Extracting Flexible, Replayable Models from Large Block Traces

Appears in the tenth USENIX Conference on File and Storage Technologies (FAST 2012)

V. Tarasov[1], S. Kumar[1], J. Ma[2], D. Hildebrand[3], A. Povzner[3], G. Kuenning[2], and E. Zadok[1]
[1]*Stony Brook University,* [2]*Harvey Mudd College, and* [3]*IBM Almaden Research*

Abstract

I/O traces are good sources of information about real-world workloads; replaying such traces is often used to reproduce the most realistic system behavior possible. But traces tend to be large, hard to use and share, and inflexible in representing more than the exact system conditions at the point the traces were captured. Often, however, researchers are not interested in the precise details stored in a bulky trace, but rather in some statistical properties found in the traces—properties that affect their system's behavior under load.

We designed and built a system that (1) extracts many desired properties from a large block I/O trace, (2) builds a statistical model of the trace's salient characteristics, (3) converts the model into a concise description in the language of one or more synthetic load generators, and (4) can accurately replay the models in these load generators. Our system is modular and extensible. We experimented with several traces of varying types and sizes. Our concise models are 4–6% of the original trace size, and our modeling and replay accuracy are over 90%.

1 Introduction

Traces are a time-honored way to collect information about real-world workloads. The information contained in traces allows a workload to be characterized using factors such as the exact size and offset of each I/O request, read/write ratio, ordering of requests, etc. By replaying a trace, users can evaluate real-world system behavior, optimize a system based on that behavior, and compare the performance of different systems [21, 23, 25, 34].

Despite the benefits of traces, they are hard to use in practice. A trace collected on one system cannot easily be scaled to match the characteristics of another. It is difficult to modify traces systematically, e.g., by changing one workload parameter but leaving all others constant. Traces are hard to describe and compare in terms that are easily understood by system implementors. Large trace files are time-consuming to distribute and can affect the system's behavior during replay by polluting the page cache or causing an I/O bottleneck [20].

In reviewing related work, we observed that in many cases replaying the exact trace is not required. Instead, it is often sufficient to use a synthetic workload generator that accurately reproduces certain specific properties. For example, a particular system might be more sensitive to the read-write ratio than to operation size. In this situation one does not really need to replay the trace precisely; a synthetic workload that emulates that read-write ratio would suffice. Of course, this example is simplistic, and in many cases one would be interested in more complex combinations of the workload parameters. However, the general idea that only some properties of the trace affect system behavior remains valid.

Because many systems respond only to a few parameters, researchers have developed many benchmarks and synthetic workload generators, such as IOzone [7], Filebench [12], and Iometer [33], which avoid many of the deficiencies of traces. But it can be difficult to configure a benchmark so that it produces a realistic workload; simple ones are not sufficiently flexible, while powerful ones like Filebench offer so many options that it can be daunting to select the correct settings.

In this work we propose to fill the gap between traces and benchmarks by converting traces into the languages of the benchmarks. We focus here on block traces due to their relative simplicity, but we plan to extend this work to other trace types, e.g., file system and NFS.

Our system creates a universal representation of the trace, expressed as a multi-dimensional matrix in which each dimension represents the statistical distribution of a trace parameter or a function. Each parameter is chosen to represent a specific workload property. We implemented the most commonly used properties, such as I/O size, inter-arrival time, seek distance, read-write ratio, etc. End users can easily add new ones as desired. For each benchmark, a small plugin converts the universal trace matrix into the specific benchmark's language.

Many workloads vary significantly during the tracing period. To address this issue, our system supports trace *chunking* across time. Within each chunk, the workload is considered to be stable and uniform and is expressed as a separate matrix. We use chunk deduplication to save space in periods where the workload is the same.

We evaluated the accuracy of our system by generating models from several publicly available traces. We first replayed each trace on a test system, observing throughput, latency, I/O queue length and utilization, power consumption, request sizes, CPU and memory usage, and the numbers of interrupts and context switches. Then we emulated the trace by running benchmarks with generated parameters on the same system, collected the same observations, and compared the results.

Our error was less than 10% on average, and 15% at

most; it can be controlled by varying several parameters. For a basic set of metrics, we converted a 1.4GB trace to the Filebench language in only 30s. The resulting trace description was 60MB, or 23.3× smaller.

2 Background and Motivation

Statistics Matter. Trace replay is a common evaluation technique because, unlike any other testing method, by definition traces represent reality. However, this realism comes at a price: the trace represents one instance of one system at one point in time. The next day's workload will inevitably be different, as will the same workload on a system with different hardware, competing workloads, etc. In the worst case, these variations might cause a system to be unintentionally optimized for an atypical operating point. Even if a trace accurately represents a target workload, rapid changes in hardware performance make it difficult to evaluate a design on a modern machine using measurements and traces captured on a different system only a few years earlier.

