Automatic Modeling of File System Workloads Using Two-Level Arrival Processes

PETER P. WARE and THOMAS W. PAGE, JR.
Ohio State University
and
BARRY L. NELSON
Northwestern University

This article describes a method for analyzing, modeling, and simulating a two-level arrival-counting process. This method is particularly appropriate when the number of independent processes is large, as is the case in our motivating application which requires analyzing and representing computer file system trace data for activity on nearly 8,000 files. The method is also applicable to network trace data characterizing communication patterns between pairs of computers. We apply cluster analysis to separate the arrival process into groups or bursts of activity on a file. We then characterize the arrival process in terms of the time between bursts of activity on a file, the time between file events within bursts, and the number of events in a burst. Finally, we model these three components individually, then reassemble the results to produce a synthetic trace generator. In order to gauge the effectiveness of this method, we use synthetically generated (simulated) trace data produced in this way to drive a discrete-event simulation of a distributed replicated file system. We compare the results of the simulation driven by the synthetic trace with the same simulation driven by the original trace data, and conclude that the synthetic data capture the essential characteristics of the empirical trace.

Categories and Subject Descriptors: C.4 [**Computer Systems Organization**]: Performance of Systems—*modeling techniques*; D.4.3 [**Operating Systems**]: File Systems Management—*distributed file systems*; D.4.8 [**Operating Systems**]: Performance—*modeling and prediction; simulation*; G.3 [**Mathematics of Computing**]: Probability and Statistics; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*discrete event*

General Terms: Algorithms, Design, Experimentation, Performance

Additional Key Words and Phrases: Clustering, data replication, file access patterns, file system, input modeling, replication, synthetic traces, trace driven simulation

This material is based on work supported by the National Science Foundation under Grant No. IRI-9501812.

Authors' addresses: P. P. Ware, T. W. Page, Jr., Department of Computer and Information Science, The Ohio State University, Columbus, OH 43210; email: ware@cis.ohio-state.edu; B. L. Nelson, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL 60208.

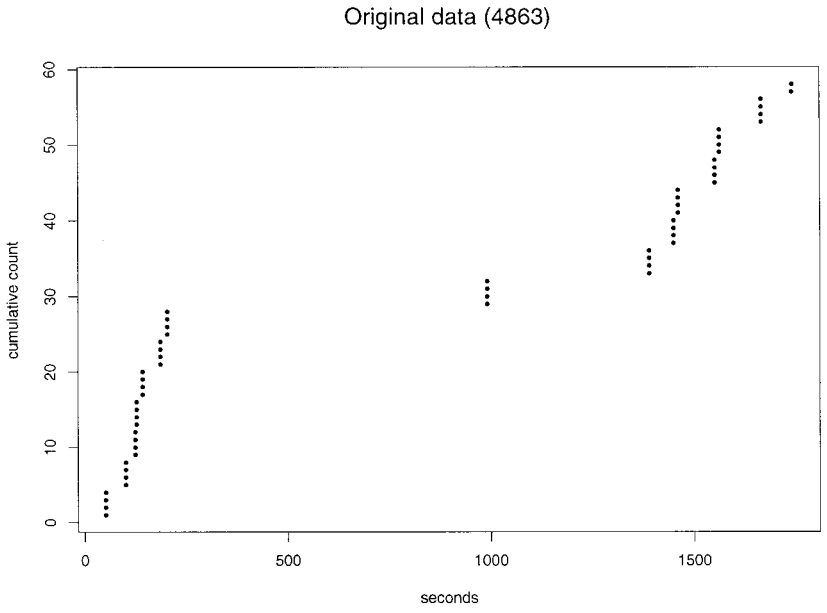

Fig. 1. Accesses to file 4863 (the vertical axis is a cumulative event count; the horizontal axis is time in seconds).

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 that has been accessed recently tends to be accessed again soon. Furthermore, accesses tend to come in "clusters"; there is a burst of activity, followed by a break, followed by another burst, and so on. Figure 1 shows the access pattern for one such file from a recorded trace. The bursty nature of the workload may have considerable impact on several aspects of file system architecture, affecting, for example, the performance of file caches. Some architectures for replicated file systems may be particularly sensitive to this bursty workload since a file may be updated multiple times during a burst before the update propagation subsystem has time to transmit the updates to all of the replicas. The desire to gauge the frequency of such "conflicting updates" via trace-driven simulation led to the work detailed here.

There are a small number of sets of trace data available for driving a file system simulation. 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, then the length of the trace may be insufficient for quality results. (Conflicting updates should be infrequent if an architecture in which they are possible at all is to be tolerated.) Despite their limited size, traces are nevertheless voluminous and hence cumbersome. They are also 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 that 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.

The central contribution of our work is to propose a flexible synthetic workload generation model, and to provide a robust automatic algorithm for fitting the model to trace data. The workload generation must be flexible in the sense of accomodating disparate access patterns and the algorithm can be run in a production mode, without manual intervention. The specific outcome of interest in our simulation model, the occurrence of conflicting updates, results primarily from the access pattern on individual files; hence it is critical to capture the access pattern of individual objects, and not simply some aggregate characteristics of the full trace data. High quality models are essential because the file system incident that interests us is a rare event whose occurrence is very sensitive to the per-file activity that drives the system. The need for hands-off modeling is forced by the large number of input models that must be developed; the data set described here represents access to almost 8,000 files. Robustness is critical because the data sets are heterogeneous, including completely regular sequences of events, entirely random (i.e., Poisson-like) sequences of events, and sequences of events that combine somewhat regular structure with a high degree of randomness.

1.1 Trace Data Characteristics

This article makes use of trace data gathered at DEC SRC [Hisgen 1990]. It totals over two gigabytes, representing approximately 29 million file system access events collected over a four-day period from 114 DEC Firefly workstations running the experimental (UNIX-like) Taos operating system. Trace events are timestamped with a millisecond accuracy. We selected for analysis the busiest two-hour subset of the data and did not distinguish between read and write events. The rest of this article deals with this subset. Clock synchronization algorithms running between the machines in the distributed file system maintain the collection of clocks in close (but not perfect) synchronization.

The traces were recorded at the system call interface on each machine and so represent a record of file access by users and programs. Traces gathered at this point are much more difficult and invasive to gather than are traces recorded from probes placed at the interface to servers that have already had cache hits filtered out at the client machines. However, it is critical to place the probe upwind of any caches if the measurements are to be used to judge caching or replica consistency.

