Proceedings of the 1996 Winter Simulation Conference
ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain

MODELING FILE-SYSTEM INPUT TRACES VIA A TWO-LEVEL ARRIVAL PROCESS

Peter P. Ware
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210, U.S.A.

Thomas W. Page, Jr.
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210, U.S.A.

Barry L. Nelson
Department of Industrial Engineering and Management Sciences
Northwestern University
Evanston, IL 60208, U.S.A.

ABSTRACT

A method for analyzing, modeling and simulating a two–level arrival–counting process is presented. This method is particularly appropriate when the number of independent processes is large. The initial motivation for this method was the need to analyze and represent computer file system trace data that involves activity on some 8,000 files. The method is also applicable to network trace data characterizing communication patterns between pairs of computers.

Cluster analysis with a novel stopping rule is used to decompose the arrival process into groups. The resulting clusters can be characterized using the time between clusters, the time between arrivals within clusters, and the size of each cluster. Each of these three components is then analyzed as a univariate problem.

The effectiveness of this method is measured by comparing the output of a simulation driven by the original trace data to the output of the same simulation driven by the input model.

1 INTRODUCTION

The motivation for this work is the desire to drive simulations of distributed, replicated, file systems. Much like network traffic or memory reference patterns, workloads presented to file systems tend to be bursty and exhibit a high degree of locality. That is, a file which has been accessed recently tends to be accessed again soon. Further, accesses tend to come in *clusters*; there is a burst of activity, followed by a break, followed by another burst, etc. Figure 1 is the access pattern for one such file.

We can drive file system simulation with one of several available sets of trace data. However, there are a number of limitations to the use of traces. Traces are of finite length. If the events of interest in the simulation occur sufficiently infrequently (as they do in our application), then the length of the trace may be insufficient for quality results. Despite their limited size, traces are nevertheless voluminous and hence cumbersome. They are highly inflexible, representing only what happened over some specific interval of time, which may or may not be more widely representative. Thus, even when in possession of trace data, there is motivation to produce a synthetic workload which can capture important aspects of the recorded trace, but with greater flexibility. The work reported here is a step along the path to building such a synthetic trace generator for file accesses.

1.1 Data Characteristics

Figure 2 shows the inter–arrival times for our computer file system trace data (Hisgen 1990). This data totals over 2 Gigabytes, representing approximately 29 million events collected over a four day period from 114 workstations running a Unix-like operating system. This data appears very nearly exponentially distributed. However, this view from 10,000 ft. (an altitude from which the identity of the file being accessed is obscured) is too coarse for our simulation purposes and such a workload model would not capture the critical characteristics of the arrival process. If one zooms in closer to where one can observe the data decomposed into a separate process per file, a very different pattern emerges.

Figure 1 shows this arrival–counting process at a more detailed level (access to a single file), illustrating some important characteristics. The data appears to be generated by a two–level process: one process generating clusters of events, and within each cluster, a second process generating individual events. A sequence of independent, identically distributed inter-arrival times would do a poor job of representing such patterns.

In Figure 1, the vertical axis is a cumulative count of the number of accesses to the file and the horizontal

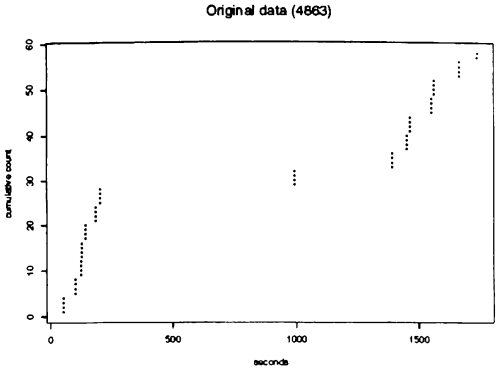

Figure 1: Several Clustered Events

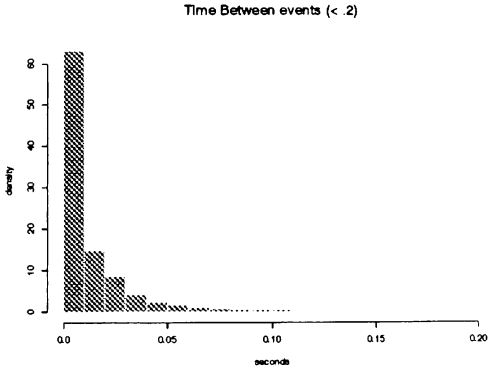

Figure 2: Probability Density Plot of Inter–arrival Times

axis is in seconds. Examining this figure, one could manually draw circles to cluster the data and model the time between the start of each cluster (the *inter–cluster* time) to form the first process. Then, within all the clusters, one could match a distribution to the time between each event (the *intra–cluster* time) to represent the second process within each cluster. A distribution for the *size* of each cluster can then be used to terminate the second process.

