

Cluster-Based Input/Output Trace Synthesis

Bo Hong * Tara M. Madhyastha * Bing Zhang †

* Department of Computer Engineering

† Department of Computer Science

University of California Santa Cruz

1156 High Street

Santa Cruz, CA 95064

{hongbo,tara,bing}@soe.ucsc.edu

Abstract

I/O traces are crucial for understanding the performance of new storage architectures. Unfortunately, traces are extremely bursty and difficult to characterize. They are large, difficult to obtain, and unwieldy. In this paper, we examine a method of trace synthesis based on cluster analysis of the time-varying characteristics of the trace. Representative trace segments are selected, and a synthesized trace is reconstructed from the segments. We show that we can achieve a 5–10% demerit factor for I/O response times with a reduction of data volume of 75–90%.

1 Introduction

Storage architects rely upon representative I/O workloads to improve systems performance. Unfortunately, actual I/O traces are extremely large, difficult to obtain and work with, and cannot be parameterized. Benchmarks are more tractable, but are often less realistic than actual workloads. There is therefore strong motivation to develop techniques to generate synthetic traces.

We have found that self-similarity at large time scales does not significantly affect disk behavior with respect to disk response times or queue lengths [10]. With this in mind, our approach to trace synthesis is to create a database of representative trace segments, using cluster analysis, from which synthetic workloads may be created. We measure the similarity of the reconstructed trace by observing that the distributions of queue lengths and response times resemble those created by the original workload.

The remainder of this paper is organized as follows. We describe related work in §2. We show that long-range dependence has little effect upon disk response times in §3. In §4 we describe our approach to synthetic trace sampling using cluster analysis. We evaluate our method in §5 and conclude with directions for future work in §6.

2 Related Work

Ganger described the difficulty of generating synthetic I/O workloads, showing that obvious simplifying assumptions to reproduce spatial and temporal access patterns have large error margins [6]. Because of these important correlations, we consider reduction of trace data rather than complete synthesis.

Clustering techniques have been widely used in workload characterization of batch and interactive systems, whose workload components are described by CPU time, number of I/O accesses, amount of memory

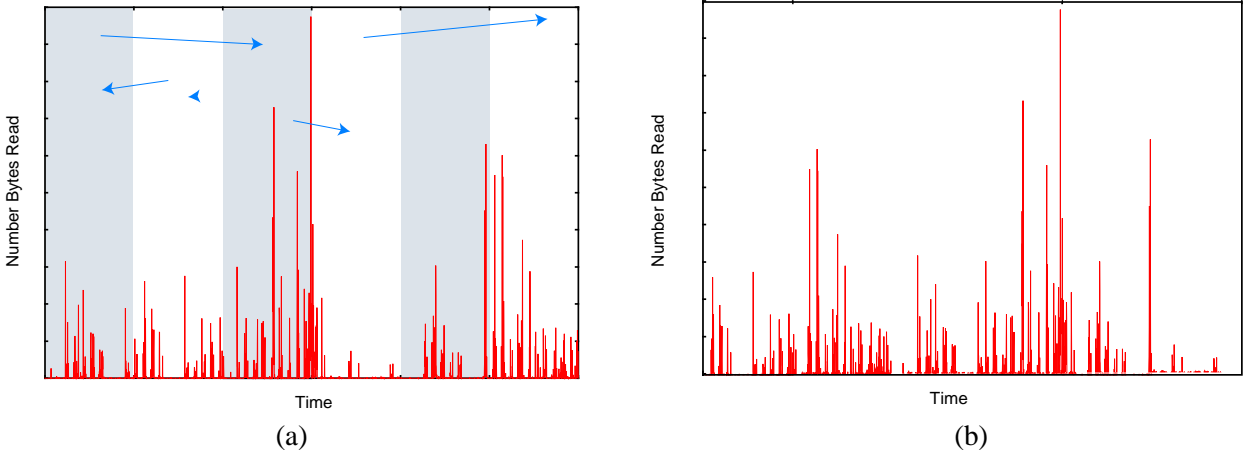


Figure 1: Shuffling traces removes long-range dependence.

used, and so on. Using these techniques, Agrawala *et al* [1] constructed a workload model for a multi-programmed computer system and Wight [21] modeled workloads from interactive systems. Serazzi [17] proposed a functional and resource-oriented procedure for workload modeling on a batch system.

Calzarossa and Serazzi [3] analyzed the fluctuations in the arrival patterns of the workload components using numerical fitting and clustering techniques and constructed a parametric model. Ware *et al* [20] proposed to use cluster analysis to separate the file access arrival process into bursts of activity and characterize it in terms of the time between bursts of activity on a file on a large, replicated file system. Cluster analysis has been used to reduce the number of processor trace data streams [14]. This approach is similar to ours, except that we are clustering offline and the goal is to identify representative time intervals instead of entire streams.

Grossglauser [7] demonstrated that it was not useful to model long-range dependence in network traffic at timescales disproportionate to the performance metrics under observation. We have shown that we can recreate disk response times and queuing behavior to reasonable accuracy by preserving long-range dependence only up to short timescales (seconds) [9, 10]. This result motivates clustering using time intervals.

3 Relevance of Long-Range Dependence in Disk Traffic

I/O workloads have a structure that researchers have proposed might help to model them called *self-similarity*. Informally, in this context, to say a time series is self-similar implies that it looks qualitatively the same at different time scales. Self-similar traffic also has the property of *long-range dependence*: the data set exhibits a slow decay in its autocorrelation function. This correlation structure is significant because self-similar traffic may be more bursty than that generated by other sources.