Figure 2 shows a histogram of the interarrival times for the full set of trace data. Viewed this way, the marginal distribution of the interarrival times appears very nearly exponential, suggesting a Poisson arrival pro-

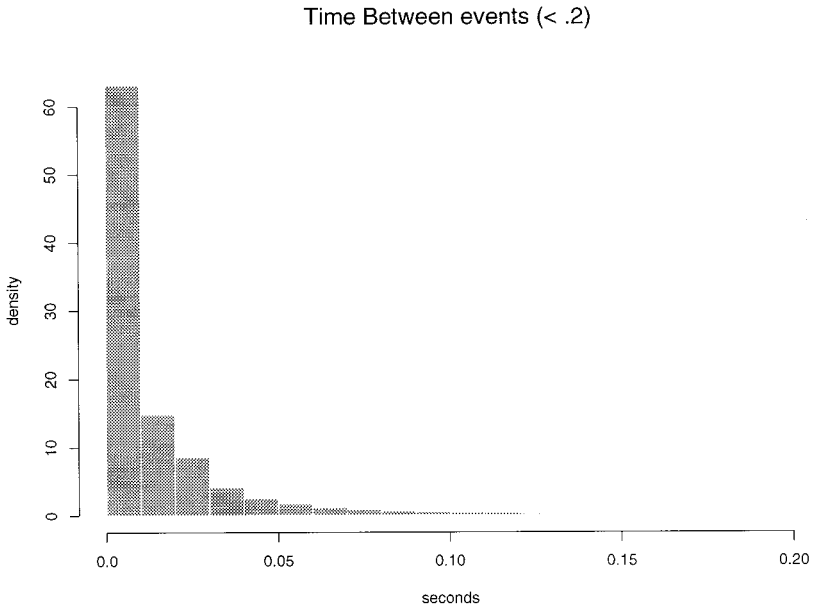

Fig. 2. Histogram of aggregate interarrival times less than 0.2 seconds for all files (3,038, or 0.73% of the sample are ≥0.2 seconds and are not displayed).

cess. However, this picture has obscured the identity of the file being accessed, a feature critical to our application. Even if file identity were not important and one were interested only in the aggregate arrival process, it would be incorrect to conclude that successive requests form a Poisson arrival process, which requires independent, as well as exponentially distributed, interarrival times. When one zooms in closer to where the decomposition into a separate process per file is visible, a very different pattern emerges and one can see strong dependence among interarrival times.

Figure 1 shows a portion of this arrival-counting process at a more detailed level (access to a single file), illustrating some important characteristics. The choice of file 4863 is based on its illustrative properties and not on file 4863 being a representative sample. The data appear 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 interarrival times would do a poor job of representing such patterns.

Figure 3 shows the histogram of interarrival times for this same file. Most of the files in the trace exhibit a similar shape: a large amount of mass near the origin, and then a smaller mass considerably farther out, with little in between. The histogram in Figure 2 is quite different as it represents the aggregate arrivals and so the interarrival times are quite small. At the file level, we consider the large number of short interevent times to be interarrival times within a cluster, whereas the smaller number

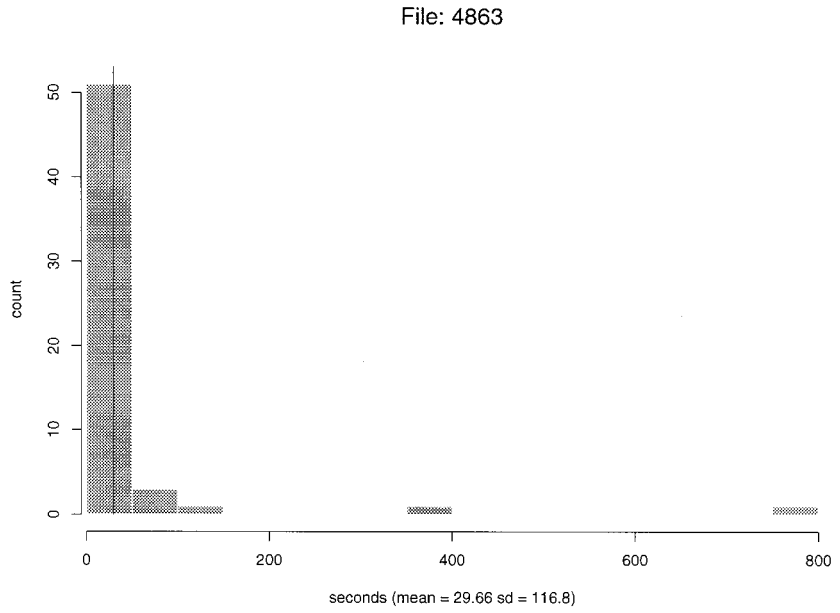

Fig. 3. Histogram of interarrival times for file 4863.

of larger interevent times represents the gaps between the end of one cluster and the first event of the next.

Given a plot of a distribution like that shown in Figure 3 for any one file, it may be possible to pick manually a value that lies between the mass at the left and that at the right and declare that value to be the maximum allowable intracluster gap. Given such a value, it is then easy to partition the interarrival times into two sets: those that represent gaps within a cluster (the *intracluster* times), and those that represent gaps between clusters (the *intercluster* times). This is equivalent to drawing boxes manually around the clusters in Figure 1. All interarrival gaps that lie entirely within boxes are intracluster gaps, and those not within boxes are intercluster gaps. A method that determines where to draw the boxes around each cluster should ensure that the maximum intracluster gap is less than the minimum intercluster gap. For file 4863 our method resulted in values of 3.09 for the maximum intracluster gap and 10.5 for the minimum intercluster gap.

Once the data are partitioned in this way, we can fit a distribution to the time between the start of each cluster to form the first process. Then, within all the clusters, we can match a distribution to the time between each event to represent the second level of the process. We assume the same process is generating intracluster arrivals for all clusters. For file trace data this is reasonable as such arrivals are probably generated within inner loops of an application. 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 are composed of nearly 8,000 such objects. 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 allowable intracluster gap) yield good results over the entire heterogeneous data set; that is, a value that is an ordinary intracluster gap for one file may represent an intercluster gap for another.

1.2 Clustering Approach

In this article, 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 for other applications, it has ideal characteristics for file access data, as shown in Section 4. 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 4.1 introduces a new stopping rule designed specifically for this application.

The methodology we propose uses the Johnson translation system [Johnson 1949] to fit a distribution to the intercluster times, intracluster times, and the cluster sizes. Each of these is treated as an independent univariate distribution if clustering successfully identifies a two-level process. The Johnson translation system is a natural choice given our need for a robust model: within this single system of distributions exists a unique distribution that matches any finite first four moments (equivalently sample mean, variance, skewness, and kurtosis) that can occur in a data set. This property offers much more flexibility than standard one- and two-parameter families, such as the exponential, gamma, or lognormal, and there exists software support for fitting Johnson distributions. In addition, random-variate generation from the Johnson system is straightforward.

The effectiveness of this methodology is evaluated by comparing the output of a 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.

1.3 Road-Map of the Article

The next section reviews related work. Section 3 gives some background on the replicated file system application that provided the original motivation for this research and also serves as the validation test of the input modeling methodology. Section 4 details the automatic clustering technique and presents the dynamic stopping rule. Then Section 5 presents our

methods for modeling the input trace as a two-level process and for generating synthetic trace data from the model. Section 6 presents our experimental validation of the synthetic trace generation methods. Finally, Section 7 summarizes and concludes the article.

2. RELATED WORK AND ALTERNATIVE APPROACHES

Much of the previous work on generating synthetic file reference traces has been done with the purpose of stress testing or performance comparison of file system servers. The load as seen by a file server is quite unlike the load generated by users; the caching at client machines filters out most of the locality of reference that is typically present in the users' access patterns. A file that a user accesses once is likely to be accessed again, but the server will only see the first reference, with subsequent references served out of the client cache [Willick et al. 1993; Froese and Bunt 1996]. Hence the kind of burstiness in the per-file trace that motivates this work does not appear in the load presented to file servers.

An example of a synthetic workload generator for file servers is found in Bodnarchuk and Bunt [1991]. In this work, Bodnarchuk and Bunt have monitored a network using a passive tap in promiscuous mode to gather a trace of network file system (NFS) operations on their network. They pick the NFS operation type according to the fraction observed in their trace data, pick the designated file system from a discrete distribution, and then pick the designated file at random, choose a data size (for read and write requests) by sampling from three uniform distributions, and select an interarrival time using a hyperexponential distribution. A tool was then built that generated NFS operations in real-time.