An automated method for doing such clustering is needed when a data set is composed of more than a handful of such fine–grained patterns. The data shown in Figure 2 is composed of nearly 8,000 such objects (0.73% of the samples (3038) are $>= 0.2$ seconds and are not displayed). Each file may have its own access pattern, some of which may be characterized by frequent long bursts, others by rare short flurries of activity, with still others showing little clustering. Manual clustering of such a large data set is impractical, yet no static parameters (say for maximum intra-cluster gap) yield good results over the entire non-homogeneous data set.

We adapt a well-known algorithm called *hierarchical clustering* with *single linkage*, to measure the distance between clusters. Although some of the literature on hierarchical clustering disparages this method, it has ideal characteristics for this type of data, as shown in Section 3.

The final (and critical) step in *hierarchical clustering* is to determine when clustering should be stopped (between the extremes of one cluster per event and one cluster containing all events). Frequently, such a decision is reached manually; however this is again not practical for such large data sets. Section 3.1 introduces a new stopping rule designed specifically for this application.

The methodology we propose uses the Johnson translation system to fit a distribution to the *inter–cluster times* and *intra–cluster times*, and the empirical CDF for the distribution of the *cluster size*. Each of these can be treated as an independent, univariate distribution if clustering successfully identified a two–level process.

The effectiveness of this methodology is evaluated in Section 5 by comparing the output of the same simulation driven by two representations of the same input process. The simulation is first driven by the actual trace data and then is driven by the arrivals generated from our two–level process. The output of the two simulations is compared to establish that the features of the trace data that most affect the simulation output are accurately represented by the input model.

2 BACKGROUND ON APPLICATION

The impetus for this work was the desire to answer the question, "How frequently do conflicting updates occur in a replicated data system using optimistic concurrency control?" A number of systems have been built with this principle (see for example Bayou from Xerox PARC (Terry, Theimer, Petersen, Demers, Spreitzer & Hauser 1995), Coda from CMU (Satyanarayanan, Kistler, Kumar, Okasaki, Siegel & Steere 1990) and Ficus from UCLA (Guy, Heidemann, Mak, Page, Popek & Rothmeier 1990)). However, due to differing architectures and environments, it has proved difficult to answer this question in a very general way using the actual systems. Hence, a flexible simulation was undertaken.

A replicated data object with n copies is modeled as a finite state machine with n "normal" states and

one "conflict" state (see Figure 3). In state n, all replicas are mutually consistent; that is, all updates to the object have been applied to all replicas (in the same order) and hence the replicas contain the same value. If an update (write) operation occurs while the object is in state n, the update is initially applied to one of the replicas, and the model transitions to state 1. State 1 models the situation where only one replica contains the most recent data value. The update then propagates to the other replicas asynchronously, and the model transitions up through states 2, 3, etc., as each additional replica is informed, until mutual consistency is restored and the model returns to state n. In any state $i : 1 \leq i \leq n$, i of the n replicas contains the most up-to-date data, while $n-i$ contain stale data.

If an update arrives while the object is in any state i: $i < n$, two possibilities result: either the update is applied to one of the replicas which already contains the latest version, or to one that does not. This is the essential difference between weak consistency and the traditional strongly consistent algorithms. Conventional algorithms prevent conflicts by restricting updates to one of the up-to-date replicas. However, the cost of doing so may exceed the cost of dealing with the occasional conflict, if conflicts are sufficiently rare and/or easy to repair. If the replica selection algorithm chooses one of the i replicas that is already up to date, then the state transitions back to 1 and update propagation resumes spreading knowledge of the new update. However, if one of the $n - i$ replicas containing an old version is selected, then a state exists in which there is no longer a total ordering of file versions, an *update–update* conflict exists and the model transitions to the conflict state. When in the conflict state, C, update propagations cannot in general restore a correct, where "correct" is defined as one-copy-serializable, mutually consistent replicated data value. Any further updates to the object while in the conflict state will leave the file in the conflict state. Another type of operation called "repair" is required to restore a new dominant version which must then be propagated to all of the replicas before the file is once again mutually consistent. This operation is ignored.