Our hypothesis is that previous events cannot affect disk behavior beyond a certain threshold, determined by system parameters, so modeling long range dependence at larger timescales is unnecessary. To test this hypothesis, we study how disk response time and queue lengths change as we gradually destroy long-range dependence in the traces by shuffling increasingly smaller intervals. This approach is identical to the experimental approach taken by [7], and is illustrated in Figure 1. Figure 1a shows a trace that has been divided into six intervals. These intervals are then randomly rearranged to create a new trace, as shown in Figure 1b. Within each interval, the temporal relationships are preserved, but the new trace has no long-range dependence beyond the width of the interval.

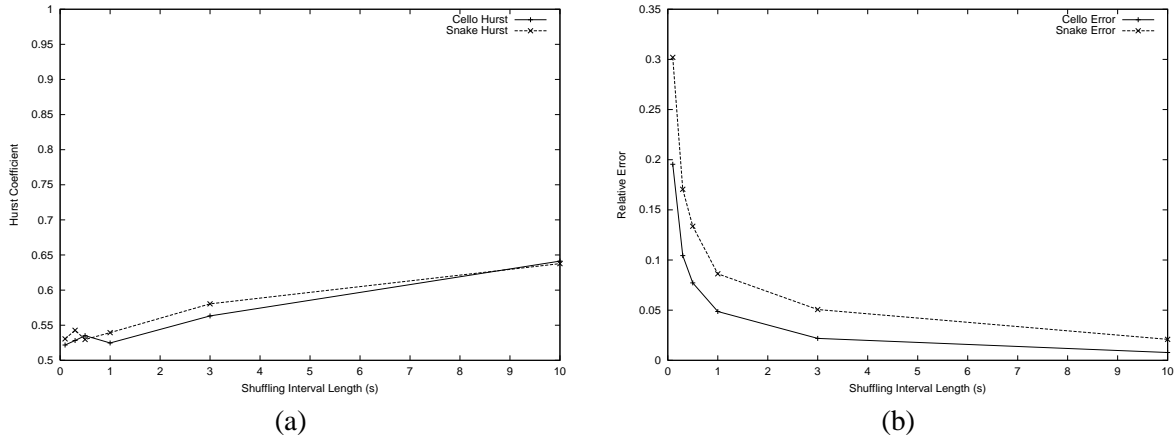


Figure 2: Effect of shuffling on (a) Hurst coefficient and (b) relative error.

We compare Hurst coefficients for traces shuffled at increasingly smaller intervals. The Hurst coefficient is a parameter used to describe the degree of self-similarity [2]. A value of H between $\frac{1}{2}$ and 1 indicates the degree of self-similarity (higher is more self-similar). We estimate H using the R/S statistic (rescaled adjusted range) [18].

Our selected workloads, described in more detail in [15], are the cello news disk traces (HP2204A) and the snake usr2 disk traces (HP97560) gathered between 05/30/92 and 06/06/92. The average I/O loads on the disks on these systems are small: approximately three requests for cello news disk and one request for snake usr2 disk per second. However, the maximum queue lengths can be very large: over 1000 requests on cello news disk and over 60 requests on snake usr2 disk. In general snake traces are more bursty than cello, and the logical sequentiality (percentage of requests that are at adjacent disk addresses or addresses spaced by the file system interleave factor) of cello and snake is 2% and 29%, respectively.

We examine the numerical metric of disk performance used in [16] to validate disk models: the root mean squared (RMS) horizontal distance between the cumulative distribution functions (CDF) of I/O response times. We vary the shuffle interval length from 10 seconds to 0.1 seconds and use both the shuffled and unshuffled traces to drive the Pantheon [22] disk simulator.

Figures 2a and 2b show the Hurst coefficient using the R/S method, and the relative error for the shuffled traces, respectively. The Hurst coefficient is a measure of long-range dependence; as intuition dictates, the smaller the time interval, the fewer long-range correlations are preserved and the lower the Hurst coefficient. For one day, $H = 0.79$ for snake and $H = 0.89$ for cello.

Despite the lack of long-range dependence, particularly indicated by the fluctuation of the Hurst coefficient at small intervals (< 1 second), the relative error for the shuffled traces is relatively small; at 1 second it is approximately 5% for cello and 9% for snake. The distributions of queue lengths of traces shuffled at intervals > 1 second are similar to those of the real traces [9], and are not presented separately.

4 Cluster-Based I/O Trace Synthesis

From §3 we know that previous events cannot affect disk behavior beyond a certain threshold; this allows us to consider short trace intervals to be independent. We believe that an I/O trace from one time interval will be similar to those from other time intervals, and that such similar trace intervals form equivalence classes (clusters). Storage system behavior should be statistically similar when servicing I/O request trace intervals from the same equivalence class.

