CITI Technical Report 92–6

Workload Characterization of AFS File Servers

Rajalakshmi Subramanian
rajalak@citi.umich.edu

ABSTRACT

This paper describes the workload characterization of AFS file servers, based on traces collected by the file servers at CITI over a 2-4 week period. These workload characteristics have been used to compare the performance of servers running on different hardware. There are two parts to this paper. In the first part, we describe a *server model* that will be used to drive a synthetic workload. In the second part, we build a *client model* consisting of the requests made by different user types. We show that the user community can be broken into distinct types, where all the users of a specific type exhibit similar request patterns. This *clustering* information is used in conjunction with the workload characteristics at the server to predict loads on servers, given the community that needs to be served. Finally, we give an example to show how this can be done.

November 23, 1992

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

Workload Characterization of AFS File Servers

Rajalakshmi Subramanian

November 23, 1992

1. Introduction

Performance evaluation studies of computer systems frequently depend on the presence of a real or synthetic workload under which the performance indices are measured. For the derived results to be accurate, the synthetic workload must be representative of the real workload that will run on the system. The aim of this study is to describe the workload characteristics of the AFS servers at Center for Information Technology Integration (CITI). The approach we took was to use traces collected continuously at the file servers over a period of 2-4 weeks. The traces contain a record for every single call made to the file server. Any variations in referencing patterns, due to the time of the day or the day of the week (such as weekends etc.), can be extracted from the traces.

One advantage of extracting information from traces is that it represents an accurate workload for the system. It also preserves the correlation effects in the workload. Another advantage of using traces is that the specific load they are used to generate can be reproduced in future experiments. The parameters in a synthetic workload that are based on information extracted from the traces can be easily changed to test scalability or other performance indices. The disadvantage is that the traces obtained from one environment may not be representative of the behavior patterns of another.

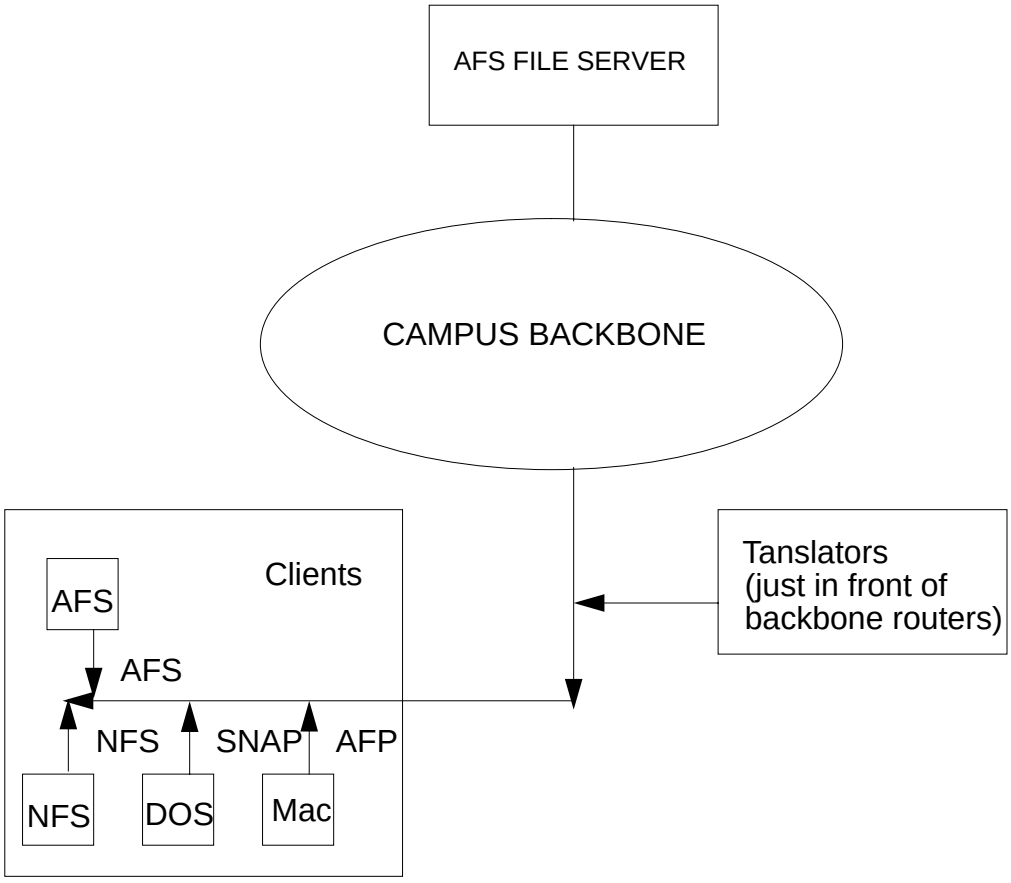
There are two parts to this paper. In the first part, the workload characteristics have been extracted from traces collected at the file server end, and these have then been generalized. Using these characteristics we have built a model of server usage. We intend to use this model to build a synthetic workload to drive servers and obtain performance data. It can also be used to drive a simulation. In the second part, we have built a model of the client behavior based on the user type. This information is useful in predicting the load on the server given a user community.

The workload has been characterized on five different servers. The servers under consideration are named **homer**, **marge**, **loki**, **bastion**, and **babble**. Of these, **homer** and **marge** run under AIX/370, **loki** runs under MVS, **bastion** is an RS/6000 running AIX 3.0, and **babble** is an IBM RT running 4.3BSD UNIX. While characterizing the workload, we were also able to collect information on file server attributes. We used the server characteristics to compare the relative performances of servers running on different hardware and to help identify bottlenecks and anomalous behavior.