Much of the modeling of distributed filing and database systems assumes arrivals are generated by a Poisson process for purposes of analytic tractability (see, e.g., Singhal [1990]). Since the load at a server is the superposition of the workloads generated by a large number of (perhaps) independent loads, the assumption may be justified (although most measurements show excessive variance). However, when the goal is to model overall system load (where the probe is placed between the user and the workstation and not between the workstation and the server), the assumptions are not warranted and successive accesses are indeed highly correlated, giving rise to this research.

This article addresses the situation where the input modeler is in possession of trace data and wishes to construct a synthetic "equivalent." In other situations, one may not have access to traces, but instead have some idea of how users of a file system will behave. This latter case is addressed by the SynRGen File Reference Generator by Ebling and Satyanarayanan [1994]. Their primary goal is to develop synthetic workloads to stress the Coda File System [Satyanarayanan et al. 1990] and to compare alternative UNIX file systems at their common level of abstraction, the UNIX file system API. Hence, SynRGen is appropriate for generating loads as presented to clients and not just servers. Their approach is to build micromod-

els that capture the behavior of particular classes of users. They can then instantiate user models in a very flexible manner to build a workload for an anticipated environment given only a notion of how that environment is populated with user behaviors. SynRGen is therefore a useful framework for implementing load generators; but the actual models that it uses to generate synthetic traces are left up to modelers. Their approach differs fundamentally from ours in that SynRGen models the behavior of the users, whereas this work models the behavior of the data.

Thekkath et al. [1994] propose characterizing file system workload from limited sets of trace data using an approach, borrowed from statistics, known as bootstrapping [Diaconis and Efron 1983]. They treat the trace data as samples of size N from an unknown distribution and generate new samples by selecting at random, with replacement, elements from the original trace. Thus, N^N different new samples of size N can be generated on the fly and used to drive a simulation, or a new sample of any size can be generated by continuing to resample. Assuming that the original trace is representative of the true population, they conclude that the synthetic traces will also be good representations.

Applied blindly to our data, the bootstrapping approach is equivalent to modeling the time between file access events by the empirical cdf of the observed times between events. This implicitly assumes that the interevent times are independent and identically distributed (i.i.d.) random variables, since the next interevent time is sampled from the same empirical cdf as the previous interevent time, without reference to the value of the previous interevent time. However, in much of our data there is a physical basis for the presence of clusters or bursts of file activity, which induces dependence between interevent times. Taking an extreme case, it is easy to see that the access pattern for a file that consistently experiences clusters of exactly four events, followed by a long intercluster time, then another four events, and so on, will never be well represented by an i.i.d. sequence of interevent times. (Such a pattern actually occurs in our data set, a fact that should not be surprising since many of the events are the result of the execution of deterministic programs accessing files within loops.)

A more refined bootstrapping approach applied to our data might break the trace up into segments of an intercluster time and the following cluster, and then resample these segments with replacement. This would be better than resampling interevent times independently, but would tie particular clusters to particular intercluster times, limiting the possibility of viewing new combinations not present in the data. A further refinement would be to cluster the data, as we have, then use the empirical cdfs of the intercluster times, intracluster times, and cluster sizes to resample from a two-level process. Our method, which fits distributions to these data rather than using the empirical cdfs, simply goes one step further to what statisticians call a "parametric bootstrap." The advantage of this extra step is that it smooths the data by filling in gaps that naturally appear in any finite sample, but are typically not real features of the process. In addition,

parametric families can be used to create input models when no data are available.

The two-level model used here is closely related to Jain and Routhier's [1986] "Packet Train" model for local area network traffic. A burst of packets on the "track" between a pair of nodes in a network is viewed as a group of packets that travel together, analogous to the cars in a train. Thus, one is led to decompose the arrival process into intercar gaps and intertrain gaps. If no events are observed on the track for a specified maximum allowed intercar gap (MAIG), the previous train is considered ended and the next event is deemed the locomotive of the next train.

A critical step remains: how are we to select the MAIG? In Jain and Routhier [1986], a value of 500 ms was chosen via visual inspection and experimentation. This value worked well for the network traffic data used in their work. The mean intercar gap was an order of magnitude less than 500 ms, and the mean intertrain gap was several orders of magnitude larger. Most important, the data were relatively homogeneous; the same value of MAIG worked well across all of the node pairs. However, we find the file system trace data to be far more heterogeneous. In other words, a value for MAIG that is appropriate for one file is totally inappropriate for another. Therefore, a method for automatically clustering the data with a dynamic stopping rule is required.

Recent work has shown that many types of aggregate network traffic exhibit long-range dependence (LRD), in the sense that the autocorrelation between measurements diminishes very slowly as a function of their separation (or lag) in sequence (e.g., as a hyperbolic function of the lag). Empirical examples of this phenomenon include local-area networks [Leland et al. 1994], metropolitan-area networks [Cinotti et al. 1994], wide-area networks [Paxson and Floyd 1994; Klivansky et al. 1994], and variable bit rate coded video traffic [Beran et al. 1995]. With respect to the preceding Packet Train model, LRD implies that there may exist no natural MAIG.

Statistically self-similar processes are a special case of covariance stationary LRD processes in which the autocorrelation function is undiminished by averaging nonoverlapping subsequences of measurements. A characteristic of self-similar network traces is that they appear equally bursty when viewed over a wide range of different time scales. Leland et al. [1994] have detected the presence of self-similarity (not merely LRD) in Ethernet traces. Willinger et al. [1995] show that aggregating many packet train sources whose distributions of train lengths and intertrain gaps exhibit infinite variance (e.g., using certain Pareto distributions) results in aggregate traffic that is statistically self-similar. They further show that the degree of self-similarity (the Hurst parameter) is predictable given a measure of the tail-heaviness of the component distributions. Several studies suggest that self-similar traffic models do a better job of representing LRD workloads [Adas and Mukherjee 1995; Leland et al. 1994; Paxson and Floyd 1994] than do traditional traffic models such as Poisson, batch Poisson, Markov arrival processes, and ARMA processes that do not cap-

ture the slow decay of the autocorrelation function. Queueing simulations conducted using these traditional traffic models tend to be optimistic with respect to congestion and delay as compared to simulations driven by recorded trace data [Adas and Mukherjee 1995; Narayan et al. 1995]. Simulations driven by self-similar models that capture long-range dependence provide more accurate results [Garroppo et al. 1995; Giordano et al. 1996].

Although long-range dependence had considerable effect on the maximum queue occupancies in the referenced network queueing models, this correlation at larger time scales plays no significant role in our file conflict simulation studies. The file conflict experiments are highly sensitive to the short-range dependence within individual files. The need for a good match to the bursty behavior on the smallest time scale leads to the approach taken in this article in which we identify the first level of clustering. Any long-range dependence in either the individual file traces, or the aggregate traffic, is irrelevant to our evaluation models. In fact, we do find some evidence of LRD in the aggregate trace data (all files aggregated), but (not surprisingly) little LRD present in the individual file traces.