If we were to assume that updates are generated by a Poisson process with rate λ, that the replica selection policy is equally likely to choose any replica for initial application of the update, and the time to propagate an update to any individual site is exponentially distributed with mean $1/\mu$, then the model becomes Markovian and is easily solved analytically.

As we have seen, however, a Poisson process is a poor model of the bursty arrival process typical of file system access, and it is precisely the pattern of rapid updates that makes conflicts more likely. Hence it is critical that the input process accurately capture this characteristic.

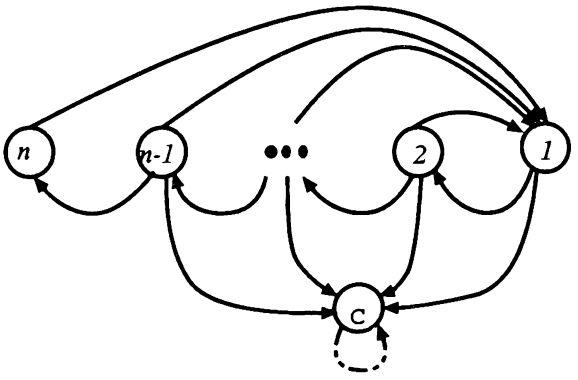

Figure 3: State Transition Diagram For An n Replica File

3 CLUSTERING

The hypothesis is that file access data (and many other types of workload) are effectively modeled by two–levels of processes; the first generating bursts, and the second generating events within bursts. In order to fit distributions for these processes for a given data set, it is necessary to "cluster" the data into bursts. We consider a form of agglomerative hierarchical clustering, as described, for example, in Jain (1991), but simplified by the fact that our application requires clustering in the time dimension only.

The data is in the form of an arrival process. Thus, each event has a time at which it occurs, and the events are totally ordered by their time. Let $T_1, T_2, \ldots, T_n$ be the sequence of access times for a particular file. The basic algorithm for agglomerative clustering is as follows:

1. Start with each event T_i in a cluster by itself and initialize the iteration counter i to 0.

2. Construct the inter–cluster distance array in which the j^{th} element is the distance between cluster j and cluster $j + 1$.

3. Find the smallest element of the distance array (if the smallest value is not unique, randomly choose one from the set of smallest distances). Say the k^{th} distance is chosen. Combine clusters k and $k + 1$.

4. Repeat steps 2 and 3 until all events are part of a single cluster (we examine earlier termination conditions below).

The above method produces a sequence of clusterings starting with each event in a cluster by itself, and terminating with all events in a single cluster. If this model is a good representation of the data set, then in between these degenerate cases lies one or more appropriate clusterings. The difficulty lies in automatically deciding which intermediate clustering is the best one. If we can determine that we have reached the point where any further iterations would combine clusters which should remain distinct, then we should terminate the algorithm and report the resulting clusters.

A second issue is how to compute distances between clusters. The statistical literature provides a large number of distance measures that are reasonable in various situations (Kaufman & Rousseeuw 1990). We employ one of the oldest measures, known as *single linkage* or *nearest neighbor*. The distance between two clusters is the minimum distance between any member of one cluster and any member of the other. Translated into our context, the "distance" between two clusters is the absolute value of the difference between the two closest event times. For example, if cluster j is $\{T_m, T_{m+1}, \ldots, T_{m+r}\}$, and cluster $j+1$ is $\{T_{m+r+1}, T_{m+r+2}, \ldots, T_{m+r+s}\}$, then the distance between them is $T_{m+r+1} - T_{m+r}$. Notice that employing single linkage implies that clusters will always be formed from sequential event times, a property that we obviously desire. In fact, the single linkage distance measure is criticized in the clustering literature for its tendency to form clusters that look like chains, but this is precisely the type of clusters for which we search.

3.1 Terminating the Algorithm

Define the random variable H to be the smallest value of the distance array chosen in each iteration. That is, H_i is the distance between the two clusters selected for merging in the i^{th} iteration of the clustering algorithm; it is monotonically increasing as the distance is always positive. Thus, the slope of the H_i curve is always nonnegative.