We have also looked at the server usage on a per *user* basis. The traces generated by each user at the servers have been used to *cluster* the workload based on the user type. The current set of user types includes software developers, students, technical writers, and system administrators. The re-

sults from this clustering of behavior patterns can be used to predict the load on a server, given a mix of user types. We used this information to build a model of the workload generated by each user type. This approach should help answer questions, such as the number of AFS servers required to support a university department that has 10 secretaries, 40 faculty members, and 500 students. The traces have also been analyzed on a per *client* basis to see the distribution of load generated by the different clients. (A *client* is defined as a workstation or machine identified by its IP address. Multiple *users* may generate requests from a single *client*.)

Figure 1. Client/Server Distribution in our campus



AFS is different from other distributed file systems in that it was designed to support a very large set of users [5]. In order to provide this scalability and maintain acceptable performance levels, it requires the client machines to maintain very large caches. A Cache Manager maintains the cache, and decides whether a request can be satisfied locally or must be forwarded to the server (using an RPC). This results in a somewhat different distribution of requests at the server than if the user were directly making calls to the server. For example, the cache manager will usually be able to satisfy a read request (fetchdata) for a frequently used file from the local cache, but will make a relatively larger number of requests to the server for getting attributes of files (fetchstatus). The larger number of file status requests occurs as a side effect of the attribute cache being fairly small. The Cache Manager also makes fetchstatus calls when it needs to ensure that the data in the cache is valid before it returns it to the user. Figure 1 shows how the clients and servers are connected at CITI. (Note that the servers and clients are in completely different physical locations.)

Section 2 provides information on what is contained in the traces. Section 3 describes the factors affecting performance. Section 4 details the server characteristics, as well as the analysis of data,

and the model based on the analysis. Section 5 illustrates the clustering of request types based on the user type, while Section 6 describes how this property can be used for server prediction. Section 7 presents conclusions.

2. Information Contained in the Traces

All the information was recorded at the server end only. The file server can handle 30 different types of service calls. These arrive from individual client workstations in the form of RPCs. To develop the model, we have, in some cases, further subdivided the types of requests into calls relevant to directories, calls relevant to files, and general calls such as gettime and getvolumeinfo.

The information in the traces include: RPC type, start time, response time, IP address of the client issuing the call, CPU usage information, user id (in some cases), number of disk reads / writes, and file access information such as position and length of request. Refer to [8] to obtain details on what information is logged for each RPC type. The logging was done continuously to allow us to observe the variation in request and response patterns at the server.

3. Workload Characteristics

The main criterion of server performance is response time. This can depend on a number of factors such as the number of disk I / Os, the number of active clients, the interarrival times of the messages, and server cache hits. Other factors that can affect the performance are the frequency distribution of requests, the size distribution of read / write requests, and the number of requests being processed by the file server simultaneously. In this study, we ignore the effects of the load on the system under which the file server is running.

To characterize the workload at the server, the following values were computed:

1. Mean interarrival times and mean response times.
2. Response time as a function of the request type.
3. Frequency distribution of requests at the server and the mean response time for each request type.
4. Analysis based on the fact that the file server can be modeled by an M/M/1 queue or a G/M/1 queue.

The following results were computed to understand the behavior patterns of users and client machines

1. Percentage of requests related to files (as opposed to directories) per user type.
2. Read/write size distribution per user type.

Table 1: Interarrival and response times in seconds on **Homer**, **Marge**, and **Loki**, grouped by component exponential distributions.

	Loki Interarrival Time	Loki Response Time	Homer Interarrival Time	Homer Response Time	Marge Interarrival Time	Marge Response Time
Total number of calls	2069110	2069110	1917755	1917755	2211434	2211434
Median	0.064809	0.002188	0.114078	0.000736	0.026644	0.000686
Mean	0.078961	0. 002003	0.118443	0.000550	0.027249	0.000562
Std Deviation	0.075067	0.000425	0.109847	0.000212	0.025156	0.000260
Coeff of Variation	0.9506	0.2089	0.9274	0.3859	0.9232	0.4624
% of calls	92.06	78.79	51.8	57.41	85.82	86.45
Mean		0.00645	0.131517	0.154703	0.384603	0.098743
Std Deviation		0.006063	0.127828	0.130568	0.356991	0.096793
Coeff of Variation		0.9399	0.9719	0.8439	0.9257	0.9802
% of calls		18.16	39.01	41.76	4.05	13.42
Mean		0.099611				
Std Deviation		0.087624				
Coeff of Variation		0.8796				
% of calls		2.84				
% Discarded	7.9	0.2	9.7	0	9.7	2.9
Dates collected	04/23-04/29	04/23-04/29	04/26-05/03	04/26-05/03	04/26-05/03	04/26-05/03

4. Model Development

Table 1 shows the mean interarrival and response times (in seconds), median, variance, and coefficient of variation for the different servers. All the values were obtained from traces collected for a period of approximately one week. The interarrival times at homer and marge have a hyperexponential distribution [3]. These have been divided into their component exponential distributions.[1] The interarrival times at **loki** have an exponential distribution. (We identified the component exponential distributions and then verified them using the Kolmogorov-Smirnov test.[2]) **Bastion** and **Babble** do not display exponential distributions for interarrival times. The division is based on the type of requests made. For example, on homer the short interarrival times occur when a series of requests were made that had a short response time, such as a fetchstatus request, while the longer interarrival times were seen between successive fetchdata requests. The presence of a very large cache at the client end reduces the traffic at the server resulting in fewer requests for large volumes of data. However, the Cache Manager [5] needs to make a large number of status requests to get attributes for files, to ensure that the data in the cache is valid. This makes up a large part of the first distribution consisting of short interarrival times. The second distribution is composed of request types that take longer to process at the server. The fetchstatus requests that appear in the second

1. A function that has a shape similar to a negative exponential distribution, but with greater variability, is called a hyperexponential function. In a queueing model, a hyperexponential distribution can be approximated by a weighted sum of exponential distributions.