Our key observation is that for many purposes, *statistics* are what matter. The exact ordering of operations, their precise timing, the blocks or files accessed, and many other details recorded in a trace are variable and would change if it were re-recorded. Thus, when we replay a trace, we do not necessarily want to reproduce every detail as precisely as possible; instead, we would like to accurately represent its statistical properties.

An advantage of thinking of traces statistically is that they become much more flexible. For example, a trace collected a decade ago would record accesses to only a fraction of the blocks on a modern disk, and at a very different rate. Compared to a bulky trace, a statistical description is much simpler to scale to a modern machine and therefore provides a convenient abstraction for performing systematic evaluation of many systems.

Generating a good description requires representative trace properties to be selected. In general, the most appropriate properties depend on the system being tested, so it is impossible to create a complete list. For most purposes, however, the parameters of interest are well defined and widely adopted, e.g., I/O rate and distribution, read/write ratio. Thus, a statistical model of a trace should be able to capture those parameters, and should be able to describe them in sufficient detail so that no important information is lost. In particular, we should not reduce complex, empirically observed distributions to overly simple mathematical models, such as Poisson arrival processes, without justification.

Some workloads may also exhibit nonstandard, or even undiscovered, properties that might alter system behavior. It is therefore advisable to preserve the original traces to ensure these properties are retained. A workload generator can be adapted to include such characteristics once they are identified.

System Response. To evaluate a system empirically, workloads are applied and appropriate metrics measure its response. Performance is often characterized by throughput, latency, CPU utilization, I/O queue length, and memory usage [39,45]. Power consumption characterizes energy efficiency [29,36].

In many papers, these metrics are summarized by statistics such as averages or distributions. But as we argue above, it is often possible to accurately evaluate these metrics without resorting to a full and detailed trace replay. If the system response to a trace emulation is similar to that of a full replay, then emulation can replace full replay without biasing the results.

To evaluate the accuracy of our trace extraction and modeling system, we surveyed papers in Usenix FAST conferences from 2008–2011 and noted that the frequently used metrics fell into four categories: (1) throughput and latency; (2) I/O utilization and average I/O queue length; (3) CPU utilization and memory usage; and (4) power consumption. Most of the surveyed papers included 1–2 of these metrics, but in our study we evaluate all four types to ensure a comprehensive comparison. We claim that if all response metrics are similar, then the trace is modeled properly. We feel that our set of metrics is sufficiently representative and comprehensive to produce reliable results. There is still a chance that an unmeasured response parameter may differ; but our system is modular and easily extensible to emulate any additional metrics one desires.

Replay Methods. We use system response to evaluate our trace emulation accuracy. However, a system's response depends on the replay method, and varies based on the goal of the study. To study peak performance, traces are often accelerated [31, 40, 44, 48]. For power efficiency, traces are usually replayed verbatim to preserve realistic idle periods [5, 9]. To stress specific subsystems, a subset of the trace is sometimes replayed [38]. Our workload models can emulate existing trace-replay methods as well as more sophisticated ones.

3 Design

Our five design goals, in decreasing priority, are:

1. **Accuracy**: Ensure that trace replay and trace emulation yield matching evaluation results.
2. **Flexibility**: First, leverage existing powerful workload generators, rather than creating new ones. Therefore, traces should be translated into models that can be accurately described using the capabilities of existing benchmarks. Second, allow users to choose anything from accurate yet bulky models to smaller but less precise ones.

3. **Extensibility**: Allow the model to include additional properties chosen by the user.
4. **Conciseness**: The resulting model should be much smaller than the original trace.
5. **Speed**: The time to translate large traces should be reasonable even on a modest machine.

Feature Extraction. The first step in our model-building process is to extract important features from the trace. We first discuss how we extract parameters from workloads whose statistical characteristics do not change over time, i.e., stationary workloads. Then we describe how to emulate a non-stationary workload.

Each block trace record has a set of fields to describe the parameters of a given request. Fields may include the operation type, offset or block number, I/O size, timestamp, etc. Our translator is field-oblivious: it considers every parameter as a number. We designate these parameters as an n-dimensional vector $\vec{p} = (p_1, p_2, ..., p_n)$. We define a *feature function* vector on $\vec{p}$:

$$\vec{f} = (f_1(\vec{p}, s_1), f_2(\vec{p}, s_2), ..., f_m(\vec{p}, s_m)) = \vec{f}(\vec{p}, s_f)$$

Each feature function represents an analysis of some property of the trace; s_i represents private state data for the i-th feature function, which lets us define features across multiple trace entries and parameters.