Intuitively, if the data is truly bursty, the early elements of the H_i sequence should be relatively small since we are combining clusters which are part of the same burst. At some step we will merge the last pair of clusters that are indeed part of the same burst. The next value of H_i represents an attempt to combine two clusters which are separated by an inter–cluster gap which should be very much larger than the greatest intra–cluster gap. At that point the slope of the H_i curve turns dramatically upward. We need to be able to recognize this point in an automated fashion.

Many clustering stopping rules have been proposed in the statistical and application literature. For example, Milligan & Cooper (1985), describes and tests 30 such rules! Most of these rules are based on the relative variability within clusters to between clusters. Our method exploits the fact that our data is an arrival counting process. Specifically, we compute a finite–difference estimate of the second derivative of the H_i curve, and end clustering when this derivative is significantly different from 0. The following argument justifies this approach:

Suppose that the file–access process is completely random, meaning that no clusters physically exist. Such a process is often well modeled as a Poisson process with arrival rate λ events per unit time. In other words, the inter–event times $G_i = T_{i+1} - T_i$, $i = 1, 2, \ldots, n-1$, are independent and identically exponentially distributed random variables with mean $1/\lambda$. Because of the nature of the single linkage distance measure, the heights $H_1, H_2, \ldots, H_{n-1}$ will therefore be the order statistics (sorted values) of $G_1, G_2, \ldots, G_{n-1}$. Thus, the smallest inter–event gap will determine the first cluster, the second smallest gap the next cluster, and so on. Using known results for the order statistics of the exponential distribution (e.g., Devroye (1986))), the expected increase (finite–difference first derivative) of H_i over H_{i-1} is

$$\mathrm{E}[H_i - H_{i-1}] = \frac{1}{\lambda(n-i)}. \qquad (1)$$

When n, the number of event times, is large, and i is not too close to n, then (1) will be nearly constant for values of i close together. Therefore, the local increase in H_i will be approximately linear, in expected value, and the second derivative approximately 0, for a process without clusters.

Our stopping rule looks for the behavior expected of a Poisson process. When the behavior of H_i departs significantly for what is expected, then we assume that we are no longer combining intra–cluster event times, and have started to combine clusters. Using the second derivative test avoids the need to specify a value of λ.

If we apply this rule to a Poisson event process, then it is likely to break the data up into a small number of very large clusters, since (1) does change rapidly for i near n. This implies that we should first subject our data sets to a Poisson–process test to determine if clustering is needed. If the hypothesis

of a Poisson process is rejected, then we apply the clustering algorithm.

On the other hand, if we apply this rule to a deterministic, completely regular process, it will place all observations in a single cluster, since the second derivative will be zero at all stages. Therefore, a single–cluster outcome should alert us to a data set that represents a regular process. However, the methodology continues to work as the inter—arrival times within one large cluster are fitted with some univariate distribution.

4 MODELING AFTER CLUSTERING

There are four steps in modeling and generating data after the sample data is clustered:

1. DATA ANALYSIS: Extract the observed inter–cluster times, intra–cluster times, and cluster–size, as represented by the random variables $\{\hat{X}_1, \ldots, \hat{X}_n\}$, $\{\hat{Y}_1, \ldots, \hat{Y}_m\}$, and $\{\hat{D}_1, \ldots, \hat{D}_n\}$, respectively. Here n is the number of clusters and $m = \sum_{i=1}^{n} \hat{D}_i - n$;

2. MODELING: Fit distributions to the observed inter–cluster times, intra–cluster times, and cluster–size data. Our software uses the `Fittr1` package (Swain, Venkatraman & Wilson 1988) to determine the parameters of the Johnson translation system (Ord 1972, Johnson 1949) for each of the random variables.

3. GENERATION: Use the parameters selected by `Fittr1` for random–variate generation of X, Y, and D. (Though D is discrete, we fit a continuous distribution and then round to the nearest integer to obtain the cluster size.)

4. SYNTHETIC TRACE: Use the generated values to form a two–level process. After an inter–cluster gap interval X_i, a new process is started. This process generates D_i events with the time between events given by $Y_j, \ldots, Y_{j+D_i}$, and then terminates. Note that for the i^{th} cluster, $j = 1 + \sum_{k=1}^{i-1}(D_i - 1)$.