2. The Kolmogorov-Smirnov (K-S) test is used to test if a sample of *n* observations is from a specified continuous distribution. In this case, we are checking to see if the observations fall into an exponential distribution.

distribution are often interspersed with fetchdata requests. Refer to Table 10 for the mean response times of each request type.[3]

Table 2: Interarrival and response times in seconds for bastion and babble. The interarrival times dont have an exponential distribution, hence only the mean values are shown. The response times are grouped by component hyperexponential distributions.

	Bastion Interarrival Time	Bastion Response Time	Babble Interarrival Time	Babble Response Time
Total number of calls	206626	206626	397353	397353
Median	0.408191	0.000788	0.028048	0.004144
Mean	2.92768	0.000497	1.57912	0.000388
Std Deviation		0.000114		0.001313
Coeff of Variation		0.2298		0.3378
% of calls	38.13	38.13	81.87	81.87
Mean	2.92768	0.003893	1.57912	0.057600
Std Deviation		0.003331		0.054008
Coeff of Variation		0.8556		0.937
% of calls	61.86	58.94	18.12	17.74
% Discarded	0	2.9	0	0.37
Dates collected	04/26-05/03	04/26-05/03	04/24-05/01	04/24-05/01

The response time distributions at all the servers is hyperexponential and has been broken up intothe component exponential distributions in the table. (**Loki** has three component exponential distributions while all the others have two.) Each set of data (mean, standard deviation, and coefficient of variation) corresponds to one exponential distribution. The first set of values for the interarrival times, and the first set of values for the response times were taken from the same data set. This holds true for the each succeeding set as well. The values from a corresponding pair of interarrival and response times for a particular server are used in the computation of utilization in Section 4.1. The percentage of data that was discarded as outliers is also given in the table.

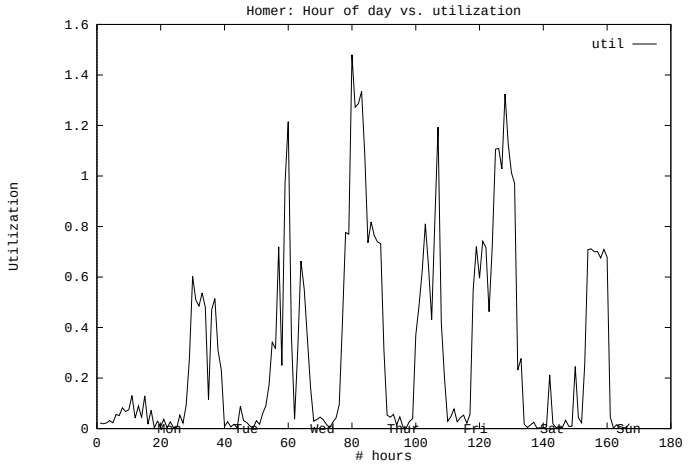
Table 3 shows the mean number of clients that generated at least one RPC per hour on each of the servers. This is seen to have very little variation throughout the week (as seen from the very low coefficient of variation). This, therefore, is no longer a factor that can contribute to the variation in response times. We also found that the number of cases where requests overlapped in time (i.e., more than one thread was active) was <1% of the total number of RPCs directed at a server. Hence, we have ignored the effect of having more than one request being processed simultaneously at the server.

Table 3: Number of active clients per hour.

	Loki	Homer	Marge	Bastion	Babble
Mean	97.455	110.818	94.175	58.39	17.801
Std Deviation	5.723	18.279	19.319	8.07	4.741
Coeff of Variation	0.058	0.164	0.205	0.138	0.266

3. The set of traces with short interarrival times on homer had the following request distribution: fetchstatus 81.17%, gettime 8.58%, getvolumestatus 3.18%, getstatistics 1.98%, giveupcallbacks 4.92%. The second set of traces had the following request distribution: fetchdata 15.709%, fetchstatus 26.46%, storedata 18.71%, storestatus 16.14%, createfile 8.86%, removefile 6.29%, rename 2.27%, gettime 3.14%. On loki, the set of traces with short interarrival times had the following request distribution: fetchstatus 89.82%, gettime 4.41%, getstatistics 1.87%, getvolumestatus 2.76%. The second set of traces had the following request distribution: fetchdata 32.11%, fetchstatus 30.28%, giveupcallbacks 18.43%, storedata 6.4%, gettime 4.03%, createfile 4%, removefile 3.14%. Only request types that constituted $\geq$ 1% of the traces have been listed.

Figure 2. Homer: Utilization on a per hour basis



4.1 Analysis of Data

From Tables 1 and 2 one can see that most of the interarrival and response times can be modeled by exponential distributions. Hence, the servers can be assumed to be M / M / 1 or G / M / 1 queues, and analyzed based on the formulae given below. U_i represents the utilization at the server. Since the number of times more than one thread is active in the server is less than 1%, we assume $i = 1$.

Utilization $U_i = X_i S_i$ where

X_i = throughput = arrival rate = exit rate

S_i = service time

90 - percentile of the response time = p_i = 2.3 * mean response time

(this means that 90% of the measures response times will be below p_i)

We calculate the values for each of the servers based on the above formulae from Tables 1 and 2. The two adjacent columns for a server have the interarrival time (from which the arrival rate can be computed), and the response time (which is the same as the service time here). **Loki** has only one distribution for interarrival times. The values from this distribution are used with the three response time distributions to get U_{11}, U_{12}, and U_{13}. For the other servers, the values from corresponding sets are used to compute U_{1j}. All time values are measured in seconds. Utilization is a dimensionless value, while 90 percentile response time is in seconds. The utilizations for the different interarrival distributions on the various servers are given below. The values in parentheses indicate the percentage of the traces recorded on that server which fall into this category. Note that the utilization values are taken over the entire trace and at any time are randomly fluctuating between high and low values. (Figure 2 is an example plot derived from data for **homer**, to show these fluctuations on a per hour basis.) The high values for **homer** occur when external users (people outside CITI) appear to be accessing directories or files. During the time the traces were collected, at least 32% of the calls made to **homer** were from clients outside the *umich* cell. Another 20% of the calls were made by the AFP[4] -AFS translator.