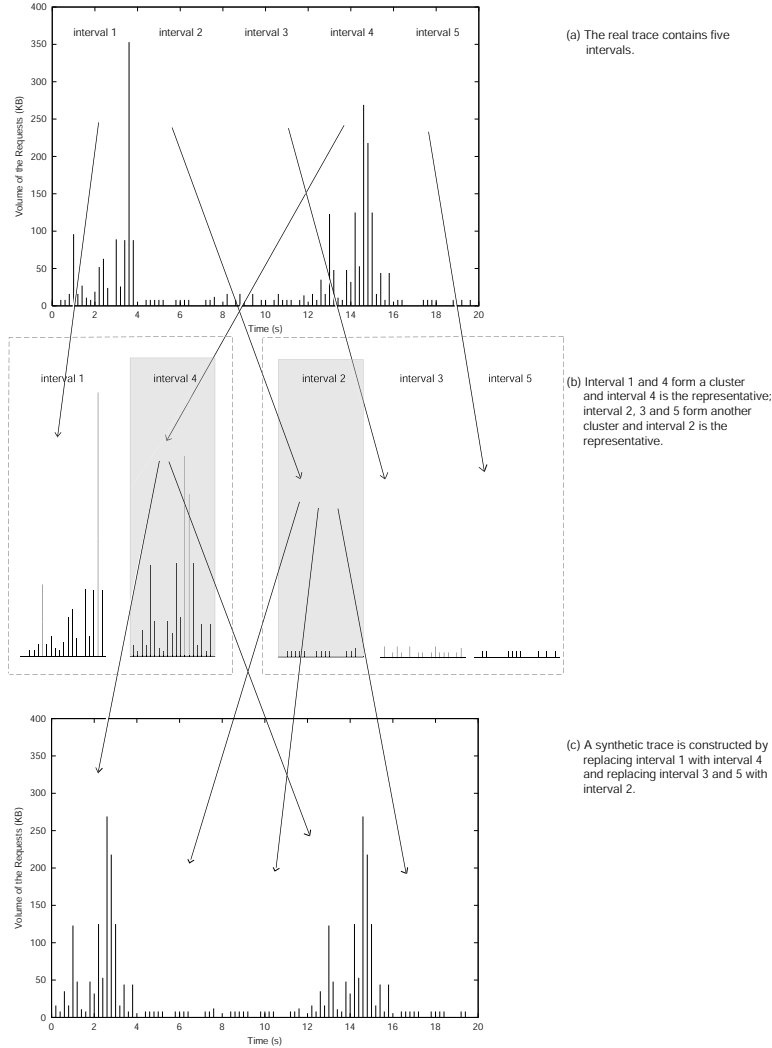


Figure 3: Replacing trace intervals with the representative interval in the same cluster to generate a synthetic trace.

This leads us to a novel method of trace synthesis: after identifying the equivalence classes among original trace intervals, we replace the intervals in the same class with the class representative to generate synthetic traces. Figure 3 is an overview of this process. Figure 3a shows a trace with five intervals (1-5). Intervals 1 and 4 form one cluster, with representative 4, and intervals 2, 3 and 5 form another cluster, with representative 2 (Figure 3b). We generate a synthetic trace by replacing interval 1 with interval 4 and intervals 3 and 5 with interval 2, as shown in Figure 3c.

4.1 Cluster Analysis

Cluster analysis is a multivariate analysis technique to classify a given set of objects (in our context an object is a trace interval). Assume each object is represented by a vector of observations $\mathbf{X}' = (X_1, X_2, \dots, X_m)$ on m variables and $\mathbf{X}_i' = (X_{i_1}, X_{i_2}, \dots, X_{i_m})$ is the measurements on the i th object. The goal of cluster analysis is to partition a given set of objects into clusters. This partition should have the following properties: (1) homogeneity within the clusters (i.e., objects that belong to the same cluster should be as similar as

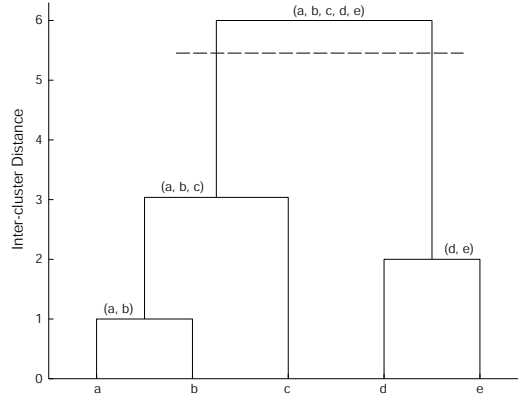


Figure 4: A dendrogram illustrating the fusion made at each stage of the hierarchical clustering analysis on the objects a, b, c, d and e. The height represents the distance between two clusters. By cutting the hierarchy, cluster (a, b, c) and cluster (d, e) are obtained.

possible) and (2) heterogeneity between clusters (i.e., objects that belong to different cluster should be as different as possible) [8]. The concept of “similarity” has to be specified according to the data, the representation of the objects.

It is of central importance in cluster analysis to measure the *similarity* or *distance* between the respective objects and clusters. We use the familiar Euclidean distance between two objects as the inter-object distance metric and the group average distance between two clusters (the average distance between all pairs of objects, one object from one cluster and one from the other) as the inter-cluster distance metric.

A vast variety of clustering techniques have been developed over the last three decades [5]. There are two popular kinds of clustering techniques: hierarchical and partitional. We focus on the *agglomerative hierarchical clustering technique* because it is one of the most widely-used clustering techniques and can provide a starting point for more complex clustering procedures [5].

The agglomerative hierarchical clustering technique proceeds in the following way: initially each object is considered to be a cluster, then iteratively, two “closest” clusters are chosen according to a specified measurement of distance and fused into a new cluster. The procedure repeats until all the objects belong to a single cluster. A two-dimensional *tree diagram* known as a *dendrogram* illustrates the fusion made at each iteration of the analysis. Figure 4 shows the procedure of applying such techniques to a set containing five objects (a – e). In hierarchical clustering, smaller clusters can be obtained by “cutting” a dendrogram. Researchers must decide “where to cut the tree” to obtain good fitting number of clusters.

4.2 Choosing a Metric Set: Theoretical Approach

We reduce each trace interval to a set of metrics that describe a point in a multi-dimensional metric space. The choice of metrics describing I/O traces is critical to our synthetic method. A badly chosen metric set will misrepresent the importance of various trace characteristics, which can lead to large synthetic errors.

Burstiness in I/O traces has huge impact on disk performance. An intuitive statistic, the aggregation ratio a , can describe local burstiness in I/O trace intervals: if we divide a trace interval into smaller time windows that may or may not contain requests, the aggregation ratio a of the interval is the ratio of the number of requests in the interval to the number of non-empty windows. The more bursty the trace, the more requests arrive in a short time and the higher the ratio. The ratio changes with the window length: we choose a window length so that each non-empty window corresponds approximately to a single burst of queued requests.