None of the related work known to us addresses the issue of creating a synthetic workload model that matches the burstiness characteristics of a given set of trace data on a file-by-file basis. This is what is required to drive discrete event simulation studies such as the one detailed in the next section. This simulation application is both the motivation behind this workload model, and the means by which we evaluate the effectiveness of the model at capturing the burstiness in the recorded traces.

3. THE APPLICATION

Distributed file systems and database systems sometimes maintain multiple copies of a single data object. Keeping multiple replicas can improve availability (in the face of machine or network failures) and reduce access latency (by increasing the probability that there is a replica "near" where it is needed). However, when there is an update, something must be done to maintain the consistency of the different replicas. Consistency control approaches can be partitioned into *conservative* schemes, which prevent any violation of single copy semantics by performing updates atomically to all copies, and *optimistic* schemes, which take advantage of the fact that concurrent update is rare. Optimistic approaches can have superior availability and much reduced update latency. However, they allow the occasional occurrence of a conflicting update. That is, it is possible for two different replicas to be updated concurrently before either update has had the opportunity to propagate to the other replicas. If this occurs, no replica contains the unique "most recent data" and a *conflict* is said to have occurred.

The impetus for our work is 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

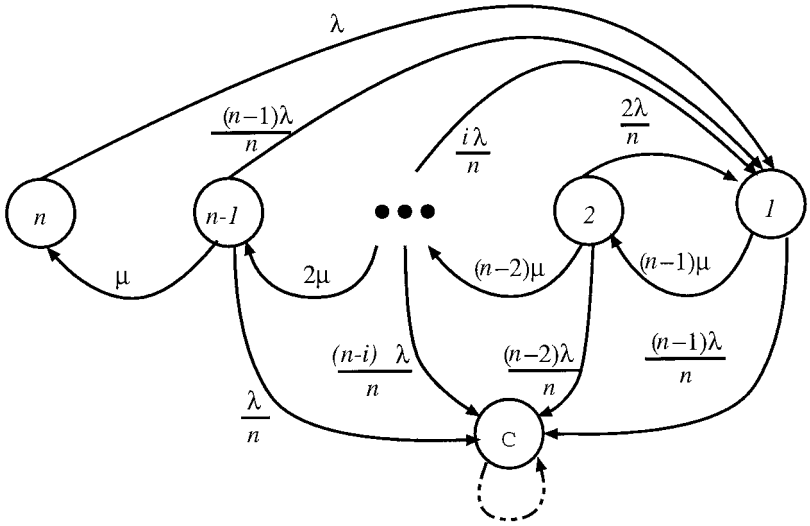

Fig. 4. State transition diagram for an n-replica file. Note that all transitions to states 1 and C are triggered by arrival events (writes to the data). All other transitions are triggered by update propagation events.

this principle (see, e.g., Bayou from Xerox PARC [Terry et al. 1995], Coda from CMU [Satyanarayanan et al. 1990], and Ficus from UCLA [Guy et al. 1990; Page et al. 1998]). 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. Since the focus of this article is the input model, rather than the file system model, we present a very simple optimistic replication model here and use it to test the effectiveness of the synthetic trace generation scheme. In a full-scale study of optimistic replication we will use a more detailed file system model.

We model a replicated data object with n copies as a finite state machine with n "normal" states and one "conflict" state (see Figure 4). 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, and so on, 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 contain the most up-to-date data, whereas $n - i$ contain stale data.

If an update arrives while the object is in any state i: $i < n$, two possibilities result: the update is applied either to one of the replicas that

already contains the latest version, or to one that does not.[1] 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,[2] 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. For purposes of this article, state C is an absorbing state (we ignore conflict repair here). A statistic of interest is then the expected time to absorption from a start state of n.

Consider the following assumptions: initial updates arrive according to a Poisson process with rate λ; any one of the replicas is equally likely to be chosen for the initial update; and the time to propagate this update to any other individual site is exponentially distributed with mean $1/\mu$. Under these conditions the model becomes Markovian and is easily analyzed 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. If we relax the assumption of a Poisson arrival process and drive a simulation of this model with trace data, then we obtain a model that is sensitive to the burstiness in the trace. Comparing the resulting behavior of this trace-driven model to a model driven by a synthetic input process provides a good basis on which to judge whether a synthetic trace reflects this critical characteristic of the actual file access patterns. Section 6 details the use of such a simulation to validate our method for analyzing and generating trace data.

This section has described a replicated file system model that is driven either via file access traces or a synthetic workload input model. This model serves the dual role of motivating the synthetic trace generation work, and evaluating its success. The next section details our method for automatic analysis of the file system traces.

4. CLUSTERING

The hypothesis is that file access data (and other types of bursty workload) are effectively modeled by two levels of processes—the first generating bursts, and the second generating events within bursts. In order to model these processes for a given data set, it is necessary to "cluster" the data into

[1]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.
[2]Where "correct" is defined as one-copy-serializable.

bursts. We consider a form of agglomerative hierarchical clustering, as described in Jain [1991], but simplified by the fact that our application requires clustering in the time dimension only.

The data are 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 $T_1, T_2, \ldots, T_n$ in a cluster by itself and initialize the iteration counter i to 0.
(2) Construct the intercluster distance array in which the jth 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 kth distance is chosen. Combine clusters k and $k + 1$.
(4) Increment i.
(5) Repeat Steps 2 to 4 until all events are part of a single cluster (we examine earlier termination conditions in the following).

The preceding 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 a two-level model is a good representation of the data set, then in between these degenerate cases lie 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 that 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 and 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.

4.1 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 ith iteration of the clustering algo-

rithm; it is nondecreasing in i as the distance is always positive. Thus, the slope of the H_i curve is always nonnegative.

Intuitively, if the data are truly bursty, the early elements of the H_i sequence should be relatively small since we are combining clusters that 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 that are separated by an intercluster gap that should be very much larger than the greatest intracluster 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 and Cooper [1985] describe and test 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 form 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 (for a moment) that no clusters exist in the file-access process. Such a process is often well modeled as a Poisson process with arrival rate λ events per unit time. In other words, the interevent 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 interevent 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

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

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.

Of course, we anticipate that there are clusters and the process is not Poisson. However, until that point at which we begin combining clusters that represent distinct bursts, the interarrival gaps should be Poisson-like. Thus, our stopping rule ceases combining clusters when the behavior of H_i departs significantly from what is expected for a Poisson process. At that point we assume that we are no longer combining intracluster event times, and have started to combine clusters. Using the second derivative test avoids the need to specify a value of λ. Based on empirical results, we chose to terminate clustering when the estimated second derivative exceeded 2.

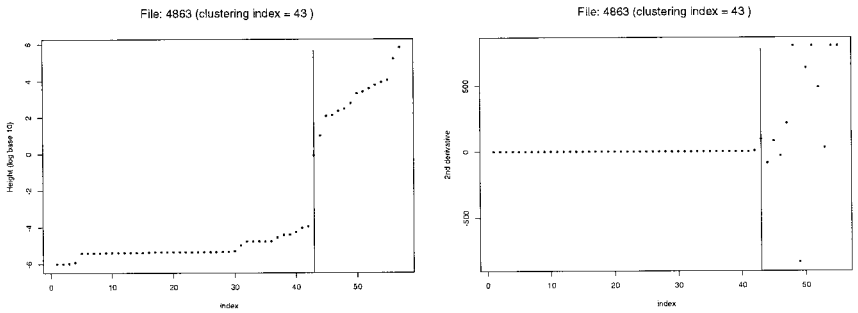

Fig. 5. Height curve and its second derivative for file 4863.