4. AppleTalk Filing Protocol

LOKI:

$U_{1_1} = 0.025\,(78.79\%)\,; U_{1_2} = 0.0816\,(18.16\%)\,; U_{1_3} = 1.26\,(2.84\%)$

$p_{1_1} = 0.0046\,(78.79\%)\,; p_{1_2} = 0.0148\,(18.16\%)\,; p_{1_3} = 0.2291\,(2.84\%)$

HOMER:

$U_{1_1} = 0.0046\,(57.41\%)\,; U_{1_2} = 1.17\,(41.76\%)$

$p_{1_1} = 0.00126\,(57.41\%)\,; p_{1_2} = 0.35581\,(41.76\%)$

MARGE:

$U_{1_1} = 0.0206\,(86.45\%)\,; U_{1_2} = 0.2567\,(13.42\%)$

$p_{1_1} = 0.00129\,(86.45\%)\,; p_{1_2} = 0.2271\,(13.42\%)$

Bastion and **babble** do not display an exponential interarrival time distribution. They do, however, show an exponential service time distribution. Hence, they can be modeled by a G/M/1 queue. **Bastion**[5] and **babble**[6] are far less utilized than the other three servers, and typically have long gaps between requests.

For a G/M/1 queue,

Utilization $U_i = 1/(E[\tau]\,\mu)$ where

$E[\tau]$ = mean interarrival time (in seconds)

μ = service rate in jobs per second

BASTION:

$U_{1_1} = 0.00017\,(38.13\%)\,; U_{1_2} = 0.00133\,(58.94\%)$

$p_{1_1} = 0.0011451\,(38.13\%)\,; p_{1_2} = 0.0089559\ldots\,(58.94\%)$

BABBLE:

$U_{1_1} = 0.00246\,(81.87\%)\,; U_{1_2} = 0.036476\,(17.74\%)$

$p_{1_1} = 0.008944\,(81.87\%)\,; p_{1_2} = 0.13248\,(17.74\%)$

5. **Bastion** is used for the most part as a Volume Location DataBase (VLDB) server. It also serves some read-only volumes.

6. **Babble** is a file server for a small set of users

Most of the servers seem to have fairly low utilization.

Table 4: Read/Write percentage distribution at the servers

Reads / Writes	Loki	Homer	Marge	Bastion	Babble
File Reads					
$\leq 4K$	36.6994	45.6667	53.1334	88.5699	71.798
$4K < size \leq 12K$	15.9017	9.86528	8.63044	0.0521921	12.841
$12k < size \leq 64K$	47.3989	44.468	38.2362	11.3779	15.360
Dir. Reads					
$\leq 4K$	97.8325	96.3048	98.4666	100	90.113
$4K < size \leq 12K$	1.8606	3.31426	1.33355	0	8.68173
$12k < size \leq 64K$	0.292621	0.349759	0.19283	0	1.03578
File Writes					
$\leq 4K$	56.3605	66.7036	50.5933	100	54.8088
$4K < size \leq 12K$	14.5703	10.802	8.6565	0	6.21089
$12k < size \leq 64K$	29.0692	22.4974	40.7502	0	38.9803

4.2 Server Workload Model

The purpose of using a workload model is to be able to generate a typical load on a server to measure performance indices. The synthetic workload should be *representative* [1] of the real workload on the server for accurate results. It is not necessary for the synthetic workload to do exactly the same things as a real workload as long is it generates the same type of load on the server. Thus, instead of using the exact mix of requests, it is possible to split them into requests with short response times (e.g., fetchstatus and gettime), requests with medium response times (e.g., fetchdata and setvolumestatus), and requests with long response times (e.g., removefile and storedata).

The server is modeled as an M / M / 1 server with the mean for $x \leq 32K$, the exponential interarrival times and response times obtained from Tables 1 and 2. The type of request generated is based on the frequency distribution of short, medium, or long response requests, obtained from the information in Table 11.

The read / write size distribution for files is bimodal with an exponential distribution for $x < 32K$ and a mode at 64K (the chunk size configured in the clients at CITI. This is the basic size of transfer of data between the cache manager on the client and the server). The read distribution for directories is exponential. Graphs 3 - 5 in Appendix A show the read / write size distributions for files and directories at all the servers.

5. Clustering Based on User Type

In order to see patterns in referencing behavior based on user types, we chose a small number of each type of user and did the analysis based on the traces generated by the selected subset of users. Clustering was observed for file referencing patterns and read / write size distributions. We also show in the example in Section 6 that on **homer** the number of disk I / Os, mean response times, and mean percent of calls that require disk I / O for each of the user types tend to form clusters.

It is possible to determine the sample size required to obtain statistically significant results by using the mean and standard deviation obtained from a small subset of the population [6]. Section 5.1 shows how to do this given the mean and standard deviation for certain parameters obtained from five technical writers, eight developers, four students, and five system administrators. We use the method in the following section to obtain valid sample sizes from which we obtain the means and standard deviations for different parameters in Section 5.2.

In Section 5.2, data has been gathered based on the assumption that each separate login session is the same statistically as having a new user of a given type. In order to validate this assumption, we need to prove two things. First, a user can be uniquely identified as belonging to a certain user type. In other words, not only do members of a cluster have similar characteristics, but they also have distinctly different characteristics from members of different clusters. Second, using different login sessions of the same user does not have a significant impact on the mean values computed for a given user type.

To prove that a user belongs to a distinct user group, we computed confidence intervals for the mean values.[7] The confidence intervals were non-overlapping at 90% confidence levels, which shows that the users can be distinctly identified based on their mean values to fall into a particular class of users.