For example, assume that p_1 and p_2 represent the I/O size and offset fields, respectively. We can then define the simple feature functions f_1—just the I/O size itself—and f_2—the logarithmic inter-arrival distance (offset difference between two consecutive requests):

$$f_1 = f_1(\vec{p}, s_1) = p_1$$

$$f_2 = f_2(\vec{p}, s_2) = \log(p_2 - s_2.\textit{prev\_offset})$$

In our translator, the user first chooses a set of m feature functions. Evaluating these functions on a single trace record results in a vector that represents a point in an m-dimensional feature space. The translator divides the feature space into buckets of user-specified size, and collects a histogram of feature occurrences in a multi-dimensional matrix—the *feature matrix*—that explicitly captures the relevant statistics of the workload, and implicitly records their correlations.

For example, using the two feature functions above, plus a third that encodes the operation (0 for reads, 1 for writes), the resulting feature matrix might look like the one in Figure 1. In this case, the trace held 52 requests of size less than 4KB and inter-arrival distance less than 1KB; of those, 38 were reads and 14 were writes.

By choosing a set of feature functions, users can adjust the workload representation to capture any important trace features. By selecting an appropriate bucket granularity, users can control the accuracy of the representation, trading off precision for computational complexity in the translator and matrix size. Stage 1 in Figure 2 shows the translator's role in the overall design.

Once the feature matrix has been created, the translator can perform a number of additional operations on it: projection, summation along dimensions, computation of conditional probabilities, and normalization. These operations can be used by the benchmark plugins (described below) to calculate parameters. For example, using the matrix in Figure 1, a plugin might first sum across the distance-vs.-size plane to calculate the total numbers of reads and writes, normalize these to find P(read), and then generate benchmark code to conditionalize I/O size on the operation type.

Clearly, the choice of feature functions affects the quality of the emulation; currently the investigator must do this based on the insight into the particular system of interest, e.g., whether it has been optimized for certain workloads that can be reflected in an appropriate feature function. We have implemented a library of over a dozen standard feature functions based on those commonly found in the literature [10, 11, 26, 30], including operation type, I/O size, offset distribution, inter-arrival distance, inter-arrival time, process identifier, etc. New feature functions can easily be added as needed to capture specialized system characteristics.

Benchmark Plugins. Once a feature matrix has been constructed from a trace, it is possible to use it directly as input to a workload generator. However, our goal in this research is not to create yet another generator. Instead, we believe that it is best to build on the work of others by using existing workload generators and benchmarks. This approach allows us to easily reuse all the exten-

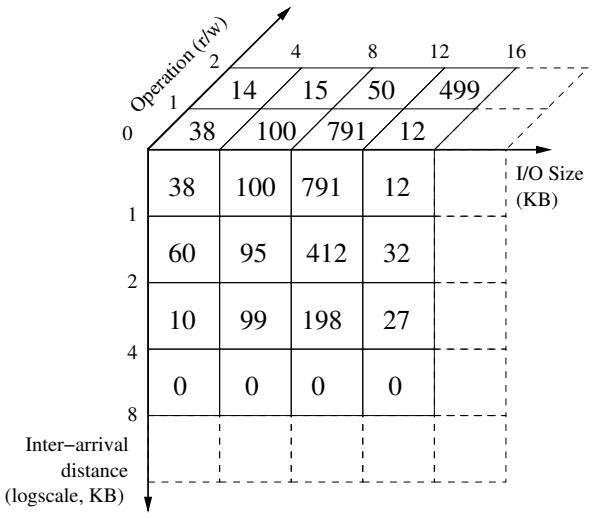


Figure 1: Workload representation using a feature matrix

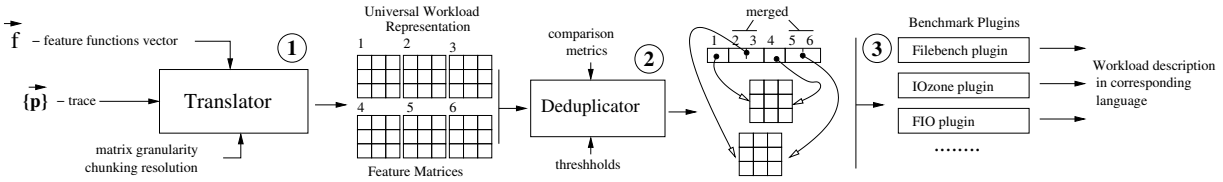


Figure 2: Overall System Design

sive facilities that these benchmarks provide. Many existing benchmarks offer a way to configure the workload that they generate; some offer command-line configuration parameters (e.g., IOzone [7] and Iometer [33]) while others offer a more extensive language for that purpose (e.g., Filebench [12] and fio [13]).

Most existing benchmarks use statistical models to generate a workload. Some of them use average parameter values; others use more complex distributions. In all cases, our feature matrices contain all the information needed to control the models used by these benchmarks. A simple plugin translates the feature matrix into a specific benchmark's parameters or language. For some benchmarks, the expressiveness of the parameters might limit the achievable accuracy, but even then the plugin will help choose the best settings to emulate the original trace's workload. Stage 3 in Figure 2 demonstrates the role of the benchmark plugins in the overall design.