The resulting clusters were not very sensitive over a range from 2 to 10 adding only several iterations to the preceding algorithm.

However, since our traces are very heterogeneous, we may have file access traces without clusters. If we apply the stopping 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 interarrival times within one large cluster are fitted with a degenerate distribution.

4.2 Clustering Example

Consider again the trace for accesses to file 4863, as displayed in Figures 1 and 3. The clustering algorithm is initialized by defining each event to be in a cluster by itself and the list of intercluster times is built and sorted in ascending order. The algorithm then begins by combining the two closest events into one cluster. In this case the minimum interevent time is 0.001 seconds. The distance between this newly created cluster and its nearest neighbor on either side is created and the ordered list of intercluster times is updated appropriately. The algorithm continues combining the nearest neighbor clusters.

Figure 5 shows the value of H_i for each iteration of the clustering algorithm for this file. The value of H_i is very small for small values of i, reflecting the likelihood that the early steps are combining events that are indeed part of the same burst. The curve is monotonically increasing with a slope very nearly zero until it "takes off" for values of i in the mid-40s. We interpret this knee in the curve as the point where we cease combining events that are part of the same cluster and start combining separate clusters. It is here that we wish to terminate clustering.

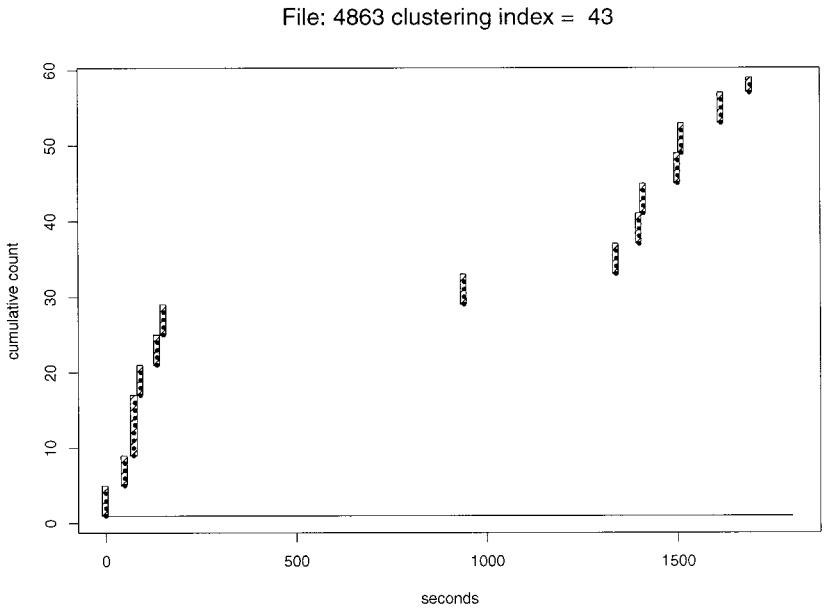

Fig. 6. Clusters automatically detected for file 4863.

Figure 5 also shows the value of the second derivative of the height curve. We can see that it is nearly zero until the mid-40s at which point it begins to jump. Our stopping rule detects that the second derivative has exceeded the threshold (chosen to be 2 for this set of data) and terminates clustering after 43 iterations. Notice that the first point at which the second derivative differs significantly from zero is precisely the point where the height curve turns sharply upward and hence successfully identifies that point at which the clustering algorithm has finished combining events within clusters and is about to combine distinct bursts. The resulting clustering is illustrated in Figure 6.

5. MODELING AFTER CLUSTERING

Now that we have partitioned the interarrival times for each file into inter- and intracluster times, we can proceed to model the processes that generated the clusters, and events within the clusters. For each separate file accessed during the trace, let the random variables X, Y, and D denote the intercluster time, the intracluster time, and the cluster size, respectively. Note that D is discrete. There are then these steps in modeling and generating data after the sample data are clustered:

(1) *Data Analysis.* Extract the observed intercluster times, intracluster times, and cluster sizes, as represented by the random variables $\{\hat{X}_1, \ldots, \hat{X}_n\}$ ($\hat{X}_1$ is the time to the first cluster), $\{\hat{Y}_1, \ldots, \hat{Y}_m\}$, and $\{\hat{D}_1, \ldots, \hat{D}_n\}$, respectively. Here n is the number of clusters and $m = \Sigma_{i=1}^{n} \hat{D}_i - n$.

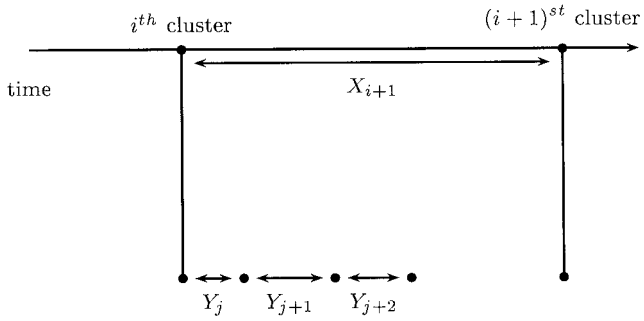

Fig. 7. A two-level process: the ith process generates $D_i = 4$ events with the time between events given by $Y_j, \ldots, Y_{j+2}$; the $i + 1$st cluster is shown starting X_{i+1} time units later.

(2) *Modeling.* Fit distributions to the observed intercluster times, intracluster times, and cluster-size data. Our software uses the Fittr1 package [Swain et al. 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. (Although D is discrete, we fit a continuous distribution due to the lack of flexible families of discrete distributions, 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 intercluster gap interval X_i, a new process is started. This process generates D_i events with the time between events given by $Y_j, \ldots, Y_{j+D_i}-2$, and then terminates. Note that for the ith cluster, $j = 1 + \Sigma_{k=1}^{i-1} (D_k - 2)$ and if $D_i = 1$ (a singleton), then it does not contribute to Y.

Figure 7 illustrates this two-level process. Figure 8 shows data generated for file 4863. The synthetic trace is visually similar to the pattern found in the original data shown in Figure 1.

6. VALIDATION EXPERIMENT

The goal of input modeling is to find a representation of the uncontrollable process that captures the features that are most critical for the application at hand. In our context, the file access patterns generated by the input model must produce (very rare) file update conflicts in the same manner as the actual file access data in the trace. Thus, we emphasize validation of the input model in terms of the output of the simulation it drives, rather than in terms of a feature-by-feature match with the original trace data.

