed:

```
rxgen/rpc_parse.c
fsint/afsint.xg
rx/rx.c
rx/rx.h
```

Rebuild `rxgen` to understand the `-L` flag:

```
cd /usr/afs/obj/rxgen
make install
```

Create a server stub with logging code:

```
rm afsint.ss.c
make afsint.ss.c RXFLAG=-L
make ’CC=cc -DLOGRXAFS’ install
```

Build the Rx library:

```
cd /usr/afs/obj/rx
rm rx.o rx_afslog.o
make ’CC=cc -DLOGRXAFS’ \
    ’XLIBS=../../dest/lib/librxkad.a ../../dest/lib/libdes.a librx.a’ \
    install
```

Build a logging fileserver:

```
cd /usr/afs/obj/viced
make install
```

Logging can be disabled with `adb`:

```
echo LogRxEnable?W0  | adb -w fileserver
```