Generating a Jump Distance Based Synthetic Disk Access Pattern

Zachary Kurmas
Dept. of Computer Science
Grand Valley State University
kurmasz@gvsu.edu

Jeremy Zito, Lucas Trevino, and Ryan Lush
Dept. of Computer Science
Grand Valley State University
{*zitoj,trevinol,lushr*}*@student.gvsu.edu*

Abstract

As part of our research into improving synthetic, block-level I/O workloads, we developed an algorithm that generates a synthetic disk access pattern based on given distributions of both (1) sectors accessed and (2) "jump distances" between successive disk accesses. Generating a synthetic disk access pattern that maintains both distributions exactly is an NP-complete problem (similar to the Traveling Salesman problem). In this paper, we (1) discuss our approximation algorithm, (2) show that it runs in a reasonable amount of time, (3) show that it reproduces both distributions with reasonable accuracy, and (4) demonstrate its overall effect on the quality of block-level synthetic workloads.

1. Introduction

Storage systems must be evaluated with respect to the workloads that will be issued to them. These evaluation workloads can come from one of two sources: (1) actual traces of existing storage systems, or (2) synthetic workloads. Both approaches have strengths and weaknesses: Traces are more accurate, but are large, inflexible, and difficult to obtain. Synthetic workloads are flexible and can be represented compactly, but are often inaccurate. Our long-term research goal is to improve the accuracy of synthetic workloads, thereby allowing storage system evaluators to use them in place of workload traces.

Improving synthetic workloads requires two steps: We must first determine which properties a synthetic workload should share with the "real" workload for which it will substitute (henceforth called the *target* workload). We must then develop algorithms to produce synthetic workloads that maintain the desired properties. We addressed step 1 previously by developing a tool, the *Distiller* that automatically identifies those workload properties on which a synthetic workload should be based [7]. The Distiller relies on a library of properties from which to choose. This paper addresses step 2 by adding a needed property called "Jump Distance" to that library. The *jump distance* between two I/O requests is the difference between their starting sectors. This measurement approximates how far a disk's read/write head must travel before serving the next request.

Section 2 provides background and further motivation. Section 3 explains our approximation algorithm. Section 4 presents our results. Section 5 discusses future work; and, Section 6 concludes.

2. Motivation and background

In this section, we discuss (1) the motivations for using synthetic workloads, (2) our previous research that led us to focus on jump distance, and (3) related work.

2.1 Why synthetic workloads

Many researchers use workload traces to evaluate storage systems. Unfortunately, the use of traces has several limitations: (1) Traces are very large. (2) Because of privacy concerns, system administrators hesitate to make traces publicly available. Finally, (3) traces are inflexible: It is difficult to modify them to represent expected future workloads.

The advantages of using synthetic workloads are that: (1) They are compact. (2) Because they are specified using only high-level parameters, they do not contain any specific information.[1] Finally, (3) they are flexible.

The challenge is that synthetic workloads are rarely accurate: When used as part of an evaluation, most synthetic workloads produce different results than the *target workload*. This inaccuracy results from our incomplete understanding of which properties a synthetic workload must share with the target workload.

2.2 Previous work

Many existing workload synthesis techniques were developed for specific studies and are not necessarily generally

[1] We are hopeful that system administrators will be more willing to provide the parameters used to generate synthetic workloads than they are to provide entire, detailed workload traces.

useful. In response, we invented the Distiller to easily and automatically evaluate which existing generation techniques produce accurate synthetic workloads for new storage systems.

The Distiller does not "invent" workload properties. Instead, it takes as input a library of properties other researchers have already invented. The Distiller may conclude that no subset of its library will produce a sufficiently accurate synthetic workload. We must then add new properties to the library. The Distiller provides useful hints, but we must do most of the work. Based on its hints, we decided to add an improved jump distance property.

We use the difference between the starting sectors of successive I/O requests ($request[x].starting\_sector - request[x-1].starting\_sector$) as an approximate[2] measure of how far the disk head must physically move to reach the starting sector of the next request. We call this difference the *jump distance* between requests $x-1$ and x.

An accurate synthetic workload must also maintain the distribution of sectors used by the target workload. Generating a list of sectors that maintains given distribution of both sectors accessed and distribution of jump distances reduces to the NP-complete Hamiltonian Path problem. As a result, we must find an approximation algorithm.

2.3 Related work

The literature describes and evaluates many techniques for generating synthetic block-level I/O workloads [1, 2, 3, 5, 10], file-level workloads and application-level I/O workloads. Most focus on accurately synthesizing the workload's arrival pattern. Two of the few that focus primarily on block-level disk access patterns are Gomez's On/Off technique [2, 3] and Wang's PQRS model [10].

Hong et al. developed a generation technique that chooses several intervals of a real trace to represent the entire trace. The algorithm then generates a complete synthetic trace by concatenating copies of those intervals [4, 5].

This technique is highly effective in generating a synthetic arrival pattern; however it fared poorly when generating the disk access pattern also, because repeating the intervals chosen to represent the entire trace tended to add too much temporal locality to the workload. We expect that our jump distance-based generator can help address the limitations of Hong's clustering-based technique.

3. Our algorithm

Our problem is to take as input the distributions of (1) sectors accessed by the target workload, and (2) jump distances within the target workload; then produce a synthetic disk access pattern. Our approach is to transform the generation problem into an instance of the Hamiltonian Path problem, then apply a brute-force, depth-first search. If the search does not find a Hamiltonian path within a reasonable amount of time, we take the longest path found and use approximation techniques to complete the access pattern.

We begin by generating a list of I/O requests whose starting sectors exhibit the desired distribution. We then search for an ordering of the requests by choosing an initial I/O request (call it d_1) and executing a search of possible paths beginning with d_1. If we find a valid path of length n, we have found an ordering of I/O requests that maintains the desired jump distance distribution. Unlike the Hamiltonian Path problem, however, we must not only check that we use each I/O request (i.e., vertex) only once;[3] but, we must also be sure we do not use a given jump distance more often than specified by the input.

When performing a depth-first search, we must specify the order in which our algorithm examines the I/O requests. We consider five orderings: (1) *Ascending* (smallest to largest starting sector), (2) *Ascending absolute value* (closest sectors first), (3) *Descending* (largest to smallest starting sector), (4) *Descending absolute value* (most distant sectors first), and (5)*Random*.

Two factors make our brute-force approach reasonable given the theoretical complexity of the problem: First, not every Hamiltonian Path problem requires exponential time. For example, it is trivial to find a Hamiltonian path in a complete graph. We are hopeful that our jump distance problems will define graphs whose structures lend themselves to quick solutions. Second, a solution need not be exact to be useful. If the workload we are attempting to model has 10% of its accesses with a jump distance of 1 sector, it is acceptable for our solution to have slightly more or fewer. We will see that this flexibility allows us to find a solution quickly.

We employ two main approximation techniques: (1) We complete the path randomly after a given amount of backtracking, and (2), we allow a limited number of sectors and/or jump distances to be used more often than specified by their respective histograms.

In the first case, we specify how many times the depth-first search may backtrack without finding a longer valid path. (We call this parameter "*max stall*".) If the search backtracks too often, we terminate it, take the partially completed path, and place the remaining unused I/O requests randomly throughout the path.

In the second case, we specify "approximation factors" for sectors used and jump distance (for example, 10% and 15% respectively), and allow a sector to be used 10% more often than specified by the distribution of sectors accessed. Similarly, we allow a given jump distance to be used 15%

[2]See the extended version of this paper for details [8].

[3]Notice that we can use each I/O request once only; however, each sector may be accessed by more than one I/O request.

more often than specified by the jump distance distribution.

4. Experimental results

We evaluated our synthetic disk access pattern generator using two different target workloads. Space constraints allow us to discuss only one here: a one-hour trace of the OpenMail E-mail application. (See [8] for additional results.) It has a mean request rate of 304 I/Os per second and a mean throughput of approximately 2.33MB/s. The complete workload is described in detail in [6]. This workload's distribution of sectors accessed makes it an interesting example with which to test our generator. Over 78% of the I/Os are write requests. Almost half of the write requests access a small percentage of the sectors. The remaining writes and all the reads are distributed much more evenly over the sectors accessed. We suspect this distribution reflects differences in the accesses to the portion of the disk array used to store meta-data, and the portion used to store individual messages.

This workload is divided into 22 *logical units* (LUs) — sets of sectors that appear to clients as separate storage systems. We analyze and generate each LU separately, which makes sense when studying jump distance because only those jumps between requests to the same physical disk have a significant affect on storage system behavior.

In this section, we present three analyses. First, we evaluate how run time, input size, and vertex ordering affect the length of the partial Hamiltonian path found. Next, we evaluate the quality of the synthetic disk access patterns with respect to the input distributions. Finally, we evaluate the quality of the synthetic disk access patterns with respect to the accuracy of the resulting synthetic workload.

4.1 Length of partial path

Because the Hamiltonian Path problem is NP-complete, it is unlikely our depth-first search will find a Hamiltonian path in a reasonable amount of time. Three factors affect the length of a partial path: (1) the length of the trace analyzed, (2) the amount of time we allow the search to run, and (3) the order in which the algorithm examines vertices.

Input size: We had originally hoped that the graph defined by the input would contain so many Hamiltonian paths that one would be quickly discovered by a brute-force search. Figure 1 shows that this is not the case. We ran our algorithm several times requesting increasingly long sequences of I/Os[4] and allowed it to run until it had not made any progress for 10 million iterations. At that point, we terminated it and reported the length of the longest path relative to the desired number of I/Os. We ran the algorithm

[4]When requesting x synthetic I/Os, we provide as input an analysis of the first x I/Os of our target workload.

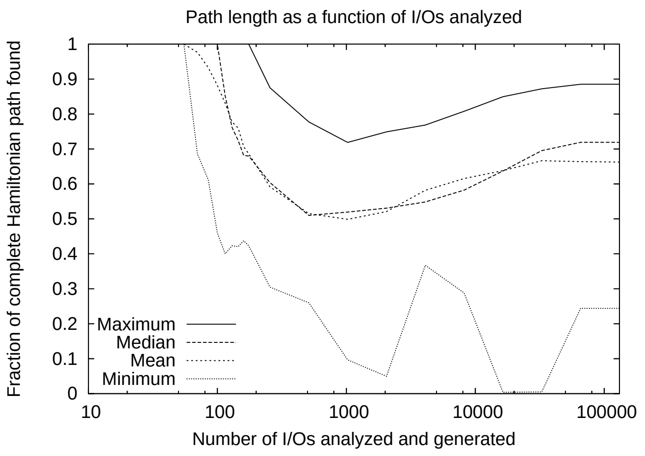

Figure 1. Our brute-force approach completes only when we desire fewer than 150 I/Os.

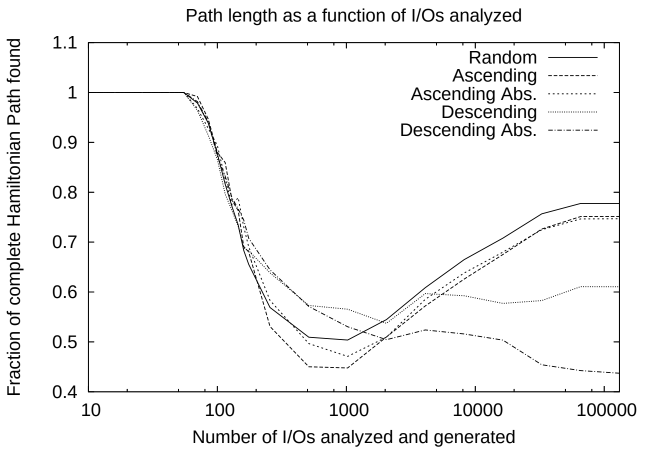

Figure 2. A random vertex order produces the longest partial Hamiltonian path.

separately on each LU. Figure 1 shows the maximum, minimum, mean, and median length of the longest path over all LUs and vertex orderings. We can see that, for any LU or vertex ordering, the algorithm is not able to complete quickly once the desired number of I/Os reaches about 150.

Note that the length of the partial path found (relative to the number of I/Os desired) dips for lengths of approximately 1,000 I/Os, then rises again. We found that the density of the graph is correlated to the length of the path found, which makes sense because it is easiest to find Hamiltonian paths in extremely dense and extremely sparse graphs.

Vertex ordering: The order in which the depth-first-search examines vertices in the graph affects how long the algorithm takes to find a Hamiltonian path. If the vertices are searched in an extremely fortuitous order, then the algorithm will find a Hamiltonian path without any backtracking. We experimented with five different orderings (explained in Section 3). Figure 2 shows how the vertex ordering affects the length of the path that is generated before the algorithm goes 10 million iterations without progress.

The random vertex ordering does best when the desired number of I/Os is large. We suspect the random vertex order does best because it is less likely to "use up" sectors and jump distances early. For example, when configured to use the "Ascending absolute value" sort, the depth-first search always considers the I/O requests with the current starting sector first. As a result, the depth first search begins building a path by repeating the initial sector until either (1) that sector is used as many times as specified in the input distribution, or (2) the jump distance 0 is used as many times as specified in the input distribution. Soon afterward, the jump distance of 1 sector is "used up". Small jump distances are easy to match (i.e., there are more pairs of requests 1 sector apart than there are pairs of sectors 2,435,812 sectors apart); thus, the two "ascending order" searches build long paths quickly. However, once the common jump distances have been used, the rate of progress slows. In contrast, by examining vertices in a random order, the depth-first search is more likely to consider less common jump distances first. The "descending order" sorts also have this property; but because larger jumps are harder to match, the search makes less progress before reaching the backtracking limit.

Maximum stall: Because the depth-first search cannot find an exact answer in a reasonable amount of time, we must terminate the algorithm at some point and complete the list of I/Os using approximation techniques. For our first set of experiments, we stopped the algorithm after it had not made any progress in 10 million iterations. Upon further investigation, however, we found that we can achieve almost the same result while terminating the algorithm much earlier. The path after 100,000 stalls was 98% the length of the path after 10 million stalls. From these results, we have chosen to run our depth first search only until it goes 1 million iterations without producing a longer path.

4.2 Distribution quality

Our results from the previous section show that we cannot use the depth-first search alone to generate a synthetic disk access pattern. In this section, we evaluate two methods of creating a complete synthetic disk access pattern: "finish randomly" and "approximation factor."

In both cases, we evaluate the quality of the resulting synthetic disk access pattern by comparing its distribution of jump distance to the input from which it was generated. We quantify the difference between two distributions of jump distance using the weighted average of the percent difference between each pair of corresponding bins in the two histograms. For example, we calculate the percent difference between bin x for the target workload and bin x for the synthetic workload as follows: $(\frac{|h_{target}[x]-h_{synthetic}[x]|}{h_{target}[x]})$.

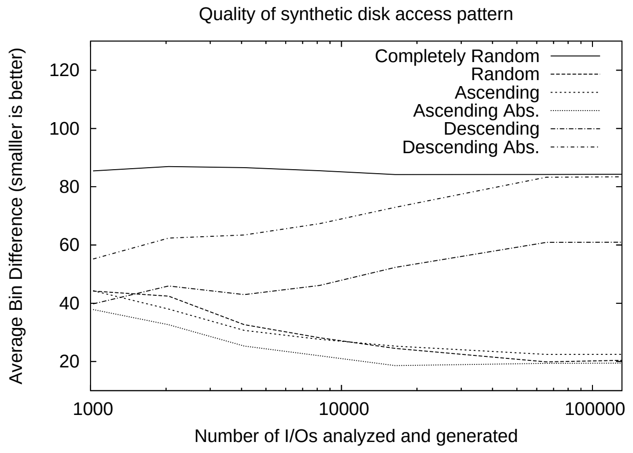

Figure 3. Ascending and random vertex orderings produce best quality synthetic access patterns.

Because there is no single, universally accepted metric for quantifying the difference between distributions, we considered five different metrics. All five exhibited similar trends; so, in the interest of space and clarity, we choose to present only the average bin difference.

Finish randomly: As described in Section 3, the first option for generating a complete disk access pattern is to wait until the search has not made any progress for the specified number of iterations, then generate the complete disk access pattern from the partial path by placing the unused I/O requests randomly throughout the path.

Figure 3 shows how the quality of our synthetic disk access patterns compare to a disk access pattern produced by choosing the sequences of sectors randomly without considering jump distance. Three of our five vertex orderings are considerably more accurate than the "completely random" access pattern. The descending and descending absolute value sort orders produce progressively lower quality access patterns as the length of the generated trace increases. We suspect this lower quality is a result of the correspondingly short partial paths from which the final disk access pattern is derived. (See Figure 1.)

Approximation factor: As described in Section 3, our second option is to allow the depth-first search to use sectors and/or jump distances more often than they appear in the target workload. We investigated different combinations of sector and jump distance approximation factors to see (1) which combinations allowed our depth-first search to finish within a reasonable amount of time and (2) how the synthetic disk access patterns produced compared in quality to those produced without any approximation factors.

Table 1 shows typical results using the random vertex ordering. Raising either approximation factor increased the length of the partial path found, but less than we had expected. The depth-first search does not find a 98% complete

Jump Dist.	Sector Count Factor							
Factor	1.00	1.01	1.05	1.10	1.25	1.50	1.75	2.00
1.00	77.4 (18.6)	77.4 (18.7)	77.6 (18.3)	77.9 (18.0)	81.2 (15.0)	86.5 (10.5)	87.8 (9.5)	91.9 (6.6)
1.01	77.1 (19.1)	77.2 (19.1)	77.8 (18.1)	78.4 (17.7)	81.3 (15.0)	86.6 (10.7)	88.0 (9.5)	92.1 (6.5)
1.05	77.8 (18.9)	77.8 (18.8)	78.1 (18.5)	78.7 (17.9)	81.8 (15.2)	87.2 (11.0)	88.8 (9.8)	92.5 (7.5)
1.10	77.7 (19.6)	77.7 (19.7)	78.4 (18.9)	79.1 (18.3)	82.6 (15.5)	88.1 (11.8)	89.8 (10.6)	93.8 (8.0)
1.25	79.7 (19.8)	79.7 (19.7)	80.5 (19.3)	80.1 (20.0)	85.4 (17.0)	91.1 (13.7)	92.8 (12.9)	95.9 (12.6)
1.50	83.5 (19.8)	83.4 (19.9)	84.0 (19.8)	84.5 (20.2)	88.9 (19.5)	95.1 (18.2)	98.0 (20.1)	99.4 (20.1)
1.75	84.2 (23.0)	84.3 (23.0)	85.5 (22.7)	87.0 (23.0)	91.8 (22.0)	98.3 (25.1)	98.9 (25.8)	99.6 (25.8)
2.00	91.0 (23.8)	91.1 (23.8)	91.6 (24.2)	92.3 (24.2)	98.5 (23.3)	100.0 (24.4)	100.0 (24.8)	100.0 (24.6)

Table 1. Length (left, as a percentage) and quality (in parentheses, lower is better) of disk access pattern generated using approximation factors.

Hamiltonian path until at least one of the approximation factors is 1.5. Fortunately, increasing the approximation factor decreases the quality of the synthetic disk access pattern by at most 50%.

4.3 Synthetic workload quality

Measuring the quality of the output distributions is useful because it tells us how well our generation algorithm reproduces the pattern it is designed to reproduce. However, the output distributions are merely "cosmetic" similarities between the target and synthetic workloads: They tell us nothing about the usefulness of the resulting synthetic workload.

In this section, we measure the usefulness of our synthetic disk access generator in producing accurate synthetic workloads. We will issue both the target and synthetic workloads to a storage system and measure the resulting distribution of response time (the percentage of I/Os that complete quickly and the percentage of I/Os that complete slowly). Then, we use three metrics to quantify the difference between the two distributions:

Mean response time: A synthetic workload should have a mean response time similar to the workload it models; however, two workloads with very different behaviors can have similar mean response times. We express this metric as the percent difference between the mean response times of the compared workloads.

Root-mean-square: RMS is the root mean square of the horizontal distance between cumulative distribution functions of response time. We express this metric as a percentage of the mean response time of the target workload. RMS is the metric used in the related work (e.g., [1], [9]).

Log area: The log area metric is the area between two CDFs plotted with a log scale on the x-axis. This metric more accurately reflects the similarity of the workload's overall performance, but de-emphasizes differences most noticeable to the user.

We use the Pantheon disk array simulator to simulate the execution of our workloads [11]. Pantheon simulates disk arrays comprising several disks connected to one or more controllers by parallel SCSI busses. The controllers have large non-volatile-RAM caches. (This general architecture is similar to the FC-60 used in [7].) Pantheon provides many additional configuration parameters including number and type of component disks, and size of cache.

Workload	MRT	RMS	Log area
Completely Random	15%	32%	6%
Ascending	1%	3%	1%
Ascending Abs.	9%	18%	4%
Descending	3%	6%	2%
Descending Abs.	13%	27%	6%
Random	11%	23%	5%

Table 2. Accuracy of synthetic workloads.

In order to observe the effects of our synthetic disk access pattern only, we modified the target workload to be read-only, have a constant request size of 8KB, and issue precisely 328 I/Os every second. Attempting to evaluate our disk access pattern in the context of the original workloads is difficult because the correlations between sectors accessed and whether those accesses are reads and writes have a large effect on storage systems with a write back cache (like the FC-60). Fixing the values for those I/O parameters not under study eliminates this problem [1].

Table 2 shows the performance of our modified workload as compared to synthetic workloads generated using various sort orders. For comparison, we also show the performance of a workload in which the sequence of sectors accessed is chosen randomly without regard to jump distance.

We can see that, although using a random sort order produces the longest Hamiltonian path, it does not produce the best synthetic workload from a performance standpoint. Neither does the ascending absolute value sort order, which produced the disk access pattern with the lowest average bin difference. We suspect the differences in performance are caused by differing amounts of temporal locality in the synthetic workload. Our algorithm is designed to reproduce only jump distances; however, different sort orders produce differing amounts of temporal locality (as explained in Section 4.1). Pantheon's log files showed that synthetic workloads based upon a random sort order had cache hit rates that are 2.7% lower than that of the target workload. In contrast, the most accurate synthetic workload has a cache hit rate close to that of the