We also use a more complicated statistic, bias p , to describe the local burstiness. Wang *et al* [19] proposed to use binomial multifractals to model bursty disk traffic. Intuitively, we consider I/O requests along a timeline to be distributed according to the “80 – 20 rule”, where 80% of the requests occur within 20% of the time. The bias p determines the exact percentage, and can be estimated from the traces. Hong and Madhyastha [10] showed that the bias p can model local temporal burstiness present in I/O traces quite well.

Other potential metrics describing I/O traces are the number of requests (n), the volume of requests (v), the number of read requests (n_{read}) and the volume of read requests (v_{read}). The number of write requests and the volume of write requests are not considered explicitly because they can be calculated from other metrics.

Spatial locality is also an important characteristic of I/O traces. For simplicity, we decided not to model the spatial locality in real trace intervals for purposes of clustering. Therefore, our initial metric set is $(p, a, n, v, n_{read}, v_{read})$. Each trace interval can be represented by a vector of observations on that set.

4.2.1 Rescaling Metrics

We rescale the original metrics before cluster analysis because the metrics under study have different units and scales, preventing direct comparison. Furthermore, the statistical techniques used in our experiments do not have the property of scale invariance. Rescaling a metric has an influential effect on further data analysis because it is equivalent to giving it greater or lesser importance than other metrics when using these on statistical analysis. A common approach is to give a metric importance inversely proportional to the measure of variability in this metric [5], which implies that the importance of a metric decreases when its variability increases. Milligan and Cooper concluded that rescaling methods based on the sample range of each metric are the most effective [13].

In our research, we presume that the metrics p and a , which directly describe the local burstiness in I/O traces, are more important than other metrics. Empirically, we choose the following rescaling method:

$$\begin{aligned} X' &= X/Range(X) \\ &= X/(Max(X) - Min(X)), \end{aligned} \tag{1}$$

where X is a metric and $Max(X)$ and $Min(X)$ is its sample maximum and minimum, respectively.

4.2.2 Metric Set Reduction

The correlations between the number of requests n and the volume of requests v and between the number of read requests n_{read} and the volume of read requests v_{read} is very high (> 0.95), which means that there is strong linear relationship between n and v and between n_{read} and v_{read} . We apply principal components analysis (PCA) techniques on the rescaled data to further reduce the metric set. The basic idea of PCA techniques is to describe the variation of a set of multivariate data in terms of a set of uncorrelated variables called principal components, each of which is a linear combination of the original variables. The objective of PCA is to reduce dimensionality to simplify later analysis.

A common and relatively *ad hoc* procedure to decide the number of components retained is the following: retain just enough components to explain a large percentage (e.g. 70–90%) of the total variation of the original variables [4]. The method we use to select which of the original metrics to retain is the B2 method [11, 12]. Assume K original variables exist and we have decided to retain the first q components. Thus q variables are to be retained and $(K - q)$ variables associated with the last $(K - q)$ components are to be rejected in the following way: the K components are considered in a reverse order, starting with the last

CLUSTER-BASED-SYNTHETIC-TRACE-GENERATION

INPUT: fraction of representative trace intervals f , a real trace R .

OUTPUT: a synthetic trace S .

ALGORITHM:

- Step 1 :** Collect metrics (p, a, n, n_{read}) , as defined in §4.2, for each non-empty trace interval when scanning the real trace R .
- Step 2 :** Partition the intervals into two groups. The intervals with $p = 0.5$, fitting a uniform distribution, form group G_u and those with $p > 0.5$, fitting binomial multifractals, form group G_b .
- Step 3 :** Rescale data (p, r, n, n_{read}) to (p', r', n', n'_{read}) in G_u and G_b , using Equation 1.
- Step 4 :** Cluster rescaled data in G_b and G_u into K_b and K_u clusters, respectively, using the agglomerative hierarchical clustering technique (Euclidean distance, group average distance). We can calculate K_b and K_u from the input parameter f and the number of I/O requests in G_b and G_u . One representative interval is chosen from each cluster. The representative of the cluster containing all empty intervals is simply a empty trace interval. The number of intervals in each cluster is the weight of the representative.
- Step 5 :** Create a database of representative trace intervals. A synthetic trace S is constructed by reassembling the representative trace intervals, which are randomly picked from the database according to their weights. The timestamps of the requests in the representative interval are updated to make the global time order correct without disturbing the local temporal relationships in the representative.

Figure 5: Cluster-based I/O synthetic trace generation.

component. A variable l is associated with the component under consideration if l has the largest coefficient in the component and has not already been associated with a previously considered component.

Our selected workloads are the cello news disk traces (HP2204A) gathered in 05/30/92 and 06/06/92 and the snake usr2 disk traces (HP97560) gathered in 05/30/92, 06/04/92 and 06/06/92. We apply the PCA technique on these workloads and the results consistently suggest to retain the first 2 – 4 components because they can explain most variation ($> 90\%$) in the original metrics. We decide to retain the first four components and reject the two metrics associated with the last two components (n or v and n_{read} or v_{read}). Considering the high correlation between n and v and between n_{read} and v_{read} , we decide to reject v and v_{read} and select the following metrics to establish the characteristic metric space: (p, a, n, n_{read}) .

4.3 Cluster-Based I/O Trace Synthesis Algorithm

We propose a new algorithm for generating synthetic traces based on a real trace. Figure 5 shows the cluster-based I/O synthesis algorithm.

