Synthesizing Representative I/O Workloads Using Iterative Distillation

Zachary Kurmas
College of Computing
Georgia Tech
kurmasz@cc.gatech.edu

Kimberly Keeton
Storage Systems Department
Hewlett-Packard Labs
kkeeton@hpl.hp.com

Kenneth Mackenzie
College of Computing
Georgia Tech
kenmac@cc.gatech.edu

Abstract

Storage systems designers are still searching for better methods of obtaining representative I/O workloads to drive studies of I/O systems. Traces of production workloads are very accurate, but inflexible and difficult to obtain. The use of synthetic workloads addresses these limitations; however, synthetic workloads are accurate only if they share certain key properties with the production workload on which they are based (e.g., mean request size, read percentage). Unfortunately, we do not know which properties are "key" for a given workload and storage system.

We have developed a tool, the Distiller*, that automatically identifies the key properties ("attribute-values") of the workload. The Distiller then uses these attribute-values to generate a synthetic workload representative of the production workload. This paper presents the design and evaluation of the Distiller. We demonstrate how the Distiller finds representative synthetic workloads for simple artificial workloads and three production workload traces.*

1. Introduction

The behavior of large enterprise storage systems is heavily dependent upon the choice of workload. Consequently, potential design and configuration decisions must be evaluated with workloads that represent how the storage system will be used in a production environment.

Researchers generally use a combination of synthetic workloads and traces from production storage systems to drive storage system design studies. Synthetic workloads are artificially generated workloads intended to induce similar behavior on the underlying storage system by preserving the properties of the target realistic workloads on which they are based (e.g., same request interarrival time distributions, request size distributions, operation mixes, and locality) [4].

Synthetic workloads help to overcome many of the limitations of production traces: (1) Synthetic workloads can be specified using only the values of high-level attributes, which do not contain any user-specific information, thereby reducing privacy concerns. (2) Summarized workload attributes may be considerably smaller than a complete trace, making them easier to store and share over the Internet. (3) Synthetic workloads may better enable hypothetical studies: adjusting the attribute-values will change the resulting synthetic workload to approximate future workloads.

The main challenge is to choose a set of attributes that specify a representative synthetic workload. Most workload analysis and synthesis techniques [5, 6, 7, 8, 9, 16, 17] attempt to reproduce only one or two important workload properties. As a result, synthesizing a representative workload requires a tedious process of searching for each of the attributes that significantly affect storage system behavior. This process must currently be repeated for every workload and storage system combination under study.

This paper presents our approach for automating this tedious process, and describes our tool, the *Distiller*. Beginning with a trace of the target workload and a set of candidate attributes, the Distiller automatically determines which attributes should be used to synthesize a workload that is representative of the target. We consider two workloads to be representative when they have similar distributions of I/O response time when replayed on a given storage system.

We have used the Distiller to automatically find the key attributes for several simple artificial workloads and three production workloads — an email server, a transaction processing database server, and a decision support database server. We show that, for all but one of the target workloads considered, the Distiller produces a synthetic workload with a response time distribution within 12% of the target workload's response time distribution.

The Distiller chooses attributes from a library of known analysis and synthesis techniques. If the library is insufficient, the Distiller can identify which I/O request parameters are measured by the missing attribute, thus helping to guide the invention of a new analysis/synthesis technique. The Distiller can easily incorporate new attributes into its library as they are discovered — either as part of this in-

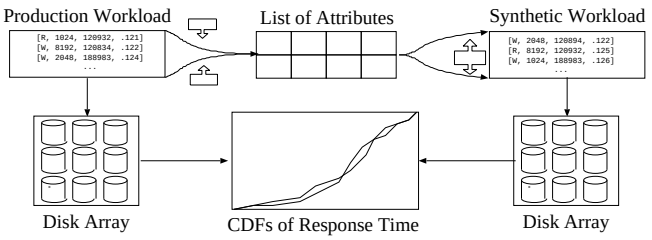

Figure 1. Problem statement.

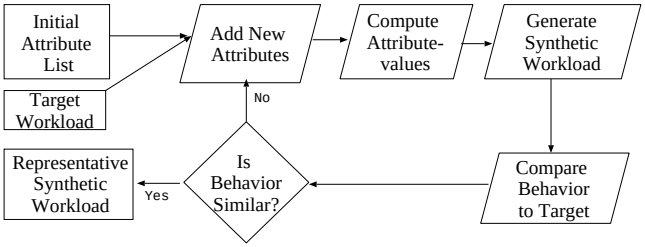

Figure 2. The Distiller's iterative loop.

vestigation, or from other investigations, as described in the literature. This extensibility allows the Distiller to easily evaluate new workloads and new storage technologies (e.g., MEMS-based storage), which may have different characteristics from known workloads and storage devices.

The remainder of this paper is organized as follows. Section 2 discusses related work, and Section 3 more formally defines the problem we are trying to solve. Section 4 discusses our automatic iterative approach for choosing attributes to generate representative synthetic workloads, and Section 5 presents experimental results. Section 6 concludes and outlines future work.

2. Related work

The literature describes and evaluates many techniques for generating synthetic block-level I/O workloads [5, 6, 7, 8, 9, 16, 17], file-level workloads [1, 3, 14] and application-level I/O workloads [11]. Our contribution is different: instead of presenting another synthetic workload generation technique, the Distiller leverages these existing techniques to automatically choose the ones that are most appropriate for the target workload and storage system. The current block-level analysis and corresponding generation techniques serve as the Distiller's "library" of candidate attributes.

