Practical applications of Iterative Workload Distillation

Zachary Kurmas[*], Kimberly Keeton[†]

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; however, they are 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 for a given workload and storage system. The Distiller then uses these attribute-values to generate a synthetic workload representative of the production workload.

The first half of this paper discusses the design of the Distiller and the potential benefits of the synthetic workloads it specifies. The second half of the paper presents our ideas on how research into the "key" properties of different workloads will benefit storage systems research. We discuss several experiments, what we hope to learn from each experiment, and the potential practical applications of what we learn.

Product status

This research is my Ph.D. thesis topic. I began this research as an intern at HP Labs, and have continued the work at Georgia Tech. Over the past several years, both HP Labs and Georgia Tech have provided generous financial support and technical advice; however, neither HP nor Georgia Tech are currently planning to release my research software as a product. My research software is not available because it requires several applications that are not publicly available.

1 Introduction

Storage systems designers are still searching for better methods of obtaining appropriate I/O workloads to drive studies of I/O systems. Traces of production workloads tend to be most accurate; however, as described in [1], the use of traces of real storage system activity has a number of limitations: (1) Traces are difficult to obtain, often for non-technical reasons. System administrators are reluctant to permit tracing on production systems; and, when traces are collected, it is often time-consuming to anonymize them to protect users' privacy. (2) Although any single trace file may not be huge, a set of trace files describing the activity of a system over a longer period of time (weeks or months) may occupy considerable space (e.g., tens of GB), making them difficult to store on-line and share over the Internet. (3) It is difficult to isolate and/or modify specific workload characteristics of a trace (e.g., arrival rate, total accessed storage capacity, distribution of request locations). This means that traces do not support explorations of slightly larger, busier, burstier, or other hypothetical future workloads.

An alternative approach is to use synthetic workloads. A synthetic workload is randomly generated in a way that preserves the key properties (henceforth called *attribute-values*[1]) of some realistic target workload. Synthetic workloads do not suffer from the limitations of workload traces: (1) Synthetic workloads can be specified using only

[*]Georgia Institute of Technology, College of Computing, 801 Atlantic Drive, Atlanta, GA 30332
[†]Hewlett-Packard Laboratories Storage Systems Department, 1501 Page Mill Rd. M/S 1134, Palo Alto, CA 94304
[1]See section 2 for a more formal definition of attribute-value.

values of the target workload's key attributes. These attribute-values do not contain user-specific information, thereby reducing privacy concerns. (2) We expect the attribute-values that specify a synthetic workload, if selected and presented optimally, to be two orders of magnitude smaller than a complete trace. (3) By adjusting the summarized attribute-values, we may be able to modify the resulting synthetic workload so it approximates future workloads.

The challenge is that we do not know precisely which attribute-values the synthetic workload workloads must share with the target workload on which it is based. If the attribute-values are not chosen carefully, the synthetic workload will not be *representative of* (i.e. behave like) the target workload. At a high level, we know that two workloads must have (among other things) the same degrees of spatial locality, temporal locality, and burstiness; however, we do not know how precisely to quantify and reproduce these properties.

Over the past several years, many different workload analysis and corresponding synthesis techniques have been developed [1, 2, 3, 4, 5, 6, 12, 13, 14]. Most of these techniques attempt to reproduce only one or two workload attributes chosen *a priori*. As a result, these techniques are useful for synthesizing only those workloads in which the reproduced attributes dominate the workload's behavior. Developing a representative synthetic workload based on these techniques requires a tedious process of searching for all of the workload's attributes that have a large effect on the behavior of the storage system, then finding appropriate techniques to reproduce each corresponding attribute-value. Because the set of important attributes may differ for different workload/storage system combinations, this process must currently be repeated for every workload/storage system under study.

Our solution was to develop a tool, the Distiller, to automate this tedious process. The Distiller, when given a workload trace and a library of possible workload attributes, automatically determines which attribute-values the target workload must share with the synthetic workload in order to be representative. These attribute-values contain the "essence" of the workload (relative to a given storage system) — that is, those workload properties that cannot be modified without affecting how the workload behaves.

By studying the essence of different workloads with respect to different storage systems, we will gain knowledge about how storage systems and workloads interact. We believe that this information will aid the design of storage system hardware, firmware, configuration policies, and analytic models. Similarly, studying how the essence of a workload changes over time may allow us to take old, out-dated traces and produce synthetic workloads that are representative of current workloads.

The first half of this paper discusses the Distiller and how we expect the synthetic workloads it produces to improve storage systems research. Section 2 presents several definitions and more formally states our research goal. Section 3 presents our experimental environment. Section 4 provides an overview of the Distiller. Section 5 lists several potential uses for synthetic workloads, gives examples of synthetic workloads specified by the Distiller, and outlines our research plans in this area.

The second half of this paper outlines our intended research into the essence of different workloads. Section 6 discusses several potential experiments, what we hope to learn from each, and the potential applications of what we learn. Section 7 discusses the trade-off between the size of the chosen attribute-values and the representativeness of the resulting synthetic workload. Section 8 discusses potential improvements to the Distiller; and section 9 concludes.

2 Terminology

Our goal is to automatically determine which attribute-values a synthetic workload must share with the target workload in order to be representative. In other words, we want to automatically find a set of attribute-values that specify a synthetic workload that behaves like the target workload on which it is based. Figure 1 illustrates this goal (where the distribution of response time is the behavior in question). In this section, we more precisely articulate this goal by defining terminology and evaluation metrics, and by describing workload formats and target storage systems.

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

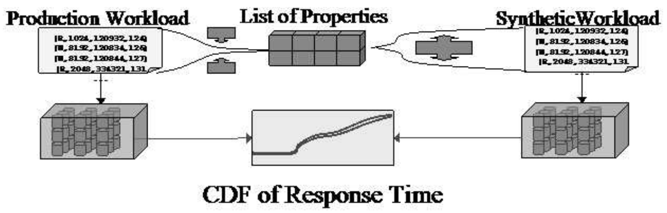

Figure 1: Problem statement. Our goal is to automatically determine which attributes are necessary to synthesize an I/O workload with a response time distribution that is representative of the original.

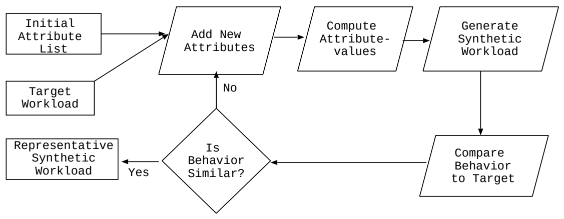

Figure 2: Distiller's iterative loop. The Distiller iteratively builds a set of attributes that specifies a representative synthetic workload.

views an attribute as a function, f, then an attribute-value is the pair $(f, f(x))$ for some workload trace x. Notice that an attribute and a target workload uniquely specify an attribute-value.

Workload trace: Our *target* workloads are traces of I/O requests collected from production environments.[2] Each I/O request comprises four parameters: the operation (i.e., "read/write") type, the request size, the location (i.e., logical address), and the arrival time. These workload traces serve as *open* workload models. Because we are using an open model, 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.

Storage systems: We designed our tool to distill the important attributes for any block-level storage system. In this paper, we will consider only disk-array-based storage systems. Disk arrays provide a block-level I/O interface by exporting *logical units (LUs)* of storage. LUs are constructed from a subset of the array's disks and configured using a particular RAID layout. An LU appears to be a single virtual "disk" to the host accessing it.

Evaluation metric: The Distiller can use any reasonable quantitative evaluation of a disk array's behavior. However, in this paper, we will consider only one disk-array evaluation metric: response time distributions. Specifically, the synthetic workload based upon the Distiller's choice of attributes should produce the same response time distribution as the target workload when played against the disk array under test.

We quantify the similarity of response time distributions using the *RMS demerit figure*, as defined in [11]. To compute the demerit figure, we plot the cumulative distribution functions (CDFs) of response time for the synthetic and target workloads, then compute the root-mean-square of the horizontal distance between the two CDFs. We will present this demerit figure in relative terms, as a percentage of the mean I/O time of the target workload.

A perfectly representative synthetic workload has a demerit figure (i.e., total error) 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 (or, in our case, different attribute-values), and *randomness error*, the error of a single synthesis technique using different random seeds [1]. Because we are playing requests against a real storage environment, we may also experience *replay error* — the experimental error due to non-determinism in the disk array and host operating system. Our experiments indicate that replay error can be as high as 7.5%. We believe a reasonable target for our automatic method is a total error (including replay, randomness, and synthesis errors) of at most 15% — a factor of two over the replay error.

3 Experimental environment

The remainder of this paper describes various aspects of our research and provides preliminary results in each area. We present here the hardware and workload traces used to collect these results.

[2]We collect a trace of a realistic workload on a production system. We then replay this trace against the storage system under test to obtain the target workload, and to permit apples-to-apples comparison with the synthetic workloads.

3.1 Storage Systems

All the experiments presented in this paper were conducted on an HP FC-60 disk array, on an HP FC-30 disk array, or using the Pantheon disk array simulator [15].

FC-60: The FC-60 disk array is populated with thirty 18 GB disks, spread uniformly across six disk enclosures, for a total of 0.5 TB of storage. The array has two redundant controllers in the same controller structure with one 40 MB/s Ultra SCSI connection between the controller enclosure and each of the six disk enclosures. The array is configured with five six-disk RAID5 LUs, each with a 16 KB stripe unit size.

The 256MB disk array cache, which is split between the two controllers, uses a write-back management policy backed with non-volatile RAM. Writes are considered to be completed once the data has been placed in the cache, resulting in a much lower latency than if they needed to be committed to the disk media. 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.

We measure response time of this disk array using the Measurement Interface Daemon (midaemon) kernel measurement system, part of the standard HP-UX MeasureWare performance evaluation suite [7]. The midaemon's I/O trace provides (among other information) the name, start time, and completion time of each system call. From this data, we can calculate the response time of individual I/O requests.

FC-30: The FC-30 is populated with thirty 4GB disks spread uniformly across 6 disk enclosures for a total of 120GB of storage. The disk controller and cache setup is similar to that of the FC-60; however, the FC-30 has only 64 MB of cache.

When running experiments on the FC-30, we use the trace replay application to measure the response time of each I/O; thus, each response time includes the overhead of the operating system. This method does not measure the performance of the disk array as accurately as the midaemon; however, it provides a more accurate measure of the latency seen by the application.

Pantheon: In addition to replaying workloads on real hardware, we also used the Pantheon disk array simulator [15]. This allowed us to evaluate the Distiller with respect to a larger variety of storage systems and storage system configurations. Specifically, we will use Pantheon to examine the effects of different *prefetch lengths* and *high water marks.*

- **Prefetching:** Upon receiving a request to read location x, a disk array configured to *prefetch* data will also read the data in locations $x + 1$ through $x + i$ (where i is set by the system administrator) and place it in the cache. Prefetching is especially beneficial for highly sequential workloads.
- **High-water mark:** When data is written to the disk, there is a high probability that that data will soon be modified. Thus, disks with NVRAM tend to avoid writing data to disk. However, the data must be eventually written to disk (*de-staged*) to make room for new data in the cache. In order to prevent requests from being queued behind such writes, the disk array often pre-emptively de-stages data when the cache is almost full. The level at which de-staging begins is called the *high water mark.*

3.2 Traces

We have thus far examined traces of four applications: a file server containing home directories of a research group (*Cello*), an e-mail server for a large company (*OpenMail*), a database server running a decision-support benchmark (*DSS*), and a transaction processing benchmark (*OLTP*).

Cello: The traces of the cello file system represent user activity on our main file server at HP Labs. The server stored a total of 63 file systems containing user home directories, news server pools, customer workload traces, HP-UX OS development infrastructure, etc., for a total of 238 GB of user data in a 479 GB physical storage. This is a typical I/O workload for a research group, involving software development, trace analysis, simulation, e-mail, etc.

OpenMail: The OpenMail workload is taken from the trace of accesses made by an OpenMail e-mail server [10] on a 640GB message store; the server was configured with 4487 users, of whom 1391 were active. The OpenMail trace has 1.1 million I/O requests, with an average size of 7KB.

DSS: The DSS workload represents decision support systems; it consists of the power and throughput tests from the DSS benchmark at the 300GB scale factor. We have thus far examined only the throughput test. This trace is one hour long and has 8.3 million I/O requests with an average size of 101KB.

OLTP: The OLTP workload represents on-line transaction processing environments. It is based on a mid-range OLTP benchmark configuration using one disk array and a two-processor server; overall, the transaction rate was 16.5K tpmC. The OLTP trace has 4.2 million I/O requests with an average size of 2KB.

4 Overview of the Distiller

In this section, we provide an overview of our tool, the Distiller. First, we explain precisely what the Distiller does. Next, we provide a high-level description of how it works. Finally, we present preliminary results that demonstrate the correctness of the Distiller.

4.1 Our approach

The Distiller takes as input a *target* workload trace and a library of attributes. It then either determines which attributes from the library are necessary to specify a representative synthetic workload, or determines that such a specification is not possible using the attributes in the library. The attributes chosen by the Distiller combined with the target workload define a set of attribute-values. These attribute-values serve as a set of constraints. Synthetic workloads must maintain the specified set of attribute-values.

At a high level, the Distiller iteratively builds a list of “key” attributes — attributes that noticeably influence the target workload’s behavior. During each iteration, the Distiller identifies one additional key attribute (from a library of candidate attributes), adds it to the list, then tests the representativeness of the resulting synthetic workload by replaying it on the storage system under test and comparing the resulting response time distribution to that of the target workload. This loop (shown in figure 2) continues until either (1) the difference between the behavior of the synthetic and target workloads is below some user-specified threshold (as described in Section 2), or (2) the Distiller determines that no set of attributes in the library will specify a representative synthetic workload.

In theory, the Distiller could simply add and evaluate attributes in an arbitrary order until the stopping conditions are met. However, doing so is likely to produce too many constraints. This poses several problems:

1. **Size:** Each attribute increases the size of the workload’s compact representation. Specifying too many attributes will negate the size benefit of using synthetic workloads.
2. **Complexity:** Generating synthetic workloads that meet a given set of constraints is not trivial. Generating a synthetic workload that meets a large set of constraints may be intractable.
3. **Essence:** Choosing too many attributes does not support our goal of finding the essence of a workload.

Evaluating all possible sets of attributes would produce a set of attributes that meet our size, complexity, and “essence” goals; however, doing so would be too time consuming. Even if each evaluation required only milliseconds, a library of only a few hundred attributes would make a exhaustive search intractable. Instead, we developed a method of estimating the maximum potential benefit of an entire group of related attributes using only a few evaluations. As a result, the Distiller is able to identify and evaluate the most useful attributes first. We present the key ideas here; see [8] for full details.

We define an attribute group as a set of attributes whose values are calculated using the same set of I/O request parameters. For example, the {location} attribute group comprises those attributes that consider only an I/Os request’s location (e.g., histogram of location values). The {location, operation type} group comprises attributes that consider both location and operation type. For example, separate histograms of location for read requests and write requests.

Trace	Disk array	Initial RMS	Best RMS	Size of compact representation	Attributes chosen by Distiller
OpenMail LU 1	FC-60	65%	6%	24%	Distribution of read locations Distribution of write locations Simple distributions for other parameters
OpenMail LU 2	FC-30	28%	5%	40%	Distribution of read locations Distribution of write locations Simple distributions for other parameters
OpenMail LU 3	FC-30	36%	8%	19%	Markov Model of (operation type, jump distance) pairs (location determined by jump distance) Simple distributions for other parameters
DSS LU 1	FC-60	68%	13%	4%	Location determined using RunCountWithinState (See [8]) Simple distributions for other parameters
OLTP LU 9	FC-30	18%	5%	70%	Location determined by jump distance (separate jump distance distributions for reads and writes) Simple distributions for other parameters
OLTP LU 11	FC-30	18%	3.1%	59%	Location determined by jump distance (only one jump distance distributions for all requests) Simple distributions for other parameters
OLTP LU 12	FC-30	14%	10%	73%	Location determined by jump distance (only one jump distance distributions for all requests) Simple distributions for other parameters
Other OLTP LUs	FC-30	4%	4%	1.5%	simple distributions for all parameters
Cello LU 1	FC-30	52%	20%	52%	Operation type and interarrival time generated by Markov model (Markov model supplemented by list of bursts) Distribution of read request size Distribution of write request size Markov Model of jump distance for read requests Markov Model of jump distance for write requests

Table 1: This table shows the quality of the best synthetic workload found by the Distiller, and the size of that workload's compact representation with respect to the size of the target trace.

To evaluate the potential benefit of all attributes in an attribute group, we compare the performance of two synthetic workloads: The first preserves almost none of the attributes in the group under test. The second preserves every attribute in that group. For example, choosing location values independently at random from a distribution preserves very few {location} attributes. In contrast, preserving the target workload's list of location values preserves every {location} attribute. The difference in performance of these two workloads is an estimate of the importance of the attributes in the group. Should there be little or no difference in performance, we know that no {location} attributes need to be preserved because even a naïve distribution of location values produces similar performance. (In practice, the synthetic workloads compared must be designed carefully so that the only differences between them are related to the attribute group under study. We refer interested readers to [8].)

Once the Distiller has identified the attribute group that contains the most important attributes, it searches that group for a suitable attribute. The Distiller evaluates the suitability of an individual attribute by comparing the performance of two synthetic workloads: The first workload preserves the attributes under test; the second does not. (As with choosing attribute groups, we have omitted several important details. We refer interested readers to [8].)

4.2 Preliminary results

We have thus far applied the Distiller to parts of each of the traces discussed in section 3.2. Table 1 presents a selection of typical results. In addition, figure 3 shows the response time distribution for the OpenMail workload trace, a synthetic workload generated using a naïve workload characterization, and a synthetic workload generated based on the output of the Distiller.

Notice that the Distiller cannot yet meet the target demerit figure of 15% for all workloads. This is because the Distiller is limited by its library of attributes. If a necessary attribute does not appear in the library, the Distiller

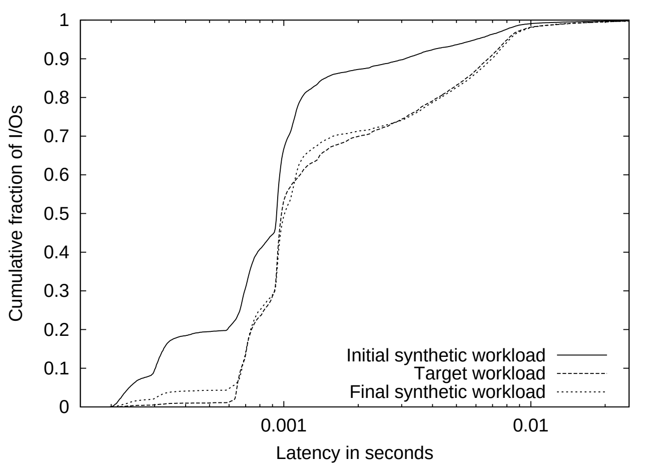

Figure 3: A naïve set of attributes produces an RMS demerit figure of 65% for OpenMail LU 1. Using the attributes specified by the Distiller reduces this error to 6%.

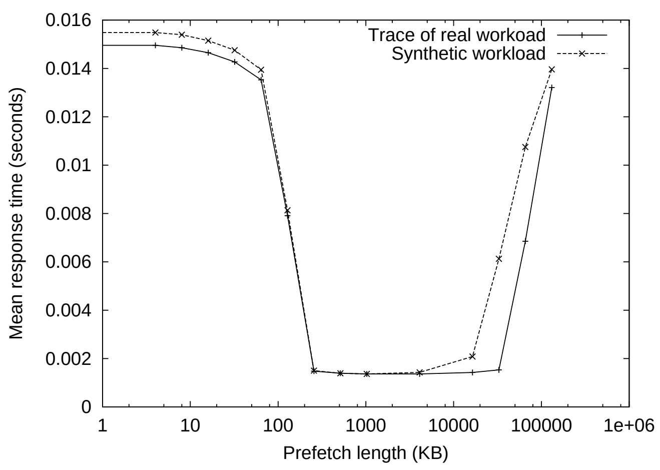

Figure 4: The synthetic workload specified by the Distiller is representative for any reasonable prefetch length.

will be unable to specify a representative synthetic workload.

In order to evaluate the Distiller independently of its library, we ran the Distiller using a set of artificial workloads as input. We generated these artificial workloads based on attributes in the Distiller's library, thereby assuring that the Distiller would be able to choose attributes that specify a representative synthetic workload. The details of these tests are in [8].

5 Uses of synthetic workloads

In this section, we discuss the potential uses of synthetic workloads. First, we discuss the use of synthetic workloads that are representative of some existing workload. Then, we discuss how the output of the Distiller may potentially be used to generate synthetic workloads that are representative of a future workload. In each case, we outline our future research plans and discuss how we believe the results will benefit storage systems research.

5.1 Synthesis of existing workloads

Before any storage system design decision can be incorporated into a product, it must be evaluated against many workloads so that the developers understand how the decision under test will affect all users. For reasons explained in the introduction, it is difficult to collect such a large and diverse set of workload traces. We expect that the synthetic workloads specified by the Distiller will serve as a useful supplement to the set of available workload traces.

5.1.1 Security and attributes chosen

Perhaps the most challenging obstacle to trace collection is the system administrators' privacy concerns. It is difficult to precisely enumerate all the information a clever adversary may be able to obtain from a full trace. Fortunately, however, the attribute-values specified by the Distiller represent a very specific set of information; system administrators should have a much easier time assessing the potential risks of making such information public. Furthermore, when optimized, these attribute-values will represent the minimum set of information necessary to reproduce a representative synthetic workload. Providing only the minimum information/access necessary to a user is a basic principle of computer security. Finally, preliminary results indicate that the Distiller tends to choose very high-level attributes. (See table 1.) This further reduces the likelihood of an insider intentionally placing sensitive information into a publicly available high-level trace description.

The OLTP benchmark has, thus far, been the simplest workload to distill. The footprint of many of the individual LUs is small enough to fit inside the FC-60's 256MB cache. These LUs are successfully characterized using only the empirical distributions of each I/O request parameter. The most complicated OLTP LUs require separate distributions of jump distance[3] for read and write requests.

We have also distilled several LUs from the DSS throughput test. This trace contains several simultaneous queries, and, as a result, contains many interleaved sequential runs.[4] We developed an attribute called "RunCountWithinState" to describe these runs. (See [8] for details.) This attribute combined with simple histograms for the other parameters successfully characterizes each DSS LU tested.

The two successfully distilled OpenMail LUs tested require separate distributions of location for read requests and write requests and an attribute that specifies the percentage of locations read from before they are written to; other parameters are characterized with simple distributions.

5.1.2 Size

Size is another challenge for those who maintain traces. Even with the exponential increase in storage capacity, storing and maintaining a suite of traces is still a non-trivial task. HP Labs has a collection of traces from over 20 different sources. These sources were traced anywhere from a few hours to a few months. In total, this collection of traces is over 500 GB.[5] We hope that a finely tuned Distiller will be able to reduce this size by a factor of 20 or more.

As seen in table 1, the size of the attribute-values chosen by the Distiller varies greatly. Many of the OLTP LUs can be concisely summarized using simple histograms. Similarly, the DSS LUs can be summarized using simple histograms, plus a list of run lengths.[6] As a result, these workloads have a compact representation that is about 2.5% that of a 900 second workload trace. In contrast, the patterns in the Open Mail and Cello workloads are more complicated. These complexities have a noticeable effect on the performance of the workload: The size of the compact representations for these workloads is about 30% that of a 900 second workload trace. The compact representations for Cello and OpenMail are fairly large because we have not yet developed finely-tuned attributes. The focus of our research is developing the Distiller and studying the chosen attributes; the development of new attributes for the Distiller's library is of lower priority.

We also note that for research purposes, we represent workload traces and attribute values using human-readable text files. We compressed the text files with bzip to approximate the size of a more efficient representation; however, a custom binary representation may be more compact than a compressed text file. We have also considered using XML to represent attribute values, then compressing the XML using *xmill* [9].

5.1.3 Other contributions

Because most design decisions are based on the resulting response times, the experiments presented in [8] should demonstrate the general usefulness of the Distiller. We will also conduct experiments in which we evaluate the Distiller's ability to produce synthetic workloads that can help make a a specific design decision. For example, we used Pantheon [15] to show that the optimal prefetch length was similar for a DSS workload trace and the synthetic trace specified by the Distiller. For this experiment, we simulated the execution of both the real and synthetic workloads using a range of prefetch lengths. Figure 4 shows that both workloads have the lowest mean response times when the prefetch length is set to 4096KB, demonstrating that the synthetic workload can be used in place of

[3] "Jump distance" refers to the difference between the location of the first byte of a I/O request and the last byte of the previous request.

[4] A "run" is a sequence of I/Os with jump distance of 0. An interleaved run is a sequence of I/Os which can be partitioned into several runs.

[5] This is a rough estimate obtained by running the *du* utility on the trace directory

[6] Because the runs tend to very long, there are relatively few runs that must be described. The representation for a workload with many short runs would not be nearly as concise.

the original workload trace. Future work includes simulating the effects of changing cache size and the disk array's high-water mark.

Our target consumer for synthetic workloads is the storage system researcher. To the best of our knowledge, there are relatively few publicly available block-level I/O traces. Results based on only a few workloads are weaker than those obtained with respect to a large variety of workloads. Furthermore, the lack of traces may discourage research in a particular area. For example, a graduate student may choose not to pursue an research designed to improve the performance of decision-support workloads if he can only effectively evaluate that design with respect to a trace of a file server.

Any improvement in storage systems research has an overall, general benefit: Those who build storage systems learn about technology that will help improve their products; and those who use storage systems gain access to more productive equipment. However, we also see a specific benefit for businesses with unique workloads: A trace is the representation of a workload that the company's storage systems must handle efficiently. By making the trace publicly available, a company invites researchers to address its specific problems.

5.2 Synthesis of future workloads

The biggest limitation of workload traces is their inflexibility. Although a workload trace perfectly represents a workload, it can represent only an existing workload. Traces of current workloads are of limited value in predicting whether particular design decision will continue to be beneficial as the workload grows. Similarly, workload traces have a limited ability to determine whether a design decision is "stable" – that is, if small changes to the workload will have large effects on performance.

5.2.1 Approximate future workloads

It is possible to modify a workload in an ad-hoc way to approximate an anticipated future workload. For example, one could simulate the doubling of a workload's request rate by dividing each interarrival time in half. This technique may be sufficient for some workloads (e.g., databases accessed primarily for reference); however, for many workloads (e.g. file servers), a doubling of the request rate is accompanied by some increase in the the footprint (set of disk sectors read or written). The appropriate method for simulating this increase in footprint size is not as obvious.

To demonstrate the usefulness of the Distiller in generating potential future workloads, we will compare ad-hoc methods of modifying workload traces to our method of adjusting attribute values. For example, we will approximate the doubling of the cello workload in two ways: First we will divide a one hour trace in half according to process ID (taking care that each half has similar characteristics). We will then approximate the doubling of this "half-workload" in three ways: First, we will simply divide all interarrival times in half, thus doubling the request rate. Second, we will map the half-workload onto an unused area of disk, then play both the half workload, and the re-mapped half-workload together. Third, we will double the attribute-values specified by the Distiller, and generate a synthetic workload based on these modified attribute-values (the exact method of "doubling" an attribute value will depend on the attribute). We will then compare the performance of these "doubled" workloads to the target trace and see which method produced the most representative "future" workload.

5.2.2 Stability and corner cases

In addition to estimating the future benefit of a proposed design decision, we want to make sure that the expected daily variations in a workload will not cause a disproportionate degradation in performance. For example, we want to make sure that the performance of the disk array will not degrade by 100% if the sequentiality of the workload drops by 5%.

The use of synthetic workloads based on adjusted attribute-values may eliminate the need for evaluators to maintain a large suite of traces. Instead of storing a month-long file system trace, evaluators may need only store a few select hours on a few select days, and store the appropriate range of attribute-value modifications.

Similarly, evaluators may be able to use synthetic workloads based on adjusted attribute-values to search for "corner cases" — those points at which the design decision under test is no longer effective. For example, we could decrease a sequentiality attribute-value and examine the behavior of a storage system using prefetching. This will help us to find and quantify the amount of sequentiality needed for prefetching to be useful.

6 Workload essence

When optimized, the Distiller will choose only those attribute-values that determine the behavior of the workload with respect to the storage system under test — in other words, the true "essence" of the workload. Changing the workload in a way that affects the essence will affect the behavior of the workload, whereas changing the workload in any other way will not. Because the Distiller is automatic, we can easily find the essence of many different workloads. As a result, it is now practical to obtain and compare the essence of many different workloads with respect to many different storage systems and configurations. This section discusses specifically what comparisons we will make and what we expect to learn.

We believe the attribute-values currently chosen by the Distiller contain more information than necessary to specify a representative synthetic workload. Finding the true essence of a workload requires a much larger and more precise library of attributes that is currently available; however, we believe that the attributes chosen by the Distiller are close enough to the true essence to be of use. We will obtain and compare the Distiller's output given many different workloads run on several different disk array configurations.

6.1 Chosen attributes depend on only storage system configuration

The behavior of a workload is primarily a function of how often, and in what manner, the workload exercises each of the storage system's components (disks, busses, cache, etc.). This function is further complicated by the presence of algorithms designed to efficiently handle expected workload patterns (i.e., request re-ordering, prefetching, etc.). We believe that, for each storage system configuration, there exists a set of attributes that can be used to specify a synthetic workload representative of any given target workload. (In other words, the set of attribute-values obtained by measuring target workload specify a synthetic workload representative of the target.)

The Distiller will be much more useful if there exists a single set of useful attributes per storage system configuration. In spite of its automation, the Distiller runs for several hours per workload. Finding this single set of attributes per configuration will eliminate the need to run the Distiller for each different workload. Instead, we need only run the Distiller when studying a new storage system (e.g., MEMS-based systems).

Obtaining a configuration-wide set of attributes may also help highlight the important features of a system — for example, by providing a method of specifying the range of attribute-values for workloads the storage system handles well, and/or workloads the storage system is not designed to handle. Understanding precisely which workload patterns the disk array handles best may help researchers develop configuration/layout algorithms that maintain the desirable properties. Likewise, understanding the precise cause of bottlenecks may help researchers develop hardware or firmware to specifically address the bottleneck. This information may also help manufacturers more effectively market disk arrays by helping them identify the best product for a particular customer and then demonstrate the usefulness to the customer.

6.2 How essence changes between systems

Many evaluations are designed to help the researcher choose between two or more competing design decisions (for example, how much data to prefetch). In order for synthetic workloads to be a useful supplement to such evaluations, they must be representative with respect to all design decisions under test. Therefore, we will study how the essence of a workload changes as the the storage system or storage system configuration changes.

In preliminary testing, we found that the attributes chosen for the TPC-H workload produced synthetic workloads that were representative given any reasonable prefetch length.[7] (See figure 4.) We will repeat similar experiments given other disk array configurations, and other design decisions such as cache size, and high-water mark. Similarly, we would like to compare the attributes used to characterize a workload with respect to the FC-30 and the FC-60. From this comparison, we hope to learn how to design synthetic workloads that will be representative with respect to both arrays. Our experience comparing the FC-30 and FC-60 may help companies more effectively compare other similar disk arrays.

Comparing the attributes chosen for different disk arrays (or disk array configurations) may allow us to associate attributes with specific disk array features. For example, if the only difference between two disk array configurations is that only one prefetches requests, than any attributes chosen for only one configuration may be associated with prefetching. Such associations will help us quickly chose useful attributes for new systems: We simply consider first those attributes previously associated with the features of the new storage system. Such associations may also help developers of analytical disk array models by directly relating workload properties to behavior (performance).

6.3 How essence changes over time

Finally, we will determine if the attributes chosen by the Distiller tend to remain the same over time. If so, we will determine whether we can predict the changes in the corresponding attribute-values. Specifically, we will attempt to produce a synthetic workload representative of a file system trace collected in 2002 based on the attribute-values of a trace of that same file system taken in 1996.

Business data tends to become less sensitive as time passes. Therefore, businesses are more willing to make older traces publicly available. Unfortunately, traces of old workloads are rarely representative of current workloads. The ability to "turn old workload traces into current workload traces" will greatly increase the number of useful traces available for study. Furthermore, the ability to age traces will also help us evaluate design decisions with respect to future workloads.

7 Tradeoff

In many cases, representativeness is a matter of degree.[8] As a result, there in an inherent tradeoff between the size and complexity of the attribute-values chosen by the Distiller and the representativeness of the resulting synthetic workload. We will study this tradeoff.

We can trade representativeness for simplicity in two ways: First, we can simply exclude an attribute from the compact representation. Second, we can vary the precision of most attributes. For example, a distribution of values is usually represented using a histogram. Reducing the number of histogram bins reduces the size of the representation, and (in most cases) reduces the representativeness of a synthetic workload based on that histogram.

The first tradeoff is of interest because it leads to a potential ranking of attributes. One could argue that the "most important" attribute is the one that, when removed, results in the least representative synthetic workload. This raises two questions:

1. Can we assign a percent contribution to each attribute? The correct way to do this is not obvious because the effects of attributes are not additive. Removing two attributes from a workload characterization can change the representativeness of the synthetic workload by almost any percentage.

2. Are some attributes, in general, "more important" than others? The answer to this question will help us improve the Distiller's search algorithm. It will also help improve our understanding of how workloads and storage systems interact by demonstrating how the effects of some workload patterns dominate others.

[7] This result applies only to the simulated disk array under test.

[8] In fact, the Distiller requires a measure representativeness that is comparative. It must be able to determine which of two synthetic workloads is more representative (e.g., has a lower RMS demerit figure). This is not possible with a measure that can only determine whether a workload is representative.

The second tradeoff is also of interest. It is too time consuming to have the Distiller evaluate many different precisions of the same attribute (for example, to evaluate 100 attributes that differ only by the number of histogram bins); therefore, the Distiller evaluates only large numbers of bins and states. This results in an unnecessarily (but not prohibitively) large compact representation. We will study how the representativeness of the specified synthetic workload changes as we decrease the precision of the chosen attributes. By better understanding how the representativeness of a workload is affected by the precision of individual attributes, we will be better able to choose attributes' precisions during the execution of the Distiller.

8 Improvements to the Distiller

Although the Distiller is already a useful tool, we see two areas of potential improvement:

1. **Optimization:** As explained in section 6, the attribute-values specified by the Distiller are not the true essence of the workload. We would like to eventually be able to make claims about the minimality of (1) the attributes chosen by the Distiller, (2) a workload's compact representation, and (3) the running time of the Distiller.

2. **Immediate inclusion and exclusion of attributes:** We believe that, in some cases, it is possible to estimate the potential effect of a specific attribute without generating and replaying a synthetic workload. For example, because the standard methods of generating the arrival pattern (e.g., drawing values independently at random from a distribution) will, by default, produce workloads that are not bursty, we expect the addition of a burstiness attribute will be of little value when distilling a non-bursty target workload. Similarly, we believe that some attributes will have a range of values that will indicate when the attribute is necessary to synthesize a representative synthetic workload.

9 Conclusion

We have developed a tool, the Distiller, that automatically identifies the attributes necessary to specify an accurate synthetic workload. Specifically, the Distiller, when given a workload trace and a library of possible workload attributes, automatically determines which attribute-values the target workload must share with the synthetic workload it models in order to be representative.

The synthetic workloads specified by the Distiller can be used in two main ways to help evaluate storage system design decisions. First, they can be used to supplement the set of available workload traces. Second, unlike workload traces, synthetic workloads are flexible and can be easily modified to represent future workloads.

In addition, the Distiller's automation greatly increases the practicality of obtaining the essence of many different workloads. This allows us to compare the essence of many different workloads run on many different storage systems. We can then use the information from these studies to better understand how workloads and storage systems interact. This improved understanding will hopefully lead to more accurate storage systems evaluations, improved storage systems designs (both hardware and software), and better storage system models.

References

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

[2] M. Gomez and V. Santonja. Self-similarity in I/O workload: Analysis and modeling. In *Workshop on Workload Characterization*, November 1998. Workshop held in conjunction with the 31st annual ACM/IEEE International Symposium on Microarchitecture, Dallas, Texas.

[3] 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 (ISPASS 2000)*, pages 172–177. IEEE, April 2000.

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

[5] B. Hong and T. Madhyastha. The relevance of long-range dependence in disk traffic and implications for trace synthesis. Technical report, University of California at Santa Cruz, 2002.

[6] B. Hong, T. Madhyastha, and B. Zhang. Cluster-based input/output trace synthesis. Technical report, University of California at Santa Cruz, 2002.

[7] 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.

[8] Z. Kurmas, K. Keeton, and K. Mackenzie. Iterative distillation of I/O workloads. In *Proceedings of the 11th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2003.

[9] H. Liefke and D. Suciu. Xmill: an efficient compressor for xml data. Technical Report MS-CIS-99-26, University of Pennsylvania, 1999.

[10] Openmail. Hewlett-Packard Company. http://www.openmail.com, 2001.

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

[12] S. Sarvotham and K. Keeton. I/O workload characterization and synthesis using the multifractal wavelet model. Technical report, Hewlett-Packard Labs Storage Systems Department, 2002.

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

[14] 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 *Proceedings of the 16th International Conference on Data Engineering (ICDE02)*, 2002.

[15] J. Wilkes. The Pantheon storage-system simulator. Technical Report HPL–SSP–95–14, Storage Systems Program, Hewlett-Packard Laboratories, Palo Alto, CA, December 1995.