To show that using different login sessions of the same user does not have a significant impact on the mean values computed for a given user type, we performed a two-factor, full factorial experiment. The first factor was the mean values from login sessions of the same user. The second factor was mean values from login sessions of different users. These mean values were used to obtain the allocation of variation to the two factors, and to unexplained factors or errors. The F-ratio[8] was compared against a table of quantiles of F-variates to determine the significance of the two factors. The results showed that at a 95% confidence level, both factors did not have a significate impact on the mean values.

5.1 Determining Sample Size

The minimum sample size for a user type to accurately determine mean values can be obtained using:

$$n = \frac{(100sz)^2}{r\bar{x}} \text{ where:}$$

n = sample size, z = normal variate of the desired confidence level, r = accuracy (implies confidence level), x = mean, and s = standard deviation. These values are obtained from a small subset of the user type.

Based on the measurements from five technical writers, the mean percentage of total file related (not general, directory, or volume related) calls by a user is:

$$\bar{x} = 82.063443$$
$$s = 19.895978$$
$$r = 4.878 \text{ (for accuracy within four calls in every 82.06 calls)}$$
$$z = 1.96 \text{ (from standard tables for a 95\% confidence interval)}$$

From the formula:

$$n_{tech\_wr} = 94.898$$

7. A *confidence interval* for a mean at confidence level 100 * (1 - α) states that the population mean will fall in that interval with a probability of 1 - α.

8. F_A = F ratio to test the significance of factor A is the ratio of (variation due to factor A/ number of degrees of freedom of A) to (variation due to unexplained errors/number of degrees of freedom of errors).

Using the same reasoning, the minimum sample sizes required for other user types are:

$$\bar{x} = 80.783; s = 8.0462; r = 2.4757; (2\text{calls}/80.78); z = 196; n_{sysadmin} = 62.18$$

$$\bar{x} = 79.897; s = 7.484; r = 1.251; (1\text{call}/79.89); z = 1.96; n_{dev} = 215.42$$

$$\bar{x} = 66.15; s = 24.179; r = 4.535; (4\text{calls}/66.15); z = 1.96; n_{student} = 140.408$$

We have assumed separate login sessions as a new user of the given type, as we do not have the required number of each user type in our department. We have observed that the variation in usage between separate login sessions by the same user is as high or as low as the variation between different users of the same type. Based on this assumption, the actual values for the sample sizes we have used are $N_{tech\_wr} = 100$, $N_{students} = 204$, $N_{sysadmins} = 164$, $N_{dev} = 353$. A new login session is assumed to have started if there is more than a one hour gap between RPCs generated by an individual.

5.2 Clustering Analysis

In order to see if a cluster emerges, we determine the mean (μ) and standard deviation (σ) for the parameter of interest from the data in each login session. The coefficient of variation (σ/μ) tells us how closely the values are clustered (low values imply the presence of clusters).

Table 5 shows the average number of calls made per hour by each user type. These values have been collected only over active hours (number of calls > 0). The coefficient of variation is very high for these numbers, particularly for developers. This implies that no pattern emerges for number of calls made per user type per hour.

Table 5: Number of RPCs generated per hour

	Tech Writers	Sysadmins	Students	Developers
Mean	142.55	237.894	61.019	990.83
Median	72	134	23	116
Standard Dev.	156.166	278.4201	94.735	3013.6382
Coeff. of Variation	1.09	1.17	1.55	3.041

Table 6 shows the percentage of requests that were related to files (as opposed to directories or volumes) per active login session. The coefficient of variation is low in all cases, which indicates these numbers are a good representation of the clustering effect based on user type.

Table 6: % of rpc's related to files per active session

	Tech Writers	Sysadmins	Students	Developers
Mean	68.98	64.21	51.374	57.95
Median	71.428	64.248	58.333	61.111
Standard Dev.	19.03	13.733	24.74	20.702
Coeff. of Variation	0.275	0.213	0.481	0.357

Table 7 shows the (mean) values for read distribution for files based on user type. The coefficient of variation is very low for the case $\leq$ 4K. In the other cases it varies from acceptable (< 1.0) to high (> 1.0). This pattern follows Ousterhout's [4] statement that most read requests are for $\leq$ 11K bytes. The read size distribution for files is bimodal. (It has an exponential distribution up to 32K, and then another mode at 64K). In most cases, the coefficient of variation is low, implying these are representative numbers for each of the user types. The write distribution was found to be multimodal. These distributions can be modeled by a uniform distribution. If greater accuracy is required, the

distributions can be observed and characterized at the range level (e.g., exponential for *size* $\leq$ 12K) and the sub-distributions can be used in the model [7].

Table 7: Read size percentage distribution for Files

	Tech Writers	Sysadmins	Students	Developers
$\leq 4K$ standard dev.	87.228 3.161	72.629 15.088	55.986 18.008	66.923 21.121
$4K<size \leq 12K$ standard dev.	7.671 3.09	10.184 3.472	12.01 4.617	9.406 6.19
$12k < size \leq 64K$ standard dev.	5.1 2.48	17.186 12.334	32.002 15.978	23.67 15.951

5.3 Incorporating User Related Information into the Workload Model

The presence of clusters in user request patterns allows the incorporation of characteristics of different user types into a client model. To do this, we break the frequency distribution of requests on a per user-type basis. In section 4.2, we described the aggregate workload characteristics from the point of view of being able to analyze the performance of the server as it is today. Here, we describe the workload for each type of user, assign the number of users for each type, and then predict the performance of the server for the total set of users. As the parameters of a synthetic workload can be easily changed, we include the number of users of each type as configurable parameters of the workload model. This will allow us to determine the performance of the server under different configurations.