Researchers have used principal component analysis (PCA) to generate batch computational workloads [2]. Given a large set of workloads and a set of attribute-values that characterize those workloads, PCA computes new variables, called principal components, that best describe the workloads. These principal components are uncorrelated linear combinations of the original attribute-value variables. Applying this approach to our research presents several challenges. First, to use PCA to choose the characteristics that describe workloads with similar performance, we would need to provide a set of workloads with similar performance. This is essentially the problem that we are trying to solve. Second, the resulting principal components are linear combinations of attributes, rather than a subset of the initial list of attributes, and hence may not have an intuitive meaning.

3. Problem definition and terminology

Our goal is to automatically determine which attributes specify a synthetic workload that is representative of the target workload. Specifically, we want the synthetic and target workloads to have similar performance when played against the same underlying storage system. Figure 1 illustrates this goal. In this section, we describe workload formats and target storage systems; then we define terminology and evaluation metrics.

Storage systems: We focus on block-level, disk-array-based storage systems commonly used in enterprise environments. Disk arrays provide a block-level I/O interface by exporting *logical units (LUs)*, which appear as "virtual disks" to the host. LUs are constructed from a subset of the array's disks and configured using a particular RAID layout (e.g., RAID5). Disk arrays generally employ a large non-volatile memory cache, leading to request response times that may vary by as much as three orders of magnitude.

Workload trace: Our "target" workloads are traces of I/O requests collected from production environments. Each I/O request is described by four parameters: the *operation type* (i.e., read or write), the *request size*, the *location* (i.e., logical address, including both LU ID and byte address), and the *interarrival time*. These workload traces serve as *open* workload models; therefore, we do not identify any process- or thread-level relationships between individual requests, nor do we model the "think-time" between related I/O requests.

Attribute and attribute-value: An *attribute* is a metric for capturing a workload characteristic (e.g., mean request size, read percentage, or distribution of location values). An *attribute-value* is an attribute paired with the quantification of that attribute for a specific workload (e.g., a mean request size of 8 KB, or a read percentage of 68%).

Evaluation criteria: We can determine performance similarity by considering a variety of performance characteristics (e.g., response time, throughput) and similarity measures (e.g., root mean square distance, Kolmogorov-Smirnov test). The design of the Distiller is independent of the performance metric and similarity measure chosen. In this paper, our performance metric is request response time distribution: the Distiller's synthetic workload should main-

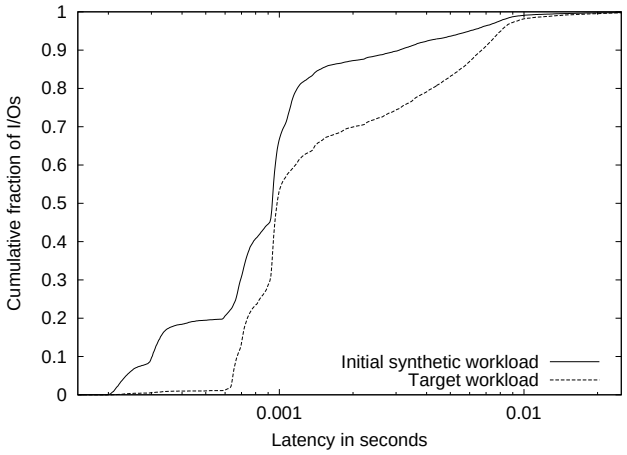

Figure 3. The Distiller cannot accurately synthesize the target Email workload using only the initial attribute list of empirical distributions for the trace parameters.

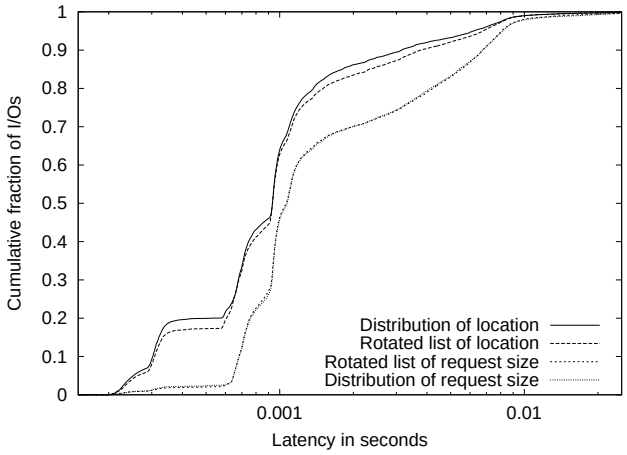

Figure 4. The difference between location lines (15% demerit) indicates the need for a $\{$location$\}$ attribute. The similarity of the request size lines (8% demerit) indicates no need for a $\{$request size$\}$ attribute.

tain the same response time distribution as the target workload when both are played against the same storage system. Our similarity metric is the *demerit figure* [13]. The demerit figure is the root mean square of the horizontal distance between the response time cumulative distribution functions (CDFs) for the synthetic and target workloads. We will present the demerit figure in relative terms, as a percentage of the mean response time of the target workload.

A synthetic workload that perfectly represents the target workload trace has a demerit figure of 0%. However, due to various experimental errors, it is difficult to achieve identical performance. Ganger distinguishes between *synthesis error*, due to the different synthesis techniques, and *randomness error*, the error of a single synthesis technique using different random seeds [5]. Because we are playing requests against a real storage environment, we may also experience *replay error*: the experimental error due to nondeterminism in the disk array and host operating system. Our experiments indicate that replay error can be as high as 10%; we therefore set our target at 12%, allowing for an additional 2% synthesis error and randomness error.

4. Our approach

In this section, we present our iterative approach for determining which attributes are necessary for synthesizing a representative I/O workload. This approach is embodied in a tool we call the *Distiller*.