In Step 1, we choose the trace interval length to be 5.12 seconds and the window length to be 10 ms when calculating the bias p and the aggregation ratio a because these values were found to limit trace synthesis error in another synthesis method [10]. We calculate the bias p as follows: if the fraction of non-empty windows is $< 3\%$, we assume that the trace interval fits a uniform distribution and set $p = 0.5$; otherwise, we assume that the trace interval fits binomial multifractals and calculate p from the trace [10].

In Step 2, we partition the non-empty trace intervals into groups G_u and G_b . Intervals in G_u are those with $p = 0.5$ that fit a uniform distribution and intervals in G_b are those with $p > 0.5$ that fit binomial

multifractals. We partition the intervals before cluster analysis because bursty trace intervals (with $p > 0.5$) and uniform trace intervals (with $p = 0.5$) have inherently different performance characteristics and should be in different clusters. Splitting them early in the clustering procedure saves computation time and memory during the final analysis.

In Step 3 and 4, we choose Equation 1 as the rescaling method, Euclidean distance as the measure of inter-object distance, and group average distance as the measure of inter-cluster distance when we apply the agglomerative hierarchical clustering techniques on the selected workloads.

As discussed in §4.1, smaller clusters can be obtained by “cutting” the single hierarchy constructed by the hierarchical clustering. Choice of an appropriate number of clusters is crucial to the success of the method: if it is too large, too many representative trace intervals are stored, reducing the trace compression. If the number of clusters is too small, the error of synthetic traces is large. The number of clusters is the same as the number of representative intervals, because each cluster has only one cluster representative. The cluster representative is the trace interval whose metrics have the minimum Euclidean distance to the cluster centroid, which is the average of all data in the cluster.

For generality, we select a certain percentage of intervals, rather than an absolute number, to be the cluster representatives. Because we partition the trace intervals into groups G_b and G_u , we choose a number of representative intervals from each group proportional to the number of requests in the group. We know that the fraction of representative intervals in a group is equal to f times the ratio of the average number of requests in a interval in the group to the average number of requests in a interval in the whole trace. The ratio for the bursty group G_b has a greater impact on the trace compression than the ratio for the uniform group G_u , because on average, bursty trace intervals have more requests than uniform trace intervals (for this workload). The ratio for G_b is higher for bursty traces (i.e. the snake traces) than for less bursty traces (i.e. the cello traces).

In Step 5, we create a database including the representatives of non-empty trace intervals and one empty trace interval, which is the representative of empty trace intervals. We generate a synthetic trace by picking intervals randomly from the database according to the weights of the representatives. We can do this because the long-range dependence within the trace does not have a significant impact on performance, as discussed in §3. In our experiments, we simply replace trace intervals from the original trace with their cluster representative to construct synthetic traces. This retains more information than random selection, however, the extra synthetic error will be small when using random selection because of the irrelevance of the long-range dependence on disk performance. We retain the spatial locality within the representative and update the timestamps of the requests in the representative interval to make the global time order correct without disturbing the local temporal relationships in the representative.

4.3.1 Trace Compression

One of the advantages of cluster-based synthesis is I/O trace size compression. Rather than storing the whole trace, we only store the requests in the representative trace intervals. We measure data reduction by the *compression ratio*, or the percentage of storage we save.

Formally, we define the compression ratio cpr in Equation 2:

$$\begin{aligned} cpr &= \frac{\# \text{ of requests in the trace} - \# \text{ of requests in the representative intervals}}{\# \text{ of requests in the trace}} \\ &= 1 - \frac{\sum_{i=1}^K n_i}{N}, \end{aligned} \tag{2}$$

where N is the number of I/O requests in the trace, K is the number of representative trace intervals, and n_i is the number of I/O requests in the i th representative interval.

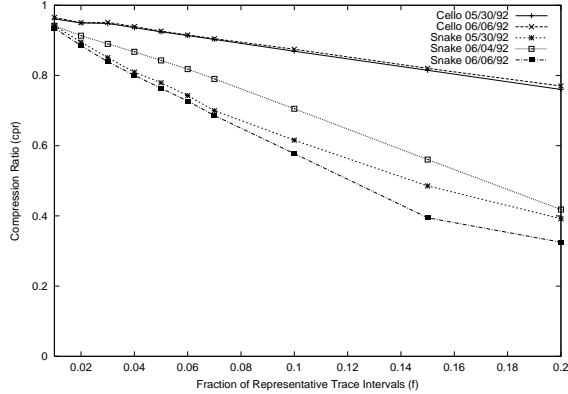


Figure 6: Compression ratio at different fractions of representative trace intervals for cello and snake.

Figure 6 shows the empirical relationship between the compression ratio cpr and the fraction of representative intervals f for different workloads. The compression ratio cpr decreases almost linearly with f . The more trace intervals we keep, the lower the compression ratio. However, the slopes of the curves of cpr and f are different for different workloads. The compression ratio of snake is lower and decreases faster than that of cello. In general, snake is more bursty than cello. As discussed in §4.3, a higher fraction of bursty intervals in G_b are chosen to be representative intervals in snake than in cello. Because on average, a uniform trace interval has fewer requests than a bursty trace interval, we achieve a higher compression ratio with the same fraction of representative intervals for cello than snake.

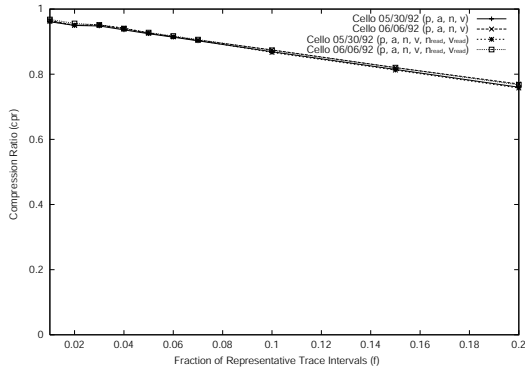
5 Simulation Results

In this section we evaluate our selection of metrics for the cluster-based trace synthesis method and analyze the accuracy of the method. We use the Pantheon [22] simulator to compare the CDF of disk response times for the original and synthetic traces. Our selected workloads are the cello news disk traces (HP2204A) gathered in 05/30/92 and 06/06/92 and the snake usr2 disk traces (HP97560) gathered in 05/30/92, 06/04/92 and 06/06/92. Hong and Madhyastha [10] used the same data set except the snake 06/04/92 trace to generate synthetic I/O workloads based on binomial multifractals.