Table 8: Read size percentage distribution for Directories

	Tech Writers	Sysadmins	Students	Developers
$\leq 4K$ standard dev.	91.419 4.822	90.621 7.431	86.942 13.472	95.89 4.651
$4K<size \leq 12K$ standard dev.	2.841 4.356	7.912 7.356	10.824 9.972	2.677 1.857
$12k < size \leq 64K$ standard dev.	5.388 6.236	1.466 0.5677	2.233 3.369	1.415 3.563
$> 64K$ standard dev.	0.350 0.784	0 0	0 0	0.021 0.06

Table 9: Write size percentage distribution for Files

	Tech Writers	Sysadmins	Students	Developers
$\leq 4K$ standard dev.	89.667 7.453	65.178 20.606	58.725 12.88	68.677 23.696
$4K<size \leq 12K$ standard dev.	4.738 4.271	9.588 4.67	14.752 8.829	9.719 5.041
$12k < size \leq 64K$ standard dev.	5.594 6.518	25.233 18.169	26.522 8.538	21.628 19.358

6. Using Clusters to Predict Server Performance

An interactive system (such as the AFS environment at CITI) can be modeled by a closed queueing network. To quote Ferrari [2], ``In the terminology of analytic modelling, we can say that interactive installations are naturally represented by finite-population, closed queueing networks.'' In this section, we use the formulae for response time and system throughput obtained using mean value analysis. We extend the methods used in mean value analysis to solve closed queueing networks, to use the information we have based on user types, and to predict the mean performance at the server given n_i users of type i.

Using mean value analysis for a closed queueing network, we have:

$$\text{System response time } R = \sum_{j=1}^{M} R_j V_j \tag{6.1}$$

$$\text{System throughput } X = \frac{N}{Z+R} \tag{6.2}$$

where M = the number of devices (a server or CPU is considered a device), R_j = response time at device j, V_j = number of visitations to device j, N = total number of users, and Z= think time in an interactive system.

In any UNIX-like system, owing to the presence of a buffer cache, all writes do not necessarily induce a disk I/O. It might be the one unlucky writer who causes the buffer cache to fill up that might bear the brunt of the disk I/Os, all of which may not have been caused by that user. On the other hand, a read that induces disk I/O is most likely to be performed on behalf of the user.

Bearing this in mind, we extend the mean value analysis approach to use the clusters we have spotted.

Let M = 2 (server and I/O device)

N_i = number of users of type i

S_i = mean service time from server seen by user of type i (base service time without any I/O)

Dr_i = mean service time per disk read

Vr_i = number of reads

Qr_i = probability that user of type i will require disk read

Dw_i = mean service time per disk write

Vw_i= number of writes

Qw_i = probability that user of type i will require disk write

$N = \Sigma N_i$ = total number of users

Z = think time

$$R_i = \text{mean response time for user of type } i = S_i + Vr_i Dr_i + Qw_i Vw_i Dw_i \tag{6.3}$$

Given that writes by users will not be accurately accounted for in the traces, we use a random number generator to generate values for Vw_i, and then take the mean value. The values for Qw_i can be accurately obtained from the traces, and Dw_i is the time taken for a typical disk read/write.

Using the value for R_i in equation (6.3) we have:

$$\text{System response time } R = \frac{\sum N_i R_i}{N} \tag{6.4}$$

$$\text{System throughput } X = \frac{N}{Z+R} = \frac{N^2}{NZ + \sum N_i R_i} \tag{6.5}$$

Example

The following example shows how the response time and system throughput is computed, given the number of users of each distinct type. Consider the server **homer** that has 4 types of users on it. Let N_1 = number of technical writers = 10, N_2 = number of system administrators = 3, N_3 = number of students = 100, N_4 = number of software developers = 30.

The mean service times (without I/O) for the four user types are [9]:

$$S_1 = 0.001154; \quad S_2 = 0.002367; \quad S_3 = 0.005905; \quad S_4 = 0.002407$$

The disk I/O time is the same in all cases, i.e:

$$Dr_{[1\text{-}4]} = Dw_{[1\text{-}4]} = 0.02$$

The number of reads/writes are:

$$Vr_1 = 3;\ Vr_2 = 3;\ Vr_3 = 2;\ Vr_4 = 2$$

$$Vw_1 = 6;\ Vw_2 = 7;\ Vw_3 = 6;\ Vw_4 = 7$$

The probability that a read/write will occur is:

$$Qr_1 = 0.7379;\ Qr_2 = 0.5741;\ Qr_3 = 0.5363;\ Qr_4 = 0.5889$$

$$Qw_1 = 0.6486;\ Qw_2 = 0.3659;\ Qw_3 = 0.2667;\ Qw_4 = 0.3717$$

Based on equation 6.3 for R_i and using the above values:

$$R_1 = 0.123260; \quad R_2 = 0.088039; \quad R_3 = 0.059361; \quad R_4 = 0.078001$$

If the think time Z = 4 seconds, using equations 6.4 and 6.5:

R = System response time = 0.068341 seconds

X = System throughput = 35.149463 users/second

7. Conclusion

This study was undertaken to develop a model of AFS file servers that would enable us to do a performance evaluation of the file servers at CITI. We developed the model from traces collected over a period of 2-4 weeks. We also developed a model for users based on what class they fall into. The study of traces on a per user basis showed that users of a particular user type have some similar characteristics. The presence of clusters helped us incorporate the effects of different types of users

9. All of the values in this section have been obtained from actual data for user types on **homer**. In all cases except for the number of disk reads/writes for students, clustering of values was observed. In other words, the coefficient of variation was low.

into the workload model. In the future, this model will help us predict the load on file servers used by a specific set of users, and plan for it accordingly. We intend to create a synthetic workload based on this model for future performance analysis.

We are continuing to collect traces on a variety of machines. Using the methodology described in this paper, we should be able to obtain hard numbers for problems such as the one described in the example in Section 6.

A. Other Server Information

Tables 10 and 11 show the mean response time and frequency distribution, respectively, of the different request types. Graphs 3 - 5 give the read / write size distribution for files on directories on the different servers.

Acknowledgments

Sarr Blumson, Redha Bournas, Edna Brenner, Peter Honeyman, Dan Hyde, Dan Kiskis, and Toby Teorey provided helpful comments on drafts of this paper.