For our initial investigations, we have implemented plugins for Filebench and IOzone. We chose Filebench for its flexibility, and IOzone because it is more suitable for micro-benchmarking. We found that it was easy to add a plugin for a new benchmark, since only a single function has to be registered with the translator. The size of the function depends on the number of feature functions and the complexity of the target benchmark.

Chunking. Many real-world traces are non-stationary: their statistical characteristics vary over time. This is especially true for traces that cover several hours, days, or weeks. However, most workload generators apply a stationary load, and cannot vary it over time. We address this issue with *trace chunking*: splitting a trace into chunks by time, such that the statistics of any given chunk are relatively stable. Finding chunk boundaries is difficult, so we first use a constant user-defined chunk size, measured in seconds. For each chunk, we compute a feature matrix independently; this results in a sequence of matrices. We then convert these fixed chunks into variable-sized ones by feeding the matrices to a deduplicator that merges adjacent similar matrices (Stage 2 in Figure 2). This optimization works well because many traces remain stable for extended periods before shifting to a different workload mode. We normalize the matrices before comparing them, so that the absolute number of requests in a chunk does not affect the comparison. We use the maximum distance between matrix cells as a metric of similarity. When two matrices are found to be similar, we average their values and use the result to represent the workloads in the corresponding time chunks.

Besides detecting varying workload phases, the deduplication process also reduces the model size. To achieve even further compression, we support all-ways deduplication: every chunk in a trace is deduplicated against every other chunk (not just adjacent ones).

Along with the matrices, we generate a time-to-matrices map that serves as an additional input to the benchmark plugins. If the target benchmark is unable to support a multi-phase workload, the plugin generates multiple invocations with appropriate parameters.

In the example in Figure 2, we set the trace duration to 60s and the initial chunk size to 10s, so the translator generated six matrices. After all-ways deduplication, only two remained.

4 Implementation

Traces from different sources often have different formats. We wanted our translator to be efficient and portable. We chose the efficient and flexible DataSeries format [2]—recommended by the Storage Networking Industry Association (SNIA)—and we selected SNIA's draft block-trace semantics [37]. We wrote converters to allow experimentation with existing traces in other formats. We also created a block-trace replayer for DataSeries, which supports several commonly used replay modes. In total we wrote about 3,700 LoC: 1,500 in the translator, 800 in the converters, 1,000 in the DataSeries replayer, and 400 in the Filebench and IOzone plugins. We plan to release these publicly.

5 Evaluation

To evaluate the accuracy, conversion speed, and compression of our system, we used multiple micro-benchmarks and a variety of real traces. In this paper we present evaluation results based on two traces: Finance1 [28] and MS-WBS [22]. The Finance1 trace captures the activity of several OLTP applications running at two large financial institutions. The MS-WBS traces were collected from daily builds of the Microsoft Windows Server operating system. The high-level characteristics of the traces are presented in Table 1.

It is fair to assume that the accuracy of our translator might depend on the system under evaluation. In

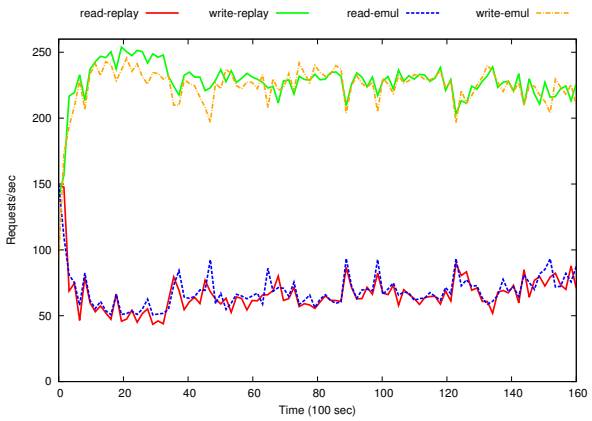


Figure 3: Reads and writes per second, Setup P, Fin1 trace.

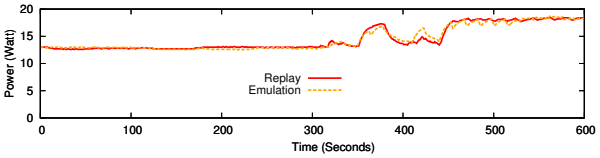


Figure 4: Disk power consumption, Setup P, MS-WBS trace.

our experiments we used a spectrum of block devices: various disk drives, flash drives, RAIDs, and even virtual block devices. In this paper we present results from two extremes of the spectrum. In the first experimental setup—*Setup P*—we used a *P*hysical machine with an external SCSI Seagate Cheetah 300GB disk drive connected through an Adaptec 39320 controller. The fact that the drive was powered externally allowed us to measure its power consumption using a WattsUp meter [43].