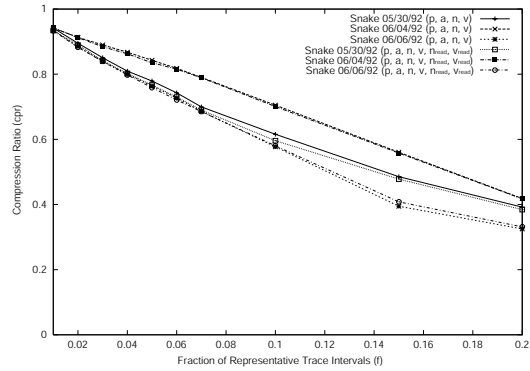
5.1 Choosing a Metric Set: Empirical Approach

In §4.2 we discussed the problem of metric selection and identified the metric set (p, a, n, n_{read}) , which is a subset of the initial metric set $(p, a, n, v, n_{read}, v_{read})$. We can verify this result empirically by comparing the compression and synthesis capability of our method using different metric sets. Our metrics are the compression ratio and the relative error of the CDF of disk response times for the original and synthetic traces. For simplicity, we refer the method using a given metric set V as the V method.

We found empirically that the abbreviated metric set (p, a, n, n_{read}) works as well as the initial metric set $(p, a, n, v, n_{read}, v_{read})$ to compress the original traces and generate quality synthetic traces. Figures 7a and 7b show that the compression ratios of the $(p, a, n, v, n_{read}, v_{read})$ and (p, a, n, n_{read}) methods are similar at different fractions of representative intervals for cello and snake traces. Figures 8a and 8b show that the relative errors of the CDF of disk response times generated by the $(p, a, n, v, n_{read}, v_{read})$ and (p, a, n, n_{read}) methods are quite close at different fractions of representative intervals for cello and snake traces. Therefore, we need only consider the abbreviated metric set for cluster synthesis, which reduces the time necessary for cluster analysis.

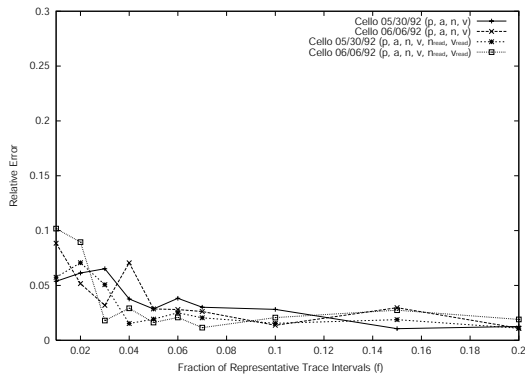


(a)

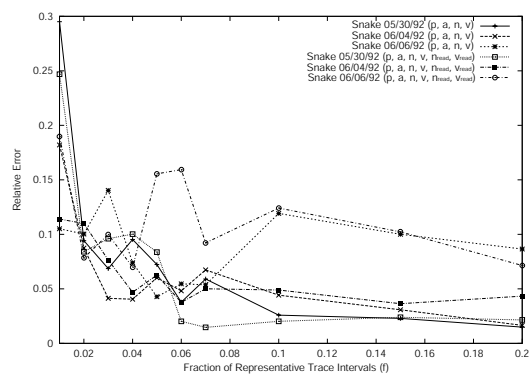


(b)

Figure 7: Compression ratio at different fractions of representative trace intervals and metric sets for (a) cello and (b) snake.



(a)



(b)

Figure 8: Relative error at different fractions of representative trace intervals and metric sets for (a) cello and (b) snake.

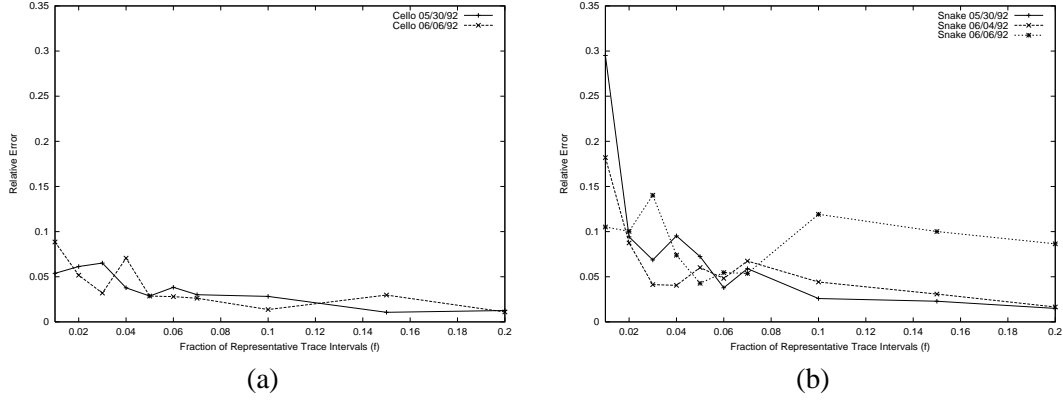


Figure 9: Relative error at different fractions of representative trace intervals for (a) cello and (b) snake.