It was Sarr's idea originally to try and see if users of a particular type exhibited similar request patterns.

Table 10: Mean Response Times (in seconds) per Request Type

Request type	Loki	Homer	Marge	Bastion	Babble
fetchdata	0.088396	0.092157	0.079983	0.022080	0.164412
fetchacl	0.063622	0.097001	0.068517	0.019916	0.110259
fetchstatus	0.002561	0.010486	0.004946	0.002521	0.006708
storedata	0.085543	0.438169	0.355648	0.012908	0.398597
storeacl	0.052195	0.167890	0.155285	0.013305	0.163302
storestatus	0.006774	0.225456	0.169963	0	0.015065
removefile	0.030992	0.404734	0.383207	0.045296	0.257349
createfile	0.008389	0.365194	0.251659	0.002056	0.114772
rename	0.018879	0.402141	0.425978	0	0.035755
symlink	0.167842	0.387016	0.422286	0.038950	3.351884
link	0.002346	0.260487	0.242612	0	0.025765
makedir	0.024639	0.299366	0.334470	0.001020	0.168624
removedir	0.049200	0.408406	0.402609	0	0.258416
setlock	0.002797	0.181377	0.175139	0	0.011853
extendlock	0.001352	0.248188	0.157246	0	0.015046
releaselock	0.003818	0.180339	0.246670	0	0.006871
getvolumestatus	0.002026	0.001773	0.001344	0.001071	0.005027
setvolumestatus	0.001516	0.147793	0.054623	0.000982	0
gettime	0.009825	0.009672	0.010475	0.005639	0.008052
getstatistics	0.001301	0.001229	0.000590	0.000860	0
giveupcallbacks	0.004007	0.000625	0.000625	0.002349	0.005408
getvolumeinfo	0	16.200396	0	0.012462	0.046279

Table 11: Frequency Distribution of Requests at Server

Request type	Loki	Homer	Marge	Bastion	Babble
fetchdata	7.039670	6.953260	9.860930	2.776390	3.532970
fetchacl	0.007019	0.017360	0.005057	0.002033	0.019100
fetchstatus	76.823600	57.154100	66.325700	36.763600	69.128600
storedata	1.402120	8.245910	0.917970	0.000451	1.070410
storeacl	0.002244	0.005685	0.001848	0.000677	0.003224
storestatus	0.431460	7.081320	5.816300	0	
removefile	0.692869	2.768400	0.168550	0.000451	0.267910
createfile	0.881642	3.912730	0.468520	0.000677	0.496630
rename	0.164848	1.002280	0.082382	0	0.219539
symlink	0.003104	0.093867	0.008267	0.000225	0.004460
link	0.010123	0.142096	0.020911	0	0.062010
makedir	0.025214	0.305665	0.056705	0.045407	0.008434
removedir	0.006303	0.179561	0.007392	0	0.000496
setlock	0.003104	0.032490	0.214460	0	0.014139
extendlock	0.411930	0.010965	0.547403	0	13.776400
releaselock	0.013657	0.028277	0.151538	0	0.011650
getvolumestatus	2.246750	1.868720	9.113460	0.053314	0.014600
setvolumestatus	0.000286	0.002792	0.000097	0.000451	0
gettime	4.326030	6.182300	2.282300	34.891800	3.038080
getstatistics	1.468880	1.135750	0.426698	8.526630	0
giveupcallbacks	4.020202	2.876327	3.517558	0.000903	1.539400
getvolumeinfo	0	0	0	16.936900	5.894320