At each interval X_i a new process is started. This process generates D_i events with the time between events given by Y and then terminates. Figure 4 shows this two–level process with the i^{th} process generating $D_i = 4$ events with the time between events given by $Y_j, \ldots, Y_{j+2}$. The $i + 1^{st}$ cluster is shown starting X_{i+1} time units later.

Figure 5 shows data generated for file 4863.

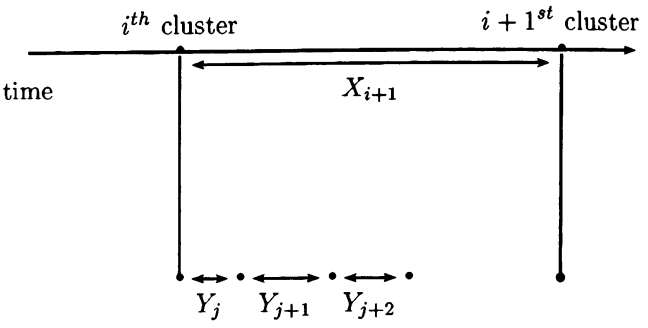

Figure 4: A two–level Process

5 VALIDATION EXPERIMENT

The purpose of this experiment is to test the hypothesis that the clustering method presented for generating file access traces which mimic the bursty behavior of the actual trace captures the characteristics of the trace data essential to the replicated file system simulation. If the results of driving a significant simulation experiment with the synthetic data are reasonably close to those generated with the original trace data, we will conclude that the hypothesis is validated.

5.1 Method

Twenty files out of 853 files that were accessed during the busiest two hour period of the data were selected at random. The clustering algorithm was run and then distributions fit to the inter-cluster gap, the intra-cluster gap, and the number of events per cluster. These distributions were then used to generate synthetic workloads for each file.

Next a simulation experiment was performed twice for each file: once driven by the actual trace data, and then driven by the corresponding synthetic workload model. A discrete event simulation of the replicated filing model presented in Section 2 was run repeatedly as μ (the rate at which updates are propagated from the most up-to-date replica) was varied from .001 to 1000. The number of replicas was fixed at 5. Each simulation run was terminated when the 95% confidence interval of the mean time to conflict (the mean time to absorption where conflicts are considered an absorbing state) was less than 5% of the mean. This results in a pair of curves for each file: one generated using the trace data and the other generated using the synthetic load model. Each curve plots the mean time to conflict as a function of the update propagation rate.

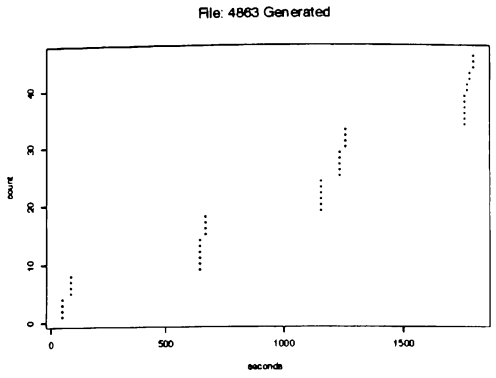

Figure 5: Generated Data Corresponding to File 4863 (see Figure 1)

Table 1: Summary Statistics For The Inter–arrival Times Of The Original and Generated Data for File 4863

Statistic	Original	Generated
mean	29.66	29.95
std. dev.	116	114
median	0.0021	0.0032
coef variation	3.93	3.83
lexis ratio	459.642	439.98
skewness	5.35	5.33
count	57	3307

5.2 Single File Illustration

Figure 5 shows the results of generating data for a single file. Visually comparing this to Figure 1 suggests similar patterns of clustering. The summary statistics in Table 1 show the original data and the generated data are close according to several measures. The mean and standard deviation are within 1%. The other values show the distributions are similar. Note the wide variability of the inter–arrival times as shown by the large standard deviation ($\sim$ 116).

Figure 6 shows the results of running the simulation with both the empirical and synthetic trace data. The mean time to conflict as a function of update rate is close for both data sets, lending support to the hypothesis that the generated data captures important characteristics of the empirical trace.