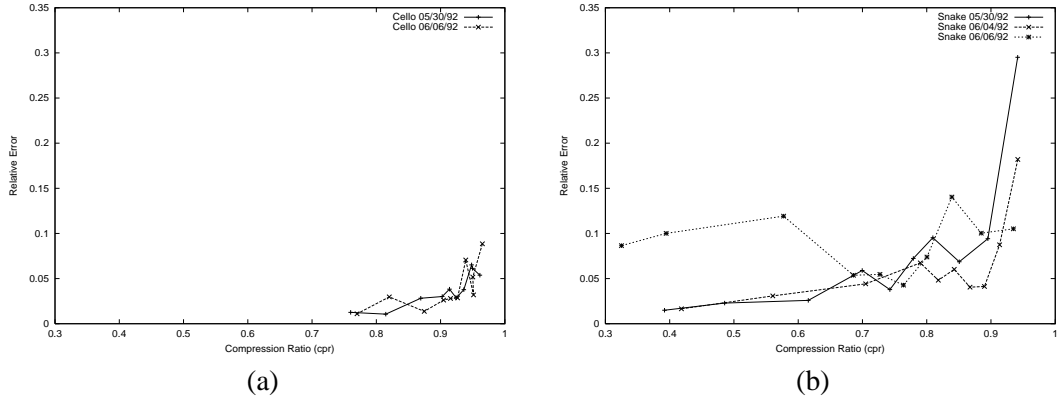


Figure 10: Relative error at different compression ratios for (a) cello and (b) snake.

5.2 Experimental Results

In this section, we evaluate the synthesis accuracy of the (p, a, n, n_{read}) method on cello traces (05/30/92 and 06/06/92) and snake traces (05/30/92, 06/04/92 and 06/06/92).

Figures 9a and 9b show the relative error of the CDF of disk response times of the (p, a, n, n_{read}) method at different fractions of representative trace intervals f for cello and snake, respectively. For the cello traces from 05/30/92 and 06/06/92 and the snake traces from 05/30/92 and 06/04/92, the relative error tends to decrease with the increase in the fraction of representative intervals, which means that as expected, with more representatives, the method can synthesize more accurate traces. The relative error is $< 5\%$ for cello and $< 10\%$ for snake with $f \geq 0.05$.

The error does not always decrease with the increase of the fraction of representative trace intervals f . There is fluctuation of relative errors with f in both workloads. In particular, Figure 9b shows that there is a significant increase in the relative error at $f = 0.1$ in the snake 06/06/92 trace; the error decreases with the increase of f when $f > 0.1$ but is still higher than the error at most of the smaller values of f . This is because our method changes the spatial locality in the real trace by replacing real trace intervals with their representatives.

Although the spatial locality within representative trace intervals is retained, the replacement of representative intervals can change spatial locality in real traces in two ways. It can decrease spatial locality of

a trace by replacing two consecutive trace intervals with spatial locality with two representatives without spatial locality. It also can increase spatial locality of a trace by replacing two consecutive trace intervals without spatial locality with two representatives with spatial locality, in particular, when the representatives are identical. Both effects interleave and affect the performance.

These effects are related to the fraction of representative trace intervals f . When f is low and the number of representative intervals is small, the chance of two consecutive real intervals sharing the same representative is high, which can increase spatial locality; however, fewer representatives retain less of the original access pattern. When f is high and the number of representative intervals is large, it is less likely that two intervals will share the same representative; however, more representatives retain more of the original access pattern.

Spatial locality among consecutive uniform intervals, which fit a uniform distribution and are non-bursty, is more important than among bursty intervals, which fit binomial multifractals. This is because in uniform intervals the response time is dominated by the mechanical movement of the disk and data transfer; queuing has little impact on the performance in uniform intervals.

We revisit the high error of the snake 06/06/92 trace shown in Figure 9b in light of our understanding of spatial locality. The snake 06/06/92 trace has fewer requests ($< \frac{1}{3}X$) and more non-empty trace intervals ($> 1.5X$) than the other snake traces. More than 50% of requests in the snake 06/06/92 trace are in the uniform intervals; on the contrary, the number for the other snake traces is 12–17%. The fraction of the bursty intervals in the snake 06/06/92 trace is very small (8%). There are many more uniform trace intervals, in terms of fraction and number, in the snake 06/06/92 trace than in the other snake traces. Consequently, spatial locality has greater impact on the performance of the snake 06/06/92 trace than the other snake traces and we observe significant performance degradation and high synthetic errors in the snake 06/06/92 trace, in particular, at the high values of f .

Figures 10a and 10b show the relative error of the CDF of disk response times of the (p, a, n, n_{read}) method at different compression ratios cpr for cello and snake, respectively. For the values of f we considered, we can see from Figure 6 that the lowest cpr is 76% for cello and 32% for snake. Thus, the data points with low compression ratios ($< 75\%$ for cello and $< 30\%$ for snake) do not exist, as shown in Figures 10a and 10b. For the cello traces from 05/30/92 and 06/06/92 and the snake traces from 05/30/92 and 06/04/92, the relative error tends to decrease with the compression ratio. For cello, the relative error is $< 5\%$ with a compression ratio $> 90\%$; for snake, it is $< 10\%$ with a compression ratio $> 75\%$.

The method is more successful for compressing original traces and generating synthetic traces for cello than snake. The logical sequentiality of cello and snake is 2% and 29%, respectively. Therefore, the replacement of representative trace intervals has less impact on the spatial locality of cello traces than that of snake traces. Cello news disk does not have cache so the caching effect introduced by our method is not as significant as snake usr2 disk.