But why not expect the input model to match the trace data across all measurable features? Input modeling of real processes—processes for which there is no "true" input model to find—always requires tradeoffs. Ideally, these tradeoffs are made by a human analyst in conjunction with appropriate statistical tools. In our application, however, the large number

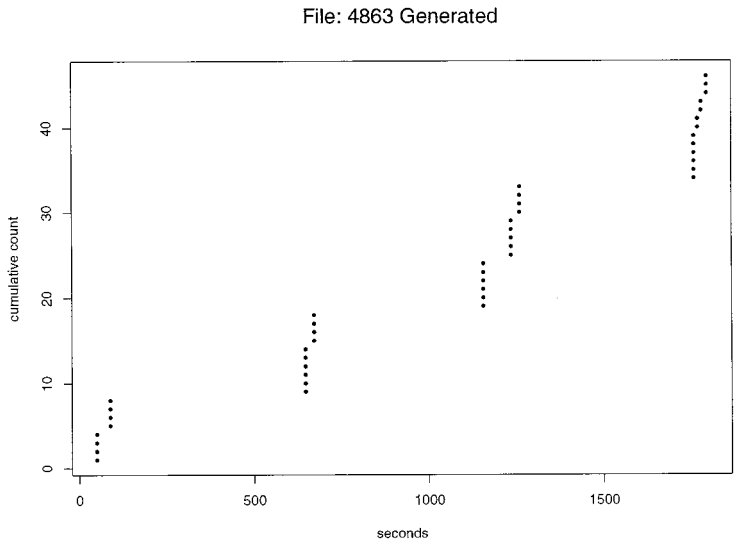

Fig. 8. Generated data corresponding to file 4863 (see Figure 1).

of input models that are needed demands an algorithmic approach that can be executed in a production mode. Any algorithmic approach will have an Achilles' heel in the form of a pathological or contrived data set that will fool the algorithm. To demand foolproof performance of an input-modeling algorithm is unreasonable; but to demand robust performance is essential. Robust performance is secured by building the input model from flexible components, structured to account for known features of the process. Our two-level representation includes as extreme cases completely random (Poisson-like) processes and completely regular (deterministic) processes, as well as the clustered processes we typically expect to see. And the Johnson translation system, which is used to represent intra- and inter-cluster times, is a very flexible, four-parameter family of distributions that can adopt any finite first four moments presented by the data. Robust performance is verified via extensive validation against a large heterogeneous collection of data sets, as described in this section.

Another, more subtle, weakness of any algorithmic approach is that a more refined input model might be obtained for each data set by a human analyst tailoring a model to the specific characteristics of the data. However, the time required to develop individual input models for thousands of files is prohibitive, and the end result is a collection of finely tuned, but unrelated models that may not be easily altered to represent different load levels or mixes of file types than those present in the data. A desirable property for our applications is the ability to easily represent expanding file systems, or file systems that do not yet exist. This capability is provided by the generic input model proposed here.

The structure of our two-level model, and the use of the Johnson system, implies that we are restricting ourselves to at most two levels, and we cannot model the types of LRD that have been observed in some aggregate

network traces. Using the Johnson translation system restricts us to distributions with finite first four moments, and therefore we will never realize inter- or intracluster distributions with the type of heavy tails that are associated with LRD for the aggregate process. However, as a practical matter any finite-length trace will have finite sample moments, so the Johnson system is capable of modeling whatever actually occurred, with moments that are arbitrarily large, but not infinite. Because we are interested in burstiness on the smallest time scale, the existence of additional levels of clustering at longer time scales is not relevant to our applications. Furthermore, we do not want to aggregate arrivals in different time scales (we need the individual file accesses), so the self-similar behavior that certain LRD processes exhibit is not required.

6.1 Method

The discrete event simulation model of a replicated file system that was described in Section 3 served as the testbed for our validation. Correct execution of the simulation program was verified by comparing simulation results against the analytical solution obtained when the process is Markov.

Trace data for the 567 files that were accessed more than 30 times during the busiest two-hour period were extracted from the complete trace. For each of these files, the clustering algorithm was run and distributions fit for the three random variables that characterized the two-level arrival-counting process. These distributions were used to generate synthetic workloads for each file.

Simulation experiments were performed for each file: the first used file accesses from the actual trace data, and the second used accesses generated by the corresponding synthetic workload model. Each experiment consisted of simulations at seven update rates, μ (the rate at which updates are propagated from the most up-to-date replica to other replicas, with the actual time to update being exponentially distributed with mean $1/\mu$) ranging from 0.001 to 1,000. The number of replicas was fixed at 5, and the replica to receive the initial update was equally likely to be any of the 5.

Each simulation run was terminated when the width of the 95% confidence interval for the mean time to conflict (the mean time to absorption, where conflicts are considered an absorbing state) was less than 5% of the mean; simulations were also terminated when five million seconds of simulated time had elapsed since the last conflict. Since the trace records were not sufficiently long to obtain the desired precision with a single pass, we treated each trace as a loop and cycled through it until the termination criterion was met. The experiment required running over 8,500 independent simulations, each of which took from several seconds to several hours to complete (the average was five minutes) requiring over 32 CPU-days on an HP9000/64 workstation. The independent simulations were distributed to over 250 idle workstations, allowing a run to complete in one night.

The data that result from each pair of experiments consist of values for the estimated conflict rate (1/estimated mean time to conflict) for each of

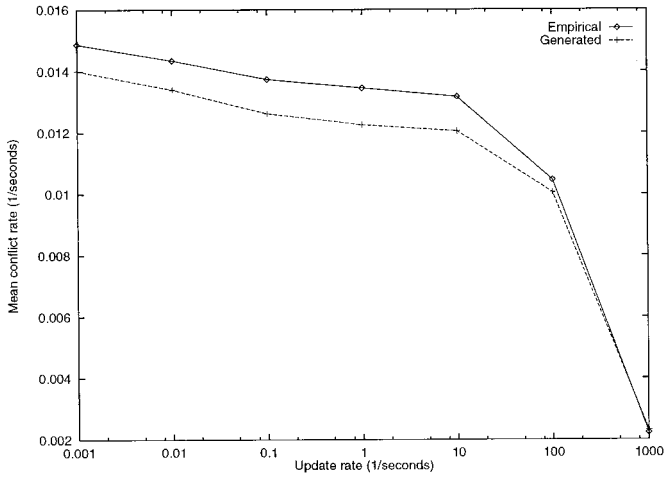

Fig. 9. Comparison of conflict rate for the original and generated data for file 4863. Number of replicas is 5.

the seven values of μ, and for each file. The first element of the pair is produced with the empirical trace data, and the second with the synthetic data. This results in a pair of curves for each file as a function of the update rate μ. One such pair of curves, for our example file 4863, is shown in Figure 9, and is discussed in the next section. In a later section we present summary statistics across all files.

6.2 Single File Illustration

Figure 8 shows the results of generating file access times for file 4863. Comparing this to Figure 1 suggests similar patterns of clustering. This file turns out to be one of the most difficult for our methods to match. Most of the intracluster gaps are very small (hundredths of a second), yet two are around three seconds. Because the data set is so small, these two larger gaps have considerable effect on the mean intracluster gap and cause, for this file, the lower conflict rate. However, visually inspecting the results of clustering shows that the automatic method did almost exactly what a human would have done. The percentile values for intracluster times also show the match to be quite good.

Figure 9 shows the results of running the simulation with both the empirical trace and synthetic data for this file. We plot the conflict rate, 1/(mean time to conflict), because the mean time to conflict approaches infinity for large values of update rate. The conflict rates, as a function of update rate, are very close in absolute value for both data sets, lending support to the hypothesis that the synthetic data capture important characteristics of the empirical trace.