5.3 Results

The results of running the experiments are presented in Table 2. This table gives the relative error ex-

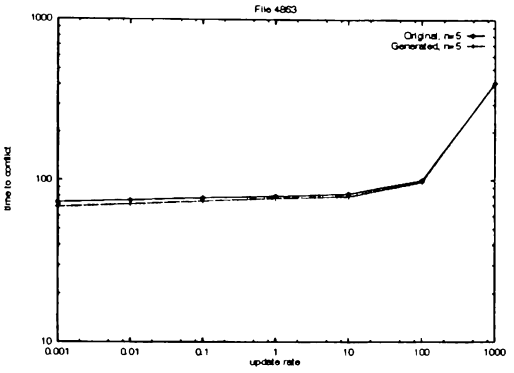

Figure 6: Mean Time to Conflict for the Original and Generated Data for File 4863

pressed as a percentage when comparing the conflict rate of the simulation using empirical data versus generated data. Generally, the results are satisfactory. Most results were within 30% of the original data.

The results that differed more than 30% come from two sources of error. The first is a basic limitation in using empirical data: the inter-arrival times are fixed even though the data suggests that smaller or larger values may exist. In this particular experiment, the mean time to conflict is closely related to the minimum inter–arrival time. A smaller minimum inter–arrival time causes conflicts to occur more often. Since the generated data can have smaller inter–arrival times, the mean time to conflict tends to be smaller for the generated data, especially as the update rate increases.

The other source of error is introduced by the sensitivity of the method to any errors in the *cluster size* distribution. The size of the clusters affects how many of the intra–cluster arrival times are generated. Since these are the predominant cause of conflicts (the intra–cluster times are typically an order of magnitude smaller then the inter–cluster times), changing the number such events can drastically affect the mean time to conflict. For our eventual file system research, this is not problematic as the nature of file systems will tend to treat clusters as a whole. For other purposes, this aspect needs refining.

6 SUMMARY

This paper presents a methodology for analyzing, modeling and simulating certain arrival–counting processes. The method is appropriate when (1) the number of such processes is too large to analyze manually, and (2) the processes resemble two–level pro-

Table 2: Relative Errors at Each Update Rate for Each File

	Update rate (1/seconds)						
File	**.001**	**.01**	**.1**	**1**	**10**	**100**	**1000**
5503	-26.48	-12.77	9.67	1.97	0.16	0.00	0.00
5479	-22.44	-9.81	9.85	1.04	0.05	0.00	0.00
3781	-18.25	-12.68	-11.21	-11.85	-18.69	-4.99	0.24
5260	-5.72	-0.12	0.42	2.85	0.12	0.00	0.00
4863	-5.51	-5.96	-7.79	-7.72	-7.35	-3.60	-0.54
7187	-4.79	-2.74	1.66	-0.06	0.00	0.00	0.00
5570	-3.96	-5.46	-1.75	-0.07	0.00	0.00	0.00
0524	-1.84	-2.14	-1.33	-0.84	-4.03	-12.30	-3.81
5488	-1.26	2.03	3.87	-2.51	0.01	0.00	0.00
7324	-0.19	0.53	-2.18	0.48	0.11	0.00	0.00
5580	1.76	6.50	17.19	1.56	0.07	0.00	0.00
5155	4.66	15.03	24.04	2.22	0.16	0.00	0.00
7174	8.84	8.48	11.89	1.07	0.12	0.00	0.00
1087	32.94	33.39	33.10	29.38	33.67	33.48	1.45
5532	46.40	57.34	42.45	6.51	0.68	0.06	0.00
0098	52.78	53.17	42.52	-15.05	-5.57	0.27	0.05
6491	54.63	59.42	46.22	4.74	0.35	0.00	0.00
5043	161.63	160.16	42.54	-17.41	-9.91	-4.14	-0.07

cesses. Trace data from both computer file systems and from communications networks typically fit these characteristics. In the case of file system traces, the data is decomposed into separate traces for each file accessed; for network traffic, communications between distinct pairs of source and destination sites constitute an appropriate decomposition. The purpose of this input modeling effort is to be able to generate synthetic traces that exhibit similar "burstiness" to the original recorded data.

The method first uses hierarchical clustering to group the data into bursts. A novel stopping rule is presented which allows the method to use different stopping parameters for each file. This is in contrast to previous work (see for example Jain & Routhier (1986)) which required a single stopping rule be applied to the whole data set.

A simple simulation modeling effort is described whose goal is to predict the frequency of concurrent updates in a lazy-update propagation replicated distributed file system. The system is very sensitive to the burstiness of the update traffic. Hence the simulation is a good test of the degree to which the method captures this characteristic of the trace data.