6 Conclusions and Future Work

We have demonstrated that we can create a synthetic trace that reproduces the response time distribution of the original with a mean error of 5–10% and a compression ratio of 75–90% by using clustering to identify representative trace segments. The advantage of this approach is the obvious reduction in I/O trace size to reproduce certain performance metrics with high accuracy. The disadvantage is that long-range performance metrics, such as caching behavior, are distorted.

This approach works better for the traces containing fewer uniform trace intervals (i.e. the cello traces) because of the disturbance of spatial locality by replacement of representative trace intervals. This could be alleviated by using variable trace interval length. When a uniform arrival pattern is detected in real traces,

a larger interval length would be considered based on the access pattern. A variation of some disk cache prefetching scheme would be helpful.

We do not employ the information of spatial locality in real I/O traces in our method. A statistic needs exploiting to quantify spatial locality in the traces. With more information on spatial locality, our method could generate more accurate synthetic traces.

Our method assumes that some trace intervals have similar temporal structures, which result in similar performance characteristics. Such an assumption could be more suitable for the cello traces than the snake traces because the Hurst coefficient, which describes the degree of self-similarity, is higher for cello than for snake. The Hurst coefficient might be used to predict the applicability of our method.

Although the clustering methodology described is an offline method, it should be possible to investigate the feasibility of online methods to dynamically collect representative trace data, reducing tracing overhead.

Acknowledgements

We gratefully acknowledge Mengzhi Wang for her help with the bias code. We are also grateful to John Wilkes and his group for his suggestions on this work and for providing us traces and the Pantheon software. This work was supported in part by the National Science Foundation under grants NSF CCR-0093051 and NSF IDM-0083130.

References

- [1] AGRAWALA, A., MOHR, J., AND BRYANT, R. An approach to the workload characterization problem. *Computer* 9, 6 (June 1976), 18–32.
- [2] BERAN, J. *Statistics for Long-Memory Processes*. Chapman & Hall, New York, NY, 1994.
- [3] CALZAROSSA, M., AND SERAZZI, G. A characterization of the variation in time of workload arrival patterns. *IEEE Transactions on Computers C-34*, 2 (February 1985), 156–162.
- [4] EVERITT, B. S., AND DUNN, G. *Applied Multivariate Data Analysis*. Oxford University Press Inc., New York, NY, 2001.
- [5] EVERITT, B. S., LANDAU, S., AND LEESE, M. *Cluster Analysis*. Oxford University Press Inc., New York, NY, 2001.
- [6] GANGER, G. R. Generating representative synthetic workloads: An unsolved problem. In *Proceedings of Computer Measurement Group* (1995), pp. 1263–1269.
- [7] GROSSGLAUSER, M., AND BOLOT, J. C. On the relevance of long-range dependence in network traffic. *IEEE/ACM Transactions on Networking* 7, 5 (1999), 629–40.
- [8] HÖPPNER, F., KLAWONN, F., KRUSE, R., AND RUNKLER, T. *Fuzzy Cluster Analysis*. John Wiley & Sons, Inc., New York, NY, 1999.
- [9] HONG, B. Techniques for synthetic I/O workload generation. Tech. Rep. UCSC-CRL-02-16, University of California at Santa Cruz, 2002.
- [10] HONG, B., AND MADHYASTHA, T. M. The relevance of long-range dependence in disk traffic and implications for trace synthesis. Tech. Rep. UCSC-CRL-02-13, University of California at Santa Cruz, 2002.

- [11] JOLLIFFE, I. Discarding variables in a principal components analysis, I: Artificial data. *Applied Statistics* 21 (1972), 160–173.
- [12] JOLLIFFE, I. Discarding variables in a principal components analysis, II: Real data. *Applied Statistics* 22 (1973), 21–31.
- [13] MILLIGAN, G., AND COOPER, M. A study of standardization of variables in cluster analysis. *Journal of Classification* 5 (1988), 181–204.
- [14] NICKOLAYEV, O. Y., ROTH, P. C., AND REED, D. A. Real-time statistical clustering for event trace reduction. *The International Journal of Supercomputer Applications and High Performance Computing* 11, 2 (Summer 1997), 144–159.
- [15] RUEMMLER, C., AND WILKES, J. Unix disk access patterns. In *Proceedings of the Winter 1993 USENIX Conference* (San Diego, CA, Jan. 1993), pp. 405–420.
- [16] RUEMMLER, C., AND WILKES, J. An introduction to disk drive modeling. *IEEE Computer* 27, 3 (Mar. 1994), 17–28.
- [17] SERAZZI, G. A functional and resource-oriented procedure for workload modeling. In *Proceedings of the 8th International Symposium on Computer Performance Modeling, Measurement and Evaluation* (Nov 1981), pp. 345 – 361.
- [18] TAQUU, M. Statistical methods for long-range dependence. <http://math.bu.edu/people/murad/methods/index.html>. WWW.
- [19] WANG, M., T.MADHYASTHA, CHAN, N., PAPADIMITRIOU, S., AND FALOUTSOS, C. Data mining meets performance evaluation: Fast algorithms for modeling bursty traffic,. In *Proceedings of 18th Internal Conference on Data Engineering* (2002).
- [20] WARE, P. P., THOMAS W. PAGE, J., AND NELSON, B. L. Automatic modeling of file system workloads using two-level arrival processes. *ACM Transactions on Modeling and Computer Simulation* 8, 3 (July 1998), 305–330.
- [21] WIGHT, A. S. Cluster analysis in workload characterization for VAX/VMS. In *Proceedings of the Computer Performance Evaluation Users Group (CPEUG) 17th Meeting* (Nov 1981), pp. 21 – 34.
- [22] WILKES, J. The Pantheon storage-system simulator. Tech. Rep. HPL-SSP-95-14, Storage Systems Program, Computer Systems Laboratory, Hewlett-Packard Laboratories, Palo Alto, CA, May 1996.