Figure 3: Loki, Homer, Marge, Bastion, Babble: Read Distribution for Files

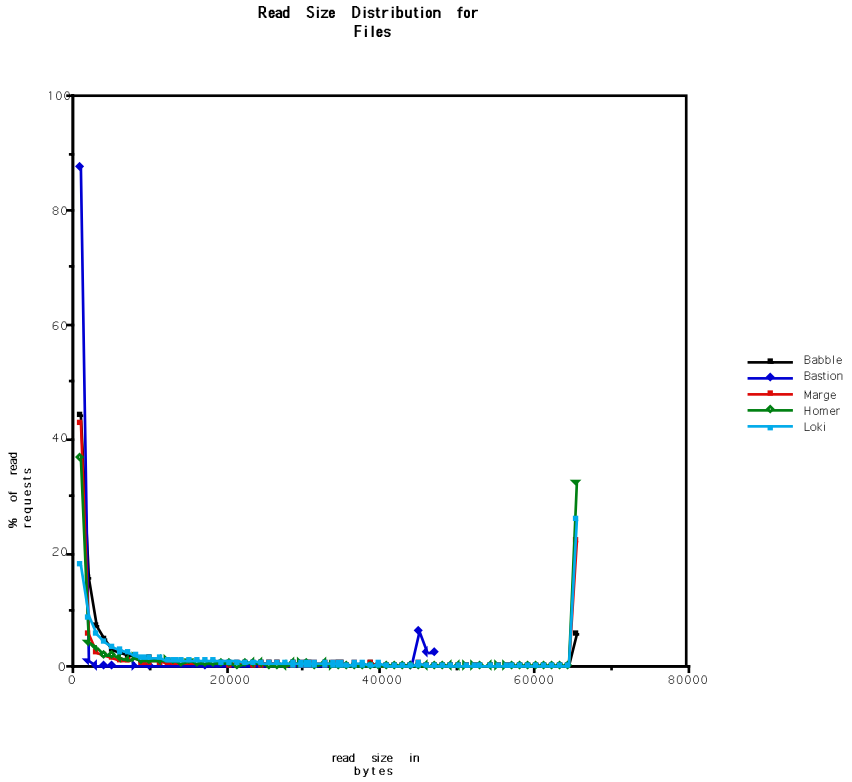


Figure 4: Loki, Homer, Marge, Bastion, Babble: Read Distribution for Files

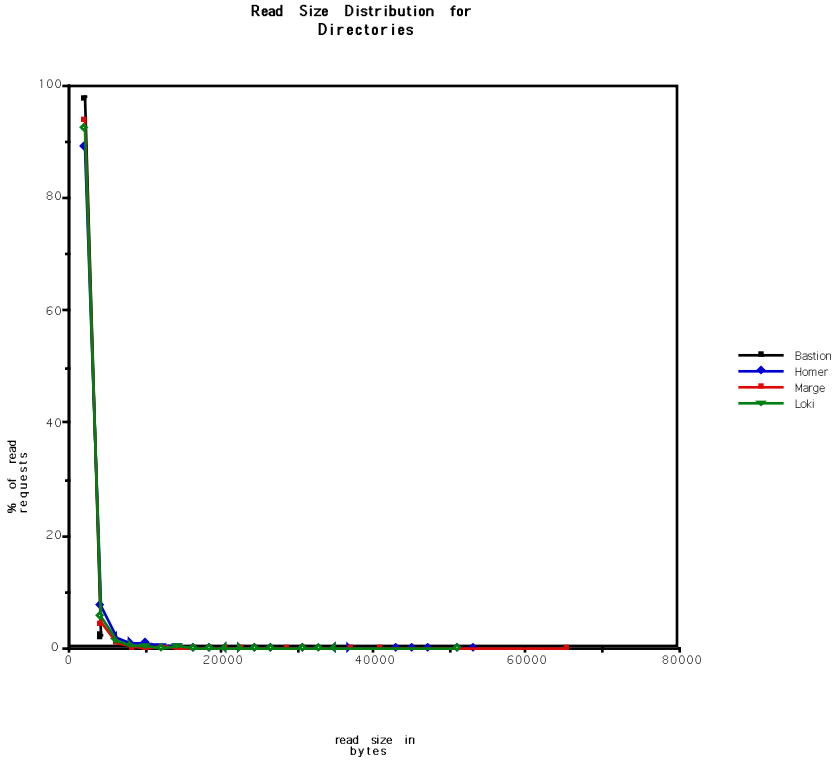
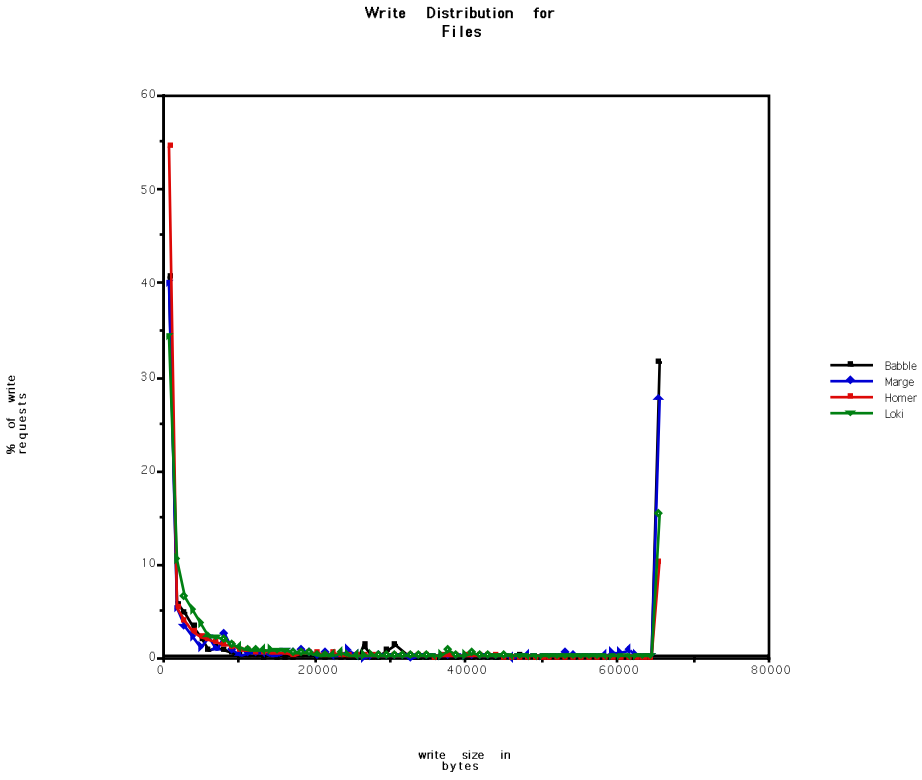


Figure 5: Loki, Homer, Marge, Bastion, Babble: Write Distribution for Files



References

[1] D. Ferrari. Workload characterization and Selection in Computer Performance Measurement. *Computer*, 5(4):18-24, July/August 1972.

[2] D. Ferrari. A performance-oriented Procedure for Modeling Interactive Workloads. *Experimental Computer Performance Evaluation*, pages 57-58, June 1980.

[3] D. Ferrari, G. Serazzi, and A. Zeigner. *Measurement and Tuning of Computer Systems*. Prentice-Hall, Inc., 1983.

[4] M.G. Baker, J.H. Hartman, M.D. Kupfer, K.W. Shirriff, and J.K. Ousterhout. Measurements of a Distributed File System. *Proceedings of the 13th Symposium on Operating System Principle ACM*, Oct. 1991.

[5] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, and M. West. Seals and Performance in Distributed File Systems. *ACM Transactions on Computer Systems*5(1): 1-8, Jan. 1992.

[6] R. Jain. *The art of Computer Systems Performance Analysis*. John Wiley & Sons, Inc., 1991.

[7] R. R. Bodnarchuk and R.B. Bunt. A Synthetic Workload Model for a Distributed System File Server. *1991 ACM Sigmetrics*, pages 50-59, May 1991.

[8] S. Blumson, P. Honeyman, T.E. Ragland, and M.T. Stolarchuk. AFS Server Logging. 1992.