Preliminary results lead us to believe that the method does a good job at modeling file system trace data.

6.1 Limitations and Follow-up Work

A limitation that should be noted is that no attempt has been made to model the time varying nature of the workload. It is well known that file system workloads are highly cyclical. For the purposes of our simulation application however, we are more interested in the worst case scenario which is represented by the busiest sections of the trace. Hence, no work is planned in this area.

As of yet, we have not attempted to model the mix of file operations. The experiment reported utilizes only update events. Further work is required to generate an appropriate distribution of event types (read, lookup, create, etc.). Similarly, identity of the site making the file system access has been ignored to this point.

A longer range goal of this work is to produce a flexible file system load generation tool. Given a set of trace data for an existing system, the tool should be able to generate events which mimic the behavior of the existing system. But further, it should be parameterized so that it can be tuned to generate predictable workloads for file systems for which no trace data are available, either because they do not yet exist, or because instrumenting them to gather the traces is infeasible. For example, one might wish to generate a trace for a file system which mimics an existing one, but has 10 times as many users accessing 5 times as many files. The work presented here is a step along that path.

REFERENCES

Devroye, L. (1986), *Non-Uniform Random Variate Generation*, Springer-Verlag, New York, NY.

Guy, R. G., Heidemann, J. S., Mak, W., Page, Jr., T. W., Popek, G. J. & Rothmeier, D. (1990), Implementation of the Ficus replicated file system, *in* 'USENIX Conference Proceedings', USENIX, pp. 63–71.

Hisgen, A. (1990), Dec firefly trace data, Data obtained from author on 8mm tape.

Jain, R. (1991), *The Art of Computer Systems Performance Analysis: Techniques for Experiment Design, Measurement, Simulation and Modeling*, John Wiley & Sons, Inc.

Jain, R. & Routhier, S. A. (1986), 'Packet trains - measurements and a new model for computer network traffic', *IEEE Journal on Selected Areas in Communications* **SAC-4**(6), 986–995.

Johnson, N. L. (1949), 'Systems of frequency curves generated by methods of translation', *Biometrika* **36**, 149–176.

Kaufman, L. & Rousseeuw, J. (1990), *Finding Groups in Data: An Introduction to Cluster Analysis*, John Wiley & Sons, Inc, New York, NY.

Milligan, G. W. & Cooper, M. C. (1985), 'An examination of procedures for determining the number of clusters in a data set', *Psychometrika* **50**(2), 159–179.

Ord, J. K. (1972), *Families of Frequency Distributions*, Griffin, London.

Satyanarayanan, M., Kistler, J. J., Kumar, P., Okasaki, M. E., Siegel, E. H. & Steere, D. C. (1990), 'Coda: A highly available file system for a distributed workstation environment', *IEEE Transactions on Computers* **39**(4), 447–459.

Swain, J. J., Venkatraman, S. & Wilson, J. R. (1988), 'Least–squares estimation of distribution functions in Johnson's translation system', *Journal of Statistical Computation and Simulation* **29**, 271–297.

Terry, D. B., Theimer, M. M., Petersen, K., Demers, A. J., Spreitzer, M. J. & Hauser, C. H. (1995), Managing update conflicts in bayou, a weakly connected replicated storage system, *in* 'Proceedings of the Fifteenth Symposium on Operating Systems Principles'.

AUTHOR BIOGRAPHIES

PETER P. WARE is a PhD student in the Computer and Information Science Department at The Ohio State University. He is interested in distributed operating systems and replicated file systems. See `http://www.cis.ohio-state.edu/~ware` for more information about him and this paper.

THOMAS W. PAGE, JR. is an Assistant Professor in the Computer and Information Science Department at The Ohio State University. He received his Ph.D. in Computer Science and Engineering from UCLA in 1989. His areas of interest include distributed operating systems, distributed and replicated file systems, concurrency control for replicated data, and distributed shared memory.

BARRY L. NELSON is an associate professor in the Department of Industrial Engineering and Management Sciences at Northwestern University. He is interested in the design and analysis of computer simulation experiments, particularly statistical efficiency, multivariate output analysis and input modeling. He is the simulation area editor for *Operations Research* and will be Program Chair for the 1997 Winter Simulation Conference.