At a high level, the Distiller iteratively builds a list of "key" attributes. During each iteration, the Distiller identifies one additional key attribute, adds it to the list, then tests the representativeness of the synthetic workload specified by the current list of key attributes. This loop (shown in Figure 2) continues until either (1) the difference between the performance of the synthetic and target workloads is below some user-specified threshold, or (2) the Distiller determines that no set of attributes in the library will specify a representative synthetic workload.

Running Email example: To make the concepts more concrete, we will use a running example to illustrate the Distiller's operation on a real production workload. Progress will be described in each section, and the results summarized in Table 4.

The target workload for this example is a 900-second trace of the workload created by the OpenMail email application. For simplicity, we will examine only one LU. The complete workload is described in more detail in [12]. Our baseline trace contains 19,769 I/Os, with an average request rate of 22 I/Os per second, and an average throughput of 164 KB/s. The workload contains highly randomized accesses using small requests that are mostly (72%) writes. Over 90% of the requests have request sizes of 8 KB or less; almost 50% are exactly 8 KB. The meta-data portion of the logical volume is frequently accessed, while the email message (i.e., data) portion of the volume does not exhibit the same temporal locality.

4.1 Initial attribute list

The Distiller's first step is to generate a synthetic workload based on a set of empirical distributions of values for the four I/O request parameters. We start with these explicit distributions because implicit distributions (e.g., normal or Poisson) have been shown to be inaccurate [5].

Running Email example: Figure 3 shows the response time distributions for the initial synthetic workload and the

Table 1. Examples of the subtractive method, using empirical distribution substitution and list rotations.

Target I/O Workload				Empirical Req Size	Rotated Req Size	Rotated Together		Rotated Apart	
Time	**Location**	**Op**	**Size**	**Size**	**Size**	**Op**	**Size**	**Op**	**Size**
0.050397	6805371	W (a)	3072 (a)	4096 (g)	3072 (f)	W (f)	3072 (f)	W (e)	3072 (f)
0.762780	7075992	R (b)	8192 (b)	3072 (a)	4096 (g)	R (g)	4096 (g)	W (f)	4096 (g)
0.789718	11463669	W (c)	3072 (c)	3072 (f)	2048 (h)	R (h)	2048 (h)	R (g)	2048 (h)
0.792745	7051243	R (d)	1024 (d)	8192 (b)	3072 (a)	W (a)	3072 (a)	R (h)	3072 (a)
0.793333	11460856	W (e)	8192 (e)	1024 (d)	8192 (b)	R (b)	8192 (b)	W (a)	8192 (b)
0.808625	11463669	W (f)	3072 (f)	2048 (h)	3072 (c)	W (c)	3072 (c)	R (b)	3072 (c)
0.808976	7049580	R (g)	4096 (g)	8192 (e)	1024 (d)	R (d)	1024 (d)	W (c)	1024 (d)
0.809001	7050244	R (h)	2048 (h)	3072 (c)	8192 (e)	W (e)	8192 (e)	R (d)	8192 (e)

target workload, which result in a demerit figure of 65%. Note the log scale on the x-axis. Given that the demerit figure is larger than the threshold of 12%, the Distiller must search for additional attributes.

4.2 Choosing an attribute group

Because the time to evaluate a synthetic workload's response time distribution is proportional to the length of the target workload, the Distiller should evaluate attributes in an intelligent order. Instead of evaluating attributes individually in an arbitrary order, the Distiller estimates the maximum potential benefit of an entire group of related attributes. Each group of related attributes captures a particular relationship within a single workload parameter or between multiple parameters. Once the Distiller determines which group holds the most potential for improvement, it can focus on finding the appropriate attribute to capture that relationship.

We define an *attribute group* as a set of attributes whose values are calculated using the same set of I/O request parameters (and thus capture information about the same relationship). For example, the {interarrival time} attribute group contains those attributes that measure only the interarrival times of different requests (e.g., mean interarrival time, the Hurst parameter). The {location, operation type} attribute group contains those attributes that measure the relationship between requests' locations and operation types (e.g., separate distributions of location values for read and write requests). All possible combinations of parameter relationships result in a total of fifteen attribute groups. By definition, each attribute is a member of exactly one attribute group.

To evaluate the potential benefit of an attribute group, the Distiller examines what happens when the relationship captured by the attribute group is destroyed. If a synthetic workload without the relationship under test performs similarly to the target workload, then the attribute group (and hence all of its member attributes) provides little or no benefit. However, if performance without the relationship is dramatically different, then the attribute group is important. We call this approach to evaluating the importance of attribute groups the *subtractive* method. The goal is to isolate the contribution of the relationship represented by the attribute group under test. For example, to isolate the effects of the {request size} attribute group, we want to separate its contribution from the contributions of {request size, operation type}, {request size, location}, and {request size, interarrival time}, as well as all three- and four-parameter attribute groups involving request size.

We isolate a given attribute group by replacing its parameter values in the target trace with either an empirical distribution or a rotation of the original values. The Distiller evaluates the potential contribution of a single-parameter attribute group using both of these techniques. Table 1 provides an example application of the subtractive method for {request size}. The left panel of the table shows the target workload. The second panel (labeled "Empirical Request Size") substitutes an empirical distribution for the request size parameter list, effectively removing all relationships from any of the groups involving request size ({request size}, {request size, location}, etc.). The third panel (labeled "Rotated Request Size") rotates the list of request sizes to maintain the intra-request size relationships, while destroying relationships between request size and the other parameters. The attribute group's contribution can then be isolated by comparing the difference between the response time distributions (using the demerit figure) for the empirical and the rotated workloads.

The Distiller evaluates the potential contribution of a multi-parameter attribute group using a combination of list rotations. First, for a parameter p, it evaluates the potential of all $\{p, x\}$ relationships (where x is another parameter) by comparing the performance for a rotated p list to the target workload. If the performance is sufficiently different, then at least one of these multi-parameter relationships must be important. The Distiller then isolates the contributions of each potential $\{p, x\}$ pairing using two

Table 2. Candidate attributes.

Attribute	Attr. Group	Description
empirical distribution	Any	histogram of values for this parameter obtained from the workload trace (This is the initial attribute for each parameter.)
list of values	Any	observed list of of values for the parameter in the target workload trace (This is the "perfect" attribute; it is used only for evaluation/comparison purposes)
Markov model	Any	higher-order Markov model with n different states, corresponding to different regions of the distribution; transition probabilities are determined empirically.
jump distance	{location}	distance (in KB) from the beginning of the previous request to the beginning of the current request.[1]
Markov model using combined attributes	{location}	Markov model using jump distance or *run count*. Run count is the number of requests in a sequential run (i.e., a series of adjacent requests with jump distance equal to the size of the previous request).

synthetic workloads with different rotations. Table 1 provides an example application of the subtractive method for {request size, operation type}. Again, the left panel represents the target workload. The fourth panel (labeled "Rotated Together") shows how request size and operation type can be rotated together, preserving the {request size, operation type}, {request size} and {operation type} relationships, while destroying the other relationships involving request size and operation type. The right panel (labeled "Rotated Apart") shows how separately rotating the two lists further destroys the {request size, operation type} relationship. The attribute group's contribution can then be isolated by computing the difference between the response time distributions (i.e., the demerit figure) for the "rotated together" workload and "rotated apart" workloads.

The Distiller's next step depends on the result of the demerit calculation for the isolated attribute group. If the demerit figure is less than the user-defined threshold, then the potential contribution for the attribute group is small enough that the addition of further attributes from that group is unwarranted. Otherwise, the attribute group's potential contribution is large enough that the Distiller should determine which attribute from the group to add to the list of key attributes.

In exploring attribute groups, the Distiller first evaluates the importance of single-parameter attribute groups (e.g., {location}, {request size}, {operation type}, and {interarrival time}). We refer to these iterations as *phase one* of the algorithm. After examining the single-parameter attribute groups, the Distiller addresses two-parameter attribute groups in *phase two*. Finally, if necessary, the Distiller will search for important three-parameter and four-parameter attributes. However, we have yet to encounter any workloads for which it is necessary to proceed beyond phase two.

Running Email example: The Distiller begins its exploration of attribute groups by applying the subtractive method to each single-parameter attribute group. Figure 4 shows two representative results for {request size} and {location}. The empirical distribution and rotated workloads for {request size} are similar, with a demerit figure of only 8%; thus, no additional {request size} attributes (beyond the default empirical distribution) are necessary. Although the distributions for the two {location} workloads look similar, the demerit figure of 15% is above our threshold of 12%. Therefore, the Distiller will search for more informative {location} attributes.

4.3 Picking an attribute

Once the Distiller has identified a promising attribute group, it must choose a specific attribute from that group. It explores the candidate attributes in a pre-determined order and evaluates how close each attribute comes to achieving the potential contribution of the attribute group. The Distiller incorporates the first eligible attribute encountered. This first-fit criterion allows us to avoid the execution time overhead of extra attribute evaluations.

The Distiller uses a variant of the subtractive method to evaluate candidate attributes. It generates a synthetic workload by using the candidate attribute, and preserving the list of original values for parameters not associated with the attribute under test. The Distiller then compares this synthetic workload against the synthetic workload that maintains "perfect information" for the attribute group (the "rotated" workload for single-parameter attributes, and the "rotated together" workload for multi-parameter attributes). If the two workloads have similar behavior (e.g., a demerit figure within the given threshold), the Distiller adds the attribute to the list of key attributes. If the two workloads have very different behavior, the attribute is not helpful and the Distiller proceeds to evaluate other candidate attributes.

[1] The literature traditionally defines jump distance as the distance between the end of the previous request and the beginning of the current request. Our modified definition allows jump distance to be strictly a {location} attribute.

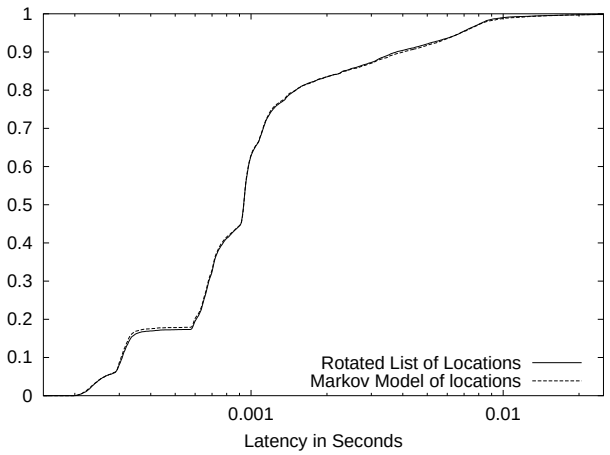

Figure 5. Markov model-generated location values are representative of the target workload's location values, so the Distiller adds the attributes to the list of key attributes.

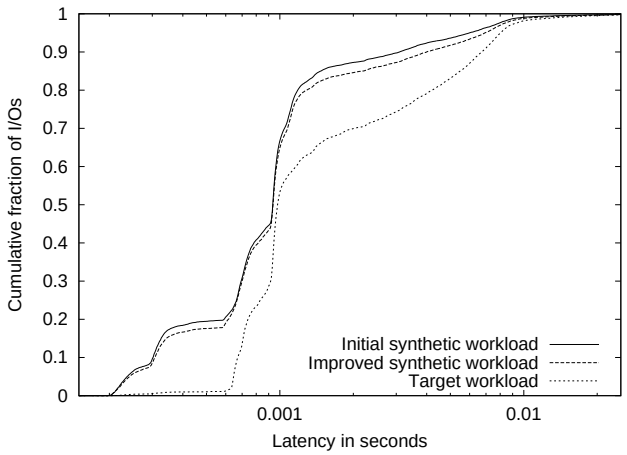

Figure 6. Although using a Markov model to choose location values improves the accuracy of the resulting synthetic workload, the improved workload is still not sufficiently representative.

If the Distiller evaluates every attribute in a group and finds none to be useful, then the library of attributes is insufficient. The user then has two options: (1) manually add more attributes to the library and re-start the Distiller; or (2) continue with the best available attribute from the library.

4.4 Attribute library

The Distiller implements a library of attributes described in the research literature. Table 2 describes the attributes for which we have implemented analysis and synthesis techniques.

The Distiller's Markov model-based attributes are all derived from a single "template" attribute: MM(d, i, s, h). To instantiate a specific attribute from this template, we must specify four parameters: (1) the *dependent parameter, d*: the request parameter being measured; (2) the *independent parameter, i*: the request parameter(s) on which the states are based; (3) the *number of states, s*, used to express the independent parameter;[2] and (4) the *history h*: the number of previous I/Os considered (e.g., based on the previous three operation types). This template allows specification of a higher-order Markov model where each state is dependent on i and h. For example, MM(loc, loc, 100, 1) refers to a Markov model where the next location (d) is determined from the location (i) of the single (h) last request. This attribute divides the address space of the LU into 100 (s) regions.

Running Email example: Recall from our earlier example that the Distiller had identified the {location} attribute group as the most promising. To explore this group, the Distiller first evaluates a Markov model of locations — MM(loc, loc, 100,1). Figure 5 shows that the Markov model-generated synthetic workload behaves very much like a workload with the original, rotated sequence of location values. Therefore, the Distiller adds the Markov model of locations to its key attribute list.

[2]Operation type has only two states: read and write. For the other request parameters, we generally use 100 states, and let each state correspond to the values between two percentiles.

4.5 Tracking progress for each iteration

After the Distiller identifies a new attribute of interest, it evaluates the synthetic workload specified by the improved attribute list. If the new workload is sufficiently representative, the iterative process concludes. Otherwise, the Distiller continues its loop of evaluating attribute groups and candidate attributes.

Running Email example: Figure 6 shows the results for the improved attribute list containing the Markov model of location values. Because the demerit figure (54%) is still well above the desired threshold, the Distiller continues.

4.6 Subsequent phases of Email example

After addressing each single-parameter attribute group, the Distiller addresses the two-parameter attribute groups. Recall that the Distiller begins this phase by comparing the single-parameter rotated workload for each I/O parameter p to the original workload trace to evaluate the potential of all $\{p, x\}$ multi-parameter attributes (where x is any other I/O parameter). It then determines whether breaking these multi-parameter relationships has a large effect on the resulting synthetic workload's behavior.

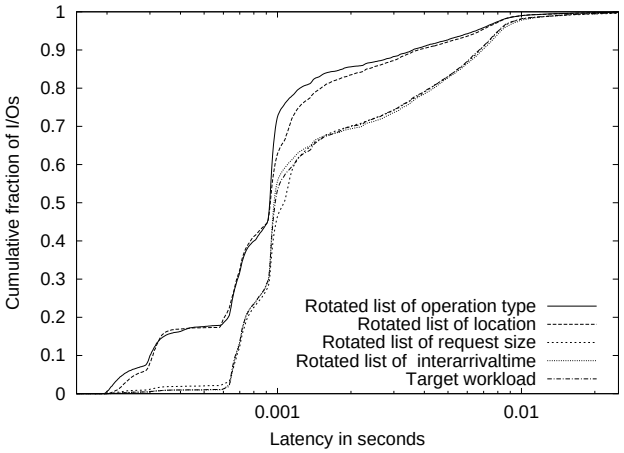

Figure 7. Inter-parameter relationships are important for operation type and location, but not for request size and interarrival time.

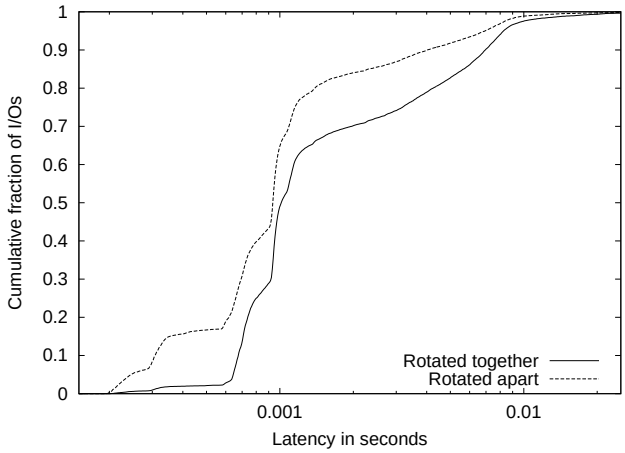

Figure 8. The potential for {operation type, location} is high, indicating that some attribute in this group will significantly affect performance.

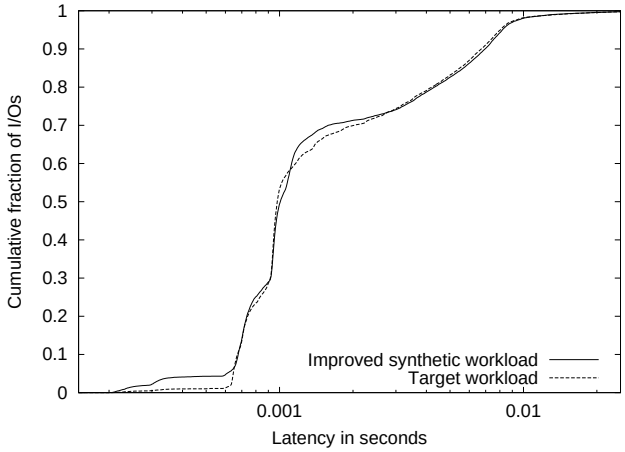

Figure 9. The improved synthetic workload (using separate Markov models for reads and writes) is representative of the target Email workload.

Figure 7 illustrates this process. We see little difference in behavior when {request size, w} and {interarrival time, x} relationships are broken; both demerit figures are less than 5%. However, the behavior of the rotated workloads for operation type and location differ significantly (demerit figures of 50% to 60%) from that of the target workload. Therefore, we conclude that some {operation type, y} attribute and some {location, z} attribute have a significant effect on behavior. The Distiller next identifies appropriate values for y and z by comparing the "rotated together" and the "rotated apart" workloads, as described in Section 4.2.

In the case of operation type, the Distiller evaluates the potential contribution of the {operation type, location}, {operation type, request size}, and {operation type, interarrival time} attribute groups. The high demerit figure (56%) in Figure 8 indicates that there is an important {operation type, location} attribute. Other experiments show that the {operation type, request size} and {operation type, interarrival time} attribute groups promise little benefit.

When the Distiller has identified an important two-parameter attribute group, it evaluates the candidate attributes as described in Section 4.3. For our example, the Distiller first evaluates separate Markov models of location values for read requests and write requests. The resulting synthetic workload has similar behavior to the workload where operation type and location were rotated together. Therefore, the Distiller adds this attribute to the list of key attributes.

Figure 9 shows the results for the Distiller's evaluation of the synthetic workload specified by the improved attribute list. Because the demerit figure (8%) is below the desired threshold, the Distiller terminates.

5. Experimental results

In this section, we present the results of using the Distiller to produce representative synthetic workloads for several target workloads. We examine both artificial workloads and production workloads. Artificial workloads are simple workloads generated from the Distiller's library, intended to verify that the Distiller works correctly. Production workloads are collected on real enterprise storage systems.

5.1 Experimental environment

The Distiller is responsible for applying the various subtractive methods, running the resulting experiments, examining the results, choosing the attribute group for improvement, and determining which attributes should be added to the list

Table 3. Workload parameters for target artificial workloads.

ID	Location	Operation Type	Interarrival Time	Request Size
W1	Uniform(0, 9GB)	50% reads	Constant(20ms)	Constant(8KB)
W2	Uniform(0, 9GB)	33% reads	Exponential(20ms)	Uniform(1KB, 128KB)
W3	MM(loc, loc, 4, 1) each state 96MB	MM(op, op, 2, 1)	MM(iat, iat, 3, 1) [.2ms, .9ms],[1ms, 5ms],250ms	MM(size, size, 4, 1) 1KB, 16KB, 64KB, 128KB
W4	MM(loc, loc, 4, 1) each state 96MB	MM(op, op, 2, 1)	MM(iat, iat, 3, 1) [.2ms, .9ms],[1ms, 5ms],250ms	MM(size, size, 4, 1) 1KB, 16KB, 64KB, 128KB
W5	MM(jump distance, loc, 4, 1) 98% probability of jump length determined by location	MM(op, op, 2, 1)	MM(iat, iat, 3, 1) [.2ms, .9ms],[1ms, 5ms],250ms	MM(size, size, 4, 1) 1KB, 16KB, 64KB, 128KB
W6	MM(loc, op, 2, 1) [0, 64MB] 95% R, 5% W [65MB, 10GB] 5% R, 95% W	MM(op, op, 2, 1)	MM(iat, op, 2, 2) W, W: .6ms R, W: 100ms W, R: 25ms R, R: 6ms	MM(iat, op, 2, 2) W, W: 128KB R, W: 65KB W, R: 2KB R, R: 16KB
W7	Runs of length Uniform(0,10) [0, 64MB] 90% R, 10% W [65MB,10GB] 10% R, 90% W	MM(op, op, 2, 1)	Four threads, each Exponential with following means: .03ms, .04ms, .05ms, .035ms	Constant(8KB)

of important attributes. It acts as an outer loop that coordinates the activities of several other software tools that help perform these tasks.

We use the HP-UX MeasureWare midaemon to collect I/O traces [10]. These traces are analyzed by the Rubicon tool [15] to produce both the attribute-values that characterize the workload and the response time distribution. Our workload generation tool takes a workload characterization from Rubicon as input, and generates a synthetic workload matching that characterization, which is then issued to a storage device.

All of the experiments presented in this paper were conducted on an HP FC-60 disk array populated with thirty 18 GB disks, spread uniformly across six disk enclosures, for a total of 0.5 TB of storage. The array is configured with five six-disk RAID5 LUs, each with a 16 KB stripe unit size. The 256MB disk array cache uses a write-back management policy backed with non-volatile RAM. Writes are considered complete once the data has been placed in the cache, and then later committed to the disk media (a process called "destaging"). Thus, from the perspective of the user application, most writes will appear as cache hits (e.g., almost "free"), provided that the write portion of the cache is not full.

5.2 Artificial workloads

In this section, we verify the correct operation of the Distiller. Table 3 shows the collection of artificial workloads we chose to stress test the Distiller. Table 4 presents the results of applying the Distiller to the artificial workloads in Table 3. Unless otherwise noted, the stopping condition for each workload is a demerit figure of 12%.

We now briefly describe each workload, the Distiller features it tests, and the experimental results:

W1 and W2 are simple workloads that are completely described by the empirical distribution attributes on the initial attribute list. The Distiller stops before even entering iteration 1 in both cases.

W3 demonstrates the Distiller's ability to find single-parameter attributes. The Markov models produce dependencies within the sequence of request parameters, but no inter-parameter dependencies.[3]

W4 shows that the Distiller can handle a temporary degradation in the accuracy of the synthetic workload (i.e., when the addition of a key attribute does not improve the representativeness of the resulting synthetic workload).

W5 demonstrates that the Distiller correctly chooses a useful attribute for the chosen attribute group, without settling on the first attribute it evaluates. During iteration 1, the Distiller tests and rejects three Markov models of location, and, instead, chooses a Markov model of jump distance.

W6 highlights the Distiller's ability to find attributes that describe multi-parameter correlations. In this workload, read and write accesses have different request sizes and are concentrated in different areas of the LU's address space. In addition, successive reads and successive writes have smaller interarrival times than a read followed by a write, or a write followed by a read. The Distiller correctly skips over single-parameter attribute groups and finds the appropriate multi-parameter attributes.

W7 shows that the Distiller can find a useful set of attributes, even if no attribute corresponds directly to the generation techniques. For this workload, we use a special run count generator to create sequential runs that vary uniformly from one to ten requests. Even though simple Markov models can only generate runs with an exponential distribution, such a Markov model is sufficient to specify a representative workload. Thus, the Distiller demonstrates that capturing runs is important, but that maintaining the exact run count

[3] For demonstration purposes, we set W3's threshold to 7% so that the Distiller would not terminate after iteration 3.

Table 4. Summary of selected workload results.

ID	Iter.	Attr. group	Attribute added	Result
W1	0		empirical distributions	3%
W2	0		empirical distributions	6%
W3	0		empirical distributions	60%
	1	{loc}	MM(loc, loc, 100, 1)	66%
	2	{op}	MM(op, op, 2, 8)	42%
	3	{size}	MM(size, size, 100, 1)	9%
	4	{iat}	MM(iat, iat, 4, 3)	5%
W4	0		empirical distributions	15%
	1	{loc}	MM(loc, loc, 10, 2)	22%
	2	{size}	MM(size, size, 100, 1)	9%
Email	0		empirical distributions	65%
	1	{loc}	MM(loc, loc, 100, 1)	56%
	2	{op, loc}	MM(loc, op, 2, 1)	6%
OLTP	0		empirical distributions	29%
Log	1	{loc}	MM(jump dist., loc, 100, 1)	6%

ID	Iter.	Attr. group	Attribute added	Result
W5	0		empirical distributions	63%
	1	{loc}	MM(jump dist, loc, 100, 1)	23%
	2	{size}	MM(size, size, 100, 1)	13%
	3	{iat}	MM(iat, iat, 100, 1)	11%
W6	0		empirical distributions	87%
	1	{op,size}	MM(size, op, 2, 1)	54%
	2	{op, loc}	MM(loc, op, 2, 1)	27%
	3	{op, iat}	MM((op,iat), (op, iat), 8, 2)	5%
W7	0		empirical distributions	78%
	1	{loc}	MM(jump dist, loc, 100, 1)	74%
	2	{op}	MM(op, op, 2, 8)	30%
	3	{op, loc}	MM(jump dist, (op, loc), 100, 1)	7%
OLTP	0		empirical distributions	53%
LU 1	1	{loc}	MM(loc, loc, 100, 1)	34%
	2	{op, loc}	MM(jump dist, op, 100, 2)	13%
DSS	0		empirical distributions	68%
	1	{loc}	Run Count Within State	18%

is not (at least for this workload).

The evaluation of these artificial workloads highlights two of the Distiller's strengths. First, the Distiller properly chooses attributes that produce representative synthetic workloads for the artificial workloads examined. (We have also demonstrated this correctness for many other artificial workloads, which are not presented here due to space considerations.) Second, the Distiller is able to identify which attributes are important, regardless of the techniques that actually generated the target artificial workloads.

5.3 Production workloads

In this section, we apply the Distiller to production workloads. Table 4 presents a summary of the results.

Email: Section 4 presented a detailed description of the Distiller's operation on one LU of this trace. It chooses a Markov model for location, which is later subsumed by separate Markov models for location based on operation type (i.e., {operation type, location}) to generate a final synthetic workload with a demerit figure of 6%.

OLTP log: This 1994 online transaction processing (OLTP) trace measures HP's Client/Server database application running a TPC-C-like workload at about 1150 transactions per minute on a 100-warehouse database. This target workload focuses on the busiest LU, the log, where accesses are highly sequential and write-only. The average request rate is 90 I/Os per second with an average throughput of 473 KB/s. The Distiller completes this trace in one iteration, only choosing an improved {location} attribute.

OLTP LU1: Accesses to the second-busiest LU in this workload are about 70% reads, with an average request rate of 90 I/Os per second and an average throughput of 57 KB/s. A visual inspection of the target trace shows accesses to groups of similar addresses, with no obvious pattern (e.g., sequential runs or strides) within each group. This workload is distilled in two iterations using a Markov model of location, which is later subsumed by Markov model of jump distance as a function of operation type; the final demerit figure is 13%.

DSS: This decision support system (DSS) trace was collected on an audited TPC-H system running the throughput test (multiple simultaneous queries) on a 300 GB scale factor data set. Accesses to this LU are read-only and nearly all 128 KB in size. The average request rate is 50 I/Os per second, with an average throughput of about 6400 KB/second. Each query generates a series of sequential I/Os. Visual inspection of the trace shows that many independent sequential streams have been interleaved together. This pattern does not match any of the previously described attributes in the Distiller's library.

When given this trace, the Distiller first identifies the need for a better {location} attribute. It then evaluates every {location} attribute in the library and finds that they all have demerit figures of at least 80%. Thus, the Distiller reports that the library's selection of {location} attributes is insufficient and terminates.

In response, we added an analyzer called "run count within state." Instead of measuring run count from the previous I/O, this analyzer measures run count from the previous *nearby* I/O. (Here "nearby" refers to spatial locality.) This analyzer can thus capture sequential runs, even if several runs are interleaved (provided that the interleaved runs are in different areas of the LU's address space). We re-ran the Distiller after adding the new analyzer to its library. The demerit figure to evaluate the potential of "run count within state" is only 17%, a great improvement over the existing set of attributes.

The DSS workload illustrates the Distiller's ability to help direct the development of new attributes when necessary. Because of the Distiller's extensible structure, it is easy to add analysis and synthesis modules for a new candidate attribute.

6. Conclusions and future work

This paper describes the design and evaluation of the *Distiller*, a tool that automatically determines the set of attributes necessary to specify a representative synthetic I/O workload. The Distiller's automation allows researchers to inexpensively design representative synthetic workloads, thereby enabling more comprehensive I/O system design studies.

The Distiller is designed around several key principles: (1) It builds the set of key attributes iteratively, adding one attribute per iteration. (2) It takes a divide-and-conquer approach by estimating the potential benefits of attribute groups and evaluating attributes in the most important attribute group first. (3) If its library of attributes proves insufficient, the Distiller can identify what relationships must be captured, thus helping to guide the invention of a new attribute. (4) Its extensible structure facilitates the incorporation of new attributes as they become available.

We demonstrated the Distiller's execution on both artificial and production workloads. These examples highlight the Distiller's key design principles and show that the Distiller can be an effective aid in the development of representative synthetic I/O workloads.

Our next step is to further improve the Distiller's operation (e.g., by adding attributes to the library) and to carefully examine the consequences of our design decisions (e.g., trade-offs between attribute-value precision and synthesis accuracy, effects of search algorithm choices). We also plan to further explore the properties of the Distiller's solutions, including optimality of the chosen attribute lists and representativeness across a broad range of workloads and storage system designs.

Acknowledgments

The authors wish to thank Eric Anderson, Mahesh Kallahalla, Terence Kelly, Ram Swaminathan and the anonymous reviewers for their comments, which greatly improved the quality of this paper. We also thank Eric Anderson, Mustafa Uysal and John Wilkes for their feedback on earlier versions of this work.

References

[1] R. R. Bodnarchuk and R. B. Bunt. A synthetic workload model for a distributed system file server. In *Proc. of SIGMETRICS*, pages 50–59, 1991.

[2] M. Calzarossa and G. Serazzi. Construction and use of multiclass workload models. *Performance Evaluation*, 19:341–352, 1994.

[3] M. R. Ebling and M. Satyanarayanan. SynRGen: an extensible file reference generator. In *Proc. of SIGMETRICS*, pages 108–117, 1994.

[4] D. Ferrari. On the foundations of artificial workload design. In *Proc. of SIGMETRICS*, pages 8–14, 1984.

[5] G. R. Ganger. Generating representative synthetic workloads: An unsolved problem. In *Proc. of the Computer Measurement Group Conf.*, pages 1263–1269, December 1995.

[6] M. E. Gomez and V. Santonja. A new approach in the analysis and modeling of disk access patterns. In *Performance Analysis of Systems and Software*, pages 172–177, 2000.

[7] M. E. Gomez and V. Santonja. A new approach in the modeling and generation of synthetic disk workload. In *Proc. of the 8th Intl. Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 199–206, 2000.

[8] B. Hong and T. Madhyastha. The relevance of long-range dependence in disk traffic and implications for trace synthesis. Technical Report UCSC-CRL-02-13, UC Santa Cruz, Available from http://www.cse.ucsc.edu/research/reports/, March 2002.

[9] B. Hong, T. Madhyastha, and B. Zhang. Cluster-based input/output trace synthesis. Technical Report UCSC-CRL-02-18, UC Santa Cruz, Available from http://www.cse.ucsc.edu/research/reports/, March 2002.

[10] HP OpenView Integration Lab. *HP OpenView Data Extraction and Reporting*. Hewlett-Packard Company, Available from http://managementsoftware.hp.com/library/papers/index.asp, version 1.02 edition, February 1999.

[11] W. Kao and R. K. Iyer. A user-oriented synthetic workload generator. In *Proc. of the 12th Intl. Conf. on Distributed Computing Systems*, pages 270–277, 1992.

[12] K. Keeton, A. Veitch, D. Obal, and J. Wilkes. I/O characterization of commercial workloads. In *Proc. of 3rd Wkshp. on Computer Architecture Support using Commercial Workloads*, 2001.

[13] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27(3):17–29, March 1994.

[14] C. A. Thekkath, J. Wilkes, and E. D. Lazowska. Techniques for file system simulation. *Software—Practice and Experience*, 24(11):981–999, November 1994.

[15] A. Veitch and K. Keeton. The Rubicon workload characterization tool. Technical Report HPL-SSP-2003-13, HP Labs, Storage Systems Department, Available from http://www.hpl.hp.com/SSP/papers/, March 2003.

[16] M. Wang, A. Ailamaki, and C. Faloutsos. Capturing the spatio-temporal behavior of real traffic data. In *Performance 2002*, 2002.

[17] M. Wang, T. M. Madhyastha, N. H. Chan, S. Papadimitriou, and C. Faloutsos. Data mining meets performance evaluation: Fast algorithms for modeling bursty traffic. In *Proc. of the 16th Intl. Conf. on Data Engineering*, 2002.