This experiment looks at the change in conflict rate as the update propagation rate μ increases. For a conflict to occur in this model, two things must happen: an operation must follow a previous operation so

Table I. Summary Statistics for the Trace Validation Experiment

Propagation rate	0.001	0.01	0.1	1	10	100	1000
Mean Relative % Error	4.31	7.542	8.259	-2.497	-4.768	-1.201	0.1458
Standard Deviation	25.84	27.82	24.33	14.94	14.5	9.706	2.346
File Count	567	567	567	567	567	567	567

closely that the first update has not had time to propagate to all of the other replicas; and replica selection must choose one of the not-yet-up-to-date replicas to serve the next write. When updates are propagated slowly, conflicts occur frequently, since it is often the case that a not-yet-up-to-date replica is selected for the next update. As updates propagate more quickly, conflicts become rarer. Beyond some update rate no more conflicts are observed within the limits of the available simulation time. The dependence on update rate is clearly illustrated in Figure 9. This particular file shows a fairly high conflict rate over the range of update propagation rates tested due to the very small intracluster gaps; in other words, it is not unusual for this file to experience back-to-back writes before update propagation has spread the first write to all five replicas.

6.3 Experimental Results

Table I and Figures 10 and 11 summarize the results of running the simulation experiment for all 567 files. The "mean relative error" reported in this table for a given value of μ is calculated as follows.

(1) For each file, compute the difference between the conflict rate obtained using synthetic data and the conflict rate obtained using the trace data at each value of μ.
(2) For each file, divide the absolute difference in conflict rate for each update rate μ by the conflict rate obtained using the empirical trace at the lowest update rate, $\mu = 0.001$.
(3) Average these relative differences across all 567 files at each value of μ.

Thus, the "relative error" is relative to the smallest update rate, which implies the largest conflict rate, for each file using the trace data. We chose this standardization because for many files the conflict rate approaches zero for large update rates, causing the relative error to approach infinity even when the absolute error is quite small. By standardizing on the largest conflict rate we appropriately discount differences between the simulations when the conflict rates are very, very small, and therefore inconsequential.

Table I provides compelling evidence that the automated synthetic load generator is representing the critical characteristics of the trace data as they relate to conflict rate. The mean relative error across all files is in the single digits, as a percent, and 50% of all files are within a 10% relative error across all update rates, even though the models were generated entirely without human tuning.

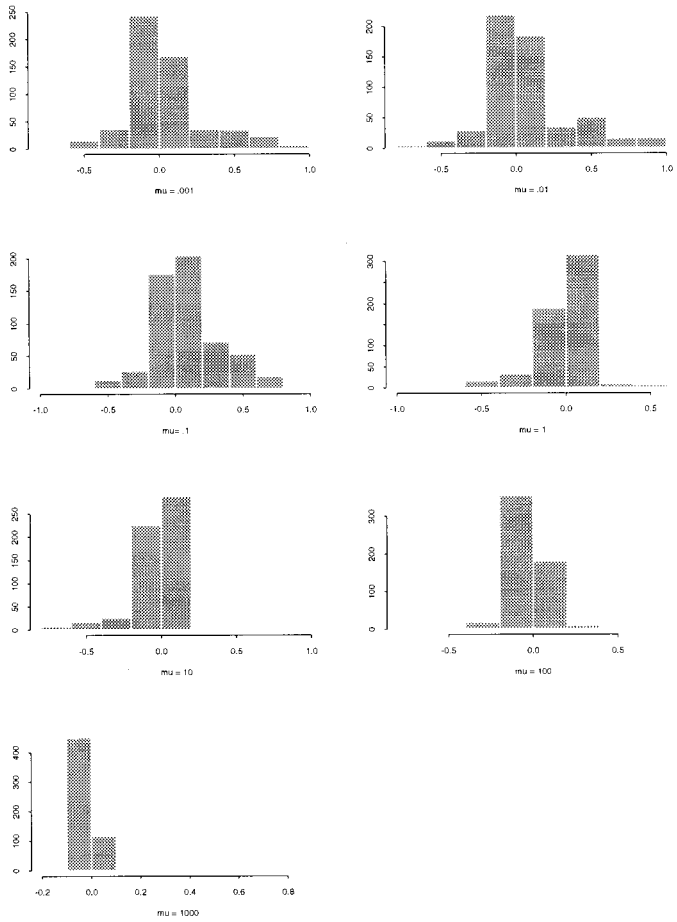

Fig. 10. Summary of relative error for various values of propagation delay.

One can also see that there is a systematic difference between the results obtained for each type of data: relative to the empirical trace data, the synthetic data tend to cause overestimation of the conflict rate for small update propagation rates, and underestimation of the conflict rate for large update propagation rates. One possible explanation for this is the wrap-around effect using empirical, and therefore finite, trace data. At large update propagation rates, the simulation requires excessively long runs to obtain a conflict, and the time to conflict is highly variable, meaning that many conflicts must be observed to achieve a sufficiently narrow confidence interval for the mean. Although the synthetic load generator is able to continue producing statistically new loads for these long runs, the empirical trace must be repeatedly rewound and run again. Thus, the effect of cutting off the last gap and joining the first gap in progress is repeated with each iteration, amplifying its impact. This rewind effect, and its

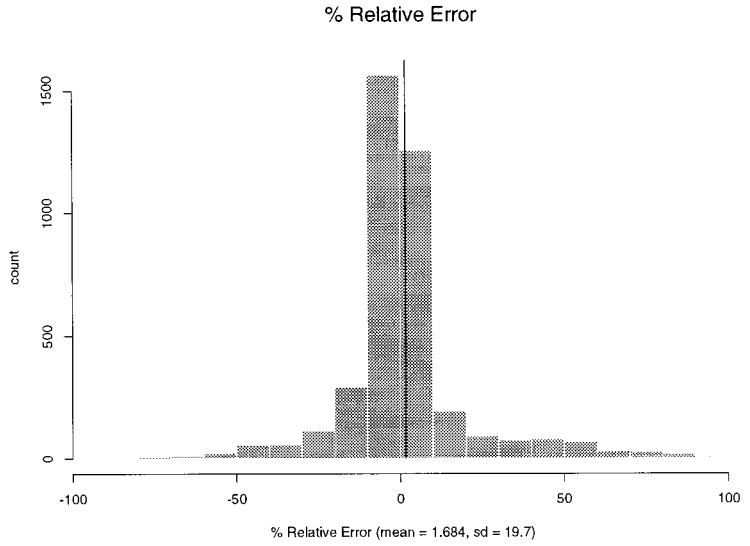

Fig. 11. Summary of relative error for all values of propagation delay.

distortion of the simulation results, is one of the reasons for preferring a synthetic load generator in the first place.

7. CONCLUSIONS

This article 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, (2) the processes are one- or two-level processes, and (3) the need is for individual file rather than aggregate trace accuracy. Trace data from both computer file systems and from communications networks typically fit these characteristics. In the case of file system traces, the data are decomposed into separate traces for each file accessed; for network traffic, communications between distinct pairs of source and destination sites would 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 arrivals into bursts. A novel stopping rule is presented that allows the method to use different clustering thresholds for each file. This is in contrast to previous work (see, e.g., Jain and Routhier [1986]) which required a single threshold, determined by inspection, to be applied to the whole data set.

The critical test for a synthetic trace generation facility is how well it captures the characteristics of the load it is intended to model. In order to assess the trace analysis and generation software, 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 model is very sensitive to the burstiness of the update traffic.

Hence it is a good test of the degree to which the method captures this characteristic of the trace data. The results of the comparison experiment lead us to conclude that the two-level approach presented here is successful in this respect.

It should be noted that no attempt was made to model the time-varying nature of the workload. It is well known that file system workloads are nonstationary. 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. Implicit in this decision is the assumption that the trace is stationary over the two-hour busy period chosen. Hence, no work is planned towards addressing nonstationary workloads.

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

In the method described here, we model the intercluster, intracluster, and cluster-size processes. A result of this approach is the potential to create overlapping clusters in the synthetic trace. Referring to Figure 7, if $X_{i+1} < \Sigma_{h=j}^{j+D_i} Y_h$ (where $j = 1 + \Sigma_{k=1}^{i-1} (D_k - 2)$ as before), then the ith and $i + 1$st clusters overlap, as the first event of cluster $i + 1$ occurs before the last event of cluster i. The result is a synthetic trace which, if reclustered, would appear to have slightly larger than predicted clusters and shorter than predicted intracluster gaps. This might result in more conflicts (shorter mean time to conflict) in the validation experiment. Since we do not see any tendency towards higher conflict rates with the synthetic traces, we conclude that this effect is not large; at least not the largest source of error present.

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 that 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 that mimics an existing one, but has 10 times as many users accessing 5 times as many files. One technique for accomplishing such a task is to sample from the set of individual file models derived from a trace, as described here. The work presented is a step along the path towards producing such a tool.

The results presented here should not be used to draw conclusions about the likelihood of conflicts with optimistic replication algorithms. The version of the file system model employed here is oversimplified, and used only as motivation and as a metric to judge the synthetic trace generation. In particular, the assumption in the file system model that the choice of the site to be updated is random is clearly unfounded and tends to greatly overestimate the occurrence of conflicts.

ACKNOWLEDGMENTS

The authors wish to acknowledge our indebtedness to the anonymous reviewers whose patience and careful attention have improved this article. We also thank Andy Hisgen of DEC SRC for making the file system trace data available to us. S. Owicki, B. Kumar, J. Gettys, and D. Hwang also contributed to the collection of the file system traces. Finally we thank Marne Cario who was helpful in early stages of this work.

REFERENCES

ADAS, A. AND MUKHERJEE, A. 1995. On resource management and QoS guarantees for long range dependent traffic. In *Proceedings of the Fourteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'95)* (Los Alamitos, CA, June), 779–787.

BERAN, J., SHERMAN, R., TAQQU, M., AND WILLINGER, W. 1995. Long-range dependence in variable bit rate video traffic. *IEEE Trans. Commun. 43,* 2–4, 1566–1579.

BODNARCHUK, R. R. AND BUNT, R. B. 1991. A synthetic workload model for a distributed systems file server. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems* (May), ACM SIGMETRICS, 50–59.

CINOTTI, M., MESE, E. D., GIORDANO, S., AND RUSSO, F. 1994. Long-range dependence in Ethernet traffic offered to interconnected DQDB MANs. In *Proceedings of the IEEE ICCS'94 Singapore,* 14–18.

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

DIACONIS, P. AND EFRON, B. 1983. Computer-intensive methods in statistics. *Sci. Am. 248,* 5 (May).

EBLING, M. R. AND SATYANARAYANAN, M. 1994. SynRGen: An extensible file reference generator. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems* (May), ACM SIGMETRICS.

FROESE, K. W. AND BUNT, R. B. 1996. The effect of client caching on file server workloads. In *Proceedings of the 29th Hawaii International Conference of System Sciences* (Jan.).

GARROPPO, R., GIORDANO, S., PAGANO, M., AND RUSSO, F. 1995. Chaotic maps generation of broadband traffic. In *Proceedings of the IEEE MICC'95* (Nov.).

GIORDANO, S., PAGANO, M., PANNOCCHIA, R., AND RUSSO, F. 1996. A new call admission control scheme based on the self-similar nature of multimedia traffic. In *Proceedings of the IEEE ICC'96* (Dallas, TX, June).

GUY, R. G., HEIDEMANN, J. S., MAK, W., PAGE, T. W., JR., POPEK, G. J., AND ROTHMEIER, D. 1990. Implementation of the Ficus replicated file system. In *USENIX Conference Proceedings* (June), 63–71.

HISGEN, A. 1990. DEC firefly trace data. Data obtained from author on 8 mm tape.

JAIN, R. 1991. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling.* Wiley, New York, NY.

JAIN, R. AND ROUTHIER, S. A. 1986. Packet trains—measurements and a new model for computer network traffic. *IEEE J. Selected Areas Commun. SAC-4,* 6 (Sept.), 986–995.

JOHNSON, N. L. 1949. Systems of frequency curves generated by methods of translation. *Biometrika 36,* 149–176.

KAUFMAN, L. AND ROUSSEEUW, J. 1990. *Finding Groups in Data: An Introduction to Cluster Analysis.* Wiley, New York, NY.

KLIVANSKY, S., MUKHERJEE, A., AND SONG, C. 1994. On long-range dependence in NSFNET traffic. Tech. Rep. GIT-CC-94-61 (Dec.), Georgia Institute of Technology.

LELAND, W., WILLINGER, W., TAQQU, M., AND WILSON, D. 1994. On the self-similar nature of Ethernet traffic (extended version). *IEEE Trans. Netw. 2,* 1 (Feb.).

MILLIGAN, G. W. AND COOPER, M. C. 1985. An examination of procedures for determining the number of clusters in a data set. *Psychometrika 50,* 2, 159–179.

NARAYAN, O., ERRAMILLI, A., WILLINGER, W., LAKSHMAN, T. V., HEYMAN, D., MUKHERJEE, A., AND LI, S. 1995. Performance impacts of self-similarity in traffic. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems* (New York, NY, May), 265–266.

ORD, J. K. 1972. *Families of Frequency Distributions.* Griffin, London.

PAGE, T., GUY, R., POPEK, G., HEIDEMANN, J., AND KUENNING, G. 1998. Perspectives on optimistic peer-to-peer distributed filing. *J. Softw. Pract. Exper. 28,* 4, 155–180.

PAXSON, V. AND FLOYD, S. 1994. Wide-area traffic: The failure of Poisson modeling. In *Proceedings of the ACM SIGCOMM'94* (Sept.), 257–268.

SATYANARAYANAN, M., KISTLER, J., KUMAR, P., OKASAKI, M., SIEGEL, E., AND STEERE, D. C. 1990. Coda: A highly available file system for a distributed workstation environment. *IEEE Trans. Comput. 39,* 4 (April), 447–459.

SINGHAL, M. 1990. Update transport: A new technique for update synchronization in replicated database systems. *IEEE Trans. Softw. Eng. 16,* 12 (Dec.), 1325–1336.

SWAIN, J. J., VENKATRAMAN, S., AND WILSON, J. R. 1988. Least-squares estimation of distribution functions in Johnson's translation system. *J. Stat. Comput. Simul. 29,* 271–297.

TERRY, D. B., THEIMER, M. M., PETERSEN, K., DEMERS, A. J., SPREITZER, M. J., AND 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* (Dec.).

THEKKATH, C. A., WILKES, J., AND LAZOWSKA, E. D. 1994. Techniques for file system simulation. *Softw. Pract. Exper. 24,* 11 (Nov.).

WILLICK, D. L., EAGER, D. L., AND BUNT, R. B. 1993. Disk cache replacement policies for network fileservers. In *Proceedings of the Thirteenth International Conference on Distributed Computer Systems,* (Pittsburgh, PA, May), 2–11.

WILLINGER, W., TAQQU, M. S., SHERMAN, R., AND WILSON, D. 1995. Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic at the source level. In *Proceedings of the ACM SIGCOMM'95.*