The second experimental setup (*Setup V*) is an enterprise-class system that has a *V*irtual machine running under the VMware ESX 4.1 Hypervisor. The VM accesses its virtual disks on an NFS server backed by a GPFS parallel file system [19, 35]. The VM runs CentOS 6.0; the ESX and GPFS servers are IBM System x3650's, with GPFS using a DS4700 storage controller. Accuracy metrics were recorded at the NFS/GPFS server.

On both setups, we first replayed traces and then emulated them using Filebench. In all experiments we set the chunk size to 20s and enabled all feature functions. We chose the matrix granularity for each dimension experimentally, by gradually decreasing it until the accuracy began to drop. During all runs we collected the accuracy parameters specified in Section 2 using the *iostat*, *vmstat*, and *wattsup* tools; we plotted graphs showing the

Characteristic	Finance1	MS-WBS
Duration	12 hours	1.5 hours
Reads/Writes (10^6)	1.2/4.1	0.7/0.6
Avg I/O size	3.5KB	20KB
Seq. Requests	11 %	47%

Table 1: High-level characteristics of the used traces

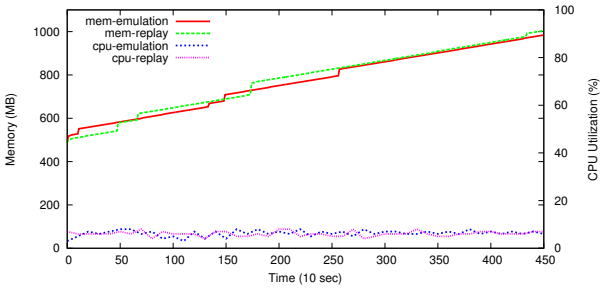


Figure 5: Memory and CPU usage, Setup P, Fin1 trace.

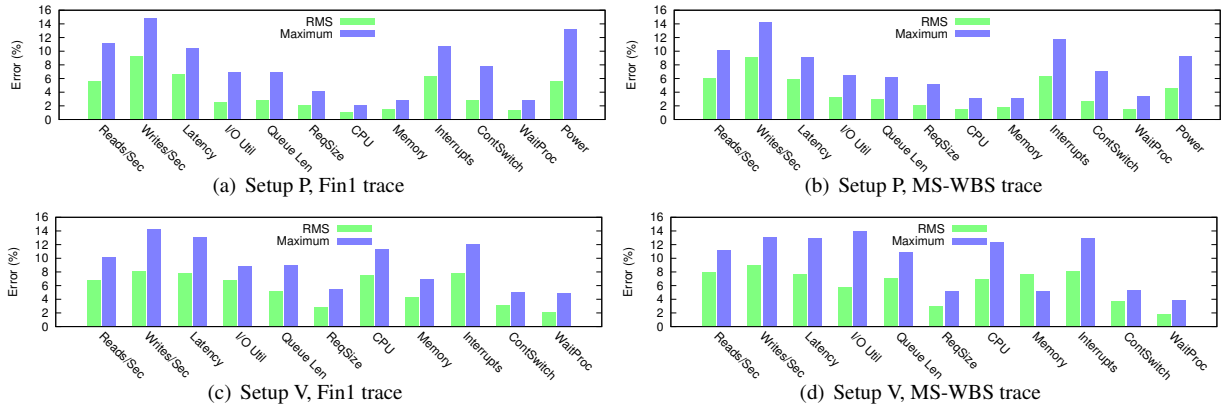
value of each accuracy parameter versus time for both replay and emulation. Due to limited space, we only present the graphs for a few representative accuracy parameters. However, we give the average and maximum emulation error for all experiments.

Figure 3 depicts how the throughput—for both reads and writes—changes with time for the Finance1 trace. The replay was performed with infinite acceleration; it took about 5 hours to complete on Setup P. The trace emulation line closely follows the replay line; the Root Mean Square (RMS) distance is lower than 6% and the maximum distance is below 15%. In the beginning of the run, read throughput was 4 times higher then later in the trace. By inspecting the model we found that the workload exhibits high sequentiality in the beginning of the trace. After startup, the read throughput falls to 50–100 ops/s, which is reasonable for an OLTP-like workload and our hardware. The write performance is 2–2.5 times higher than for read, due to the controller's write-back cache that makes writes more sequential.

Figure 4 depicts disk-drive power consumption in Setup P during a 10-minute non-accelerated replay and emulation of the MS-WBS trace. In the first 5 minutes trace activity was low, resulting in low power usage. Later, a burst of random disk requests increased power consumption by almost 40%. The emulation line deviates from the replay line by an average of 6%.

In Setup V, the GPFS server was caching requests coming from a virtual machine. As a result, the run time of the Fin1 trace was only 75 minutes. The memory and CPU consumption of the GPFS server during this time are shown in Figure 5. Memory usage rises steadily, increasing by about 500MB by the end of the run, which is the working-set size of the Fin1 trace. Discrepancies between replay and emulation are within 10%, but there are visible deviations at times when the memory usage steps up. We attribute this to the complexity of the GPFS's cache policy, which is affected by a workload parameter that we did not emulate. CPU utilization remained steadily about 10% for both replay and emulation.

Figure 6 summarizes the errors for all parameters, for both setups and traces. The maximum emulation error was below 15% and RMS distance was 10% on average. Although the maximum discrepancy might seem high,



(a) Setup P, Fin1 trace (b) Setup P, MS-WBS trace

(c) Setup V, Fin1 trace (d) Setup V, MS-WBS trace

Figure 6: Root Mean Square (RMS) and maximum relative distances of accuracy parameters for two traces and two systems.

Figure 3 shows sufficient behavioral accuracy.

The selection of feature matrix dimensions is vital for achieving high accuracy. If a system is sensitive to a workload property that is missing in the feature matrix, accuracy can suffer. For example, disk- and SSD-based storage systems may have radically different queuing and prefetching policies. To ensure high-fidelity replays across both types of systems, the feature matrix should capture the impact of appropriate parameters.

The chunk size and matrix granularity also affect the model's accuracy. Our general strategy is to select these parameters liberally at first (e.g., 100s chunk size and 1MB granularity for I/O size) and then gradually and repeatedly restrict them (e.g., 10s chunk size, 1KB I/O size) as needed until the desired accuracy is achieved. One can always be guaranteed to get high enough accuracy if sufficiently small numbers are used.

Conversion Speed and Model Size. The speed of conversion and the size of the resulting model depend on the trace length and the translator parameters. On our 2.5GHz server, traces were converted at about 50MB/s, which is close to the throughput of the 7200RPM disk drive. The resulting model without deduplication was of approximately 10–15% size of the original trace. Deduplication removed over 60% of the chunks in both the Fin1 and MS-WBS traces, resulting in a final model size reduction of 94–96%. All sizes were measured after compressing both traces and models using gzip.

6 Related Work

The body of research related to traces is large; we cite only a representative sample. Many studies have focused on accurate trace collection with minimum interference [1, 4, 24, 31, 32]. Other researchers have proposed trace-replaying frameworks at different layers in the storage stack [3,20,48,48,49]. Since a trace contains information about the workload applied to the system, a number of works focused on trace-driven workload characterization [22, 23, 25, 34]. N. Yadwadkar proposed to identify an application based on its trace [46].

After a workload is characterized, a few researchers have suggested a workload model that allows them to generate synthetic workloads with identical characteristics [6, 14–18, 41, 42, 47]. These works address only one or two workload properties, whereas we present a general framework for any number of properties. Also, we chunk data and generate workload expressions for the languages of already existing benchmarks.

The two projects most closely related to ours are Distiller [27] and Chen's Workload Analyzer [8]. Distiller's main goal is to identify important workload properties. We can use this information to intelligently define dimensions for our feature matrix. Chen uses machine learning techniques to identify the dependencies between workload features. However, the authors do not emulate traces based on the extracted information.

7 Conclusions and Future Work

We have created a system that extracts flexible workload models from large I/O traces. Through the novel use of chunking, we support traces with time-varying statistical properties. In addition, trace extraction is tunable, allowing model accuracy and size to be traded off against creation time. Existing I/O benchmarks can readily use the generated model by implementing a plugin. Our evaluation with Filebench and several block traces demonstrated that the accuracy of generated models approaches 95%, while the model size is less than 6% of the original trace size. Such concise models allow easy comparison, scaling and other modifications.

In the future we plan to support file-system-level traces, build multi-layer models, and add flexibility in the analysis phase. Our current chunking method is simple and we want to investigate alternative chunking techniques. We will also work on a graphical tool for manual trace chunking. To avoid manual selection of the translator's parameters, we want to explore various artificial intelligence approaches. To further reduce the model size, we plan to improve the compression ratio by matching empirical distributions in the feature matrix to explicit

mathematical functions. We recognize that our list of accuracy metrics is not complete and want to experiment with other accuracy parameters (e.g., latency distributions). We also plan to develop tools and techniques that will simplify various operations on our models, such as time and size scaling, and comparison to other models.

References

[1] E. Anderson. Capture, conversion, and analysis of an intense NFS workload. In *Proceedings of the Seventh USENIX Conference on File and Storage Technologies (FAST '09)*, 2009.

[2] E. Anderson, M. Arlitt, C. Morrey, and A. Veitch. DataSeries: an efficient, flexible, data format for structured serial data. *ACM SIGOPS Operating Systems Review*, 43(1), January 2009.

[3] E. Anderson, M. Kallahalla, M. Uysal, and R. Swaminathan. Buttress: A toolkit for flexible and high fidelity I/O benchmarking. In *Proceedings of the Third USENIX Conference on File and Storage Technologies (FAST '04)*, 2004.

[4] A. Aranya, C. P. Wright, and E. Zadok. Tracefs: a file system to trace them all. In *Proceedings of the Third USENIX Conference on File and Storage Technologies (FAST '04)*, 2004.

[5] T. Bisson, S.A. Brandt, and D.D.E. Long. A hybrid disk-aware spin-down algorithm with I/O subsystem support. In *Proceedings of the IEEE 2007 Performance, Computing, and Communications Conference (IPCCC)*, 2007.

[6] P. Bodik, A. Fox, M. Franklin, M. Jordan, and D. Patterson. Characterizing, modeling, and generating workload spikes for stateful services. In *Proceedings of the First ACM Symposium on Cloud Computing (SOCC)*, 2010.

[7] D. Capps. IOzone file system benchmark. `www.iozone.org`.

[8] Y. Chen, K. Srinivasan, G. Goodson, and R. Katz. Design implications for enterprise storage systems via multi-dimensional trace analysis. In *Proceedings of the 23rd ACM Symposium on Operating System Principles (SOSP '11)*, 2011.

[9] F. Douglis, P. Krishnan, and B. Bershad. Adaptive disk spin-down policies for mobile computers. In *Proceedings of the Second Symposium on Mobile and Location-Independent Computing*, 1995.

[10] M. Ebling and M. Satyanarayanan. SynRGen: An extensible file reference generator. In *Proceedings of the 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 1994.

[11] D. Ellard, J. Ledlie, P. Malkani, and M. Seltzer. Passive NFS tracing of email and research workloads. In *Proceedings of the Second USENIX Conference on File and Storage Technologies (FAST '03)*, 2003.

[12] Filebench. `http://filebench.sourceforge.net`.

[13] fio—flexible I/O tester. `http://freshmeat.net/projects/fio/`.

[14] A. Ganapathi, Y. Chen, A. Fox, R. Katz, and D. Patterson. Statistics-driven workload modeling for the cloud. In *Proceedings of the International Workshop on Information and Software as Services (WISS)*, 2010.

[15] G. Ganger. Generating representative synthetic workloads: an unsolved problem. In *Proceedings of Computer Measurement Group Conference (CMG)*, 1995.

[16] M. Gomez and V. Santonja. A new approach in the modeling and generation of synthetic workloads. In *Proceedings of the 8th Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2000.

[17] B. Hong and T. Madhyastha. The relevance of long-range dependence in disk traffic and implications for trace synthesis. In *Proceedings of the 22nd IEEE / 13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST)*, 2005.

[18] B. Hong, T. Madhyastha, and B. Zhang. Cluster-based input/output trace analysis. In *Proceedings of 24th IEEE International Performance, Computing, and Communications Conference (IPCCC)*, 2005.

[19] IBM. IBM scale out metwork attached storage. `www.ibm.com/systems/storage/network/sonas/`.

[20] N. Joukov, T. Wong, and E. Zadok. Accurate and efficient replaying of file system traces. In *Proceedings of the Fourth USENIX Conference on File and Storage Technologies (FAST '05)*, 2005.

[21] S. Kavalanekar, D. Narayanan, S. Sankar, E. Thereska, K. Vaid, and B. Worthington. Measuring database performance in online services: a trace-based approach. In *Proceedings of TPC Technology Conference on Performance Evaluation and Benchmarking (TPC TC)*, 2009.

[22] S. Kavalanekar, B. Worthington, Q. Zhang, and V. Sharda. Characterization of storage workload traces from production windows servers. In

Proceedings of IEEE International Symposium on Workload Characterization (IISWC), 2008.

[23] T. Kimbrel, A. Tomkins, R. Patterson, B. Bershad, P. Cao, E. Felten, G. Gibson, A. Karlin, and K. Li. A trace-driven comparison of algorithms for parallel prefetching and caching. In *Proceedings of the Second Symposium on Operating Systems Design and Implementation (OSDI 1996)*, 1996.

[24] A. Konwinski, J. Bent, J. Nunez, and M. Quist. Towards an I/O tracing framework taxonomy. In *In Proceedings of the International Workshop on Petascale Data Storage (PDSW)*, 2007.

[25] G. H. Kuenning, G. J. Popek, and P. Reiher. An analysis of trace data for predictive file caching in mobile computing. In *Proceedings of the Summer 1994 USENIX Conference*, 1994.

[26] Z. Kurmas. *Generating and Analyzing Synthetic Workloads using Iterative Distillation*. PhD thesis, Georgia Institute of Technology, 2004.

[27] Z. Kurmas, K. Keeton, and K. Mackenzie. Synthesizing representative I/O workloads using iterative distillation. In *Proceedings of IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems (MASCOTS)*, 2003.

[28] LASS. UMass trace pepository. `http://traces.cs.umass.edu`.

[29] T. Li and L. K. John. Run-time modeling and estimation of operating system power consumption. In *Proceedings of the 2003 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 2003.

[30] Z. Li, Z. Chen, S. M. Srinivasan, and Y. Zhou. C-Miner: Mining block correlations in storage systems. In *Proceedings of the Third USENIX Conference on File and Storage Technologies (FAST '04)*, 2004.

[31] M. P. Mesnier, M. Wachs, R. R. Sambasivan, J. Lopez, J. Hendricks, G. R. Ganger, and D. O'Hallaron. //TRACE: parallel trace replay with approximate causal events. In *Proceedings of the Fifth USENIX Conference on File and Storage Technologies (FAST '07)*, 2007.

[32] R. Moore. A universal dynamic trace for Linux and other operating systems. In *Proceedings of the 2001 USENIX Annual Technical Conference (ATC)*, 2001.

[33] OSDL. Iometer project. `www.iometer.org`.

[34] J. Ousterhout, H. Costa, D. Harrison, J. Kunze, M. Kupfer, and J. Thompson. A trace-driven analysis of the UNIX 4.2 BSD file system. In *Proceedings of the Tenth ACM Symposium on Operating System Principles (SOSP)*, 1985.

[35] F. Schmuck and R. Haskin. GPFS: A shared-disk file system for large computing clusters. In *Proceedings of the First USENIX Conference on File and Storage Technologies (FAST '02)*, 2002.

[36] P. Sehgal, V. Tarasov, and E. Zadok. Evaluating performance and energy in file system server workloads extensions. In *Proceedings of the Eighth USENIX Conference on File and Storage Technologies (FAST '10)*, 2010.

[37] Storage Networking Industry Association (SNIA). Block I/O trace common semantics (working draft). `www.snia.org/sites/default/files/BlockIOSemantics-v1.0r11.pdf`, February 2010.

[38] C. A. N. Soules, G. R. Goodson, J. D. Strunk, and G. R. Ganger. Metadata efficiency in versioning file systems. In *Proceedings of the Second USENIX Conference on File and Storage Technologies (FAST '03)*, 2003.

[39] A. Traeger, N. Joukov, C. P. Wright, and E. Zadok. A Nine Year Study of File System and Storage Benchmarking. *ACM Transactions on Storage (TOS)*, 4(2):25–80, May 2008.

[40] B. Trushkowsky, P. Bodik, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson. The SCADS director: scaling a distributed storage system under stringent performance requirements. In *Proceedings of the Nineth USENIX Conference on File and Storage Technologies (FAST '11)*, 2011.

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

[42] M. Wang, T. Madhyastha, N. Chan, and S. Papadimitriou. Data mining meets performance evaluation: fast algorithms for modeling burst traffic. In *Proceedings of 16th International Conference on Data Engineering (ICDE)*, 2002.

[43] Watts up? PRO ES Power Meter. `www.wattsupmeters.com/secure/products.php`.

[44] C. Weddle, M. Oldham, J. Qian, A. A. Wang, P. Reiher, and G. Kuenning. PARAID: a gear-shifting power-aware RAID. In *Proceedings of the Fifth USENIX Conference on File and Storage Technologies (FAST '07)*, 2007.

[45] S. Wu, H. Jiang, D. Feng, L. Tian, and B. Mao. WorkOut: I/O workload outsourcing for boosting RAID reconstruction performance. In *Proceedings*

of the Seventh USENIX Conference on File and Storage Technologies (FAST '09), 2009.

[46] N. Yadwadkar, C. Bhattacharyya, and K. Gopinath. Discovery of application workloads from network file traces. In *Proceedings of the Eighth USENIX Conference on File and Storage Technologies (FAST '10)*, 2010.

[47] J. Zhang, A. Sivasubramaniam, H. Franke, N. Gautam, Y. Zhang, and S. Nagar. Synthesizing representative I/O workloads for TPC-H. In *Proceedings of International Sypmposium on High Performance Computer Architecture (HPCA)*, 2004.

[48] N. Zhu, J. Chen, and T. Chiueh. TBBT: scalable and accurate trace replay for file server evaluation. In *Proceedings of the Fourth USENIX Conference on File and Storage Technologies (FAST '05)*, 2005.

[49] N. Zhu, J. Chen, T. Chiueh, and D. Ellard. An NFS trace player for file system evaluation. Technical Report TR-14-03, Harvard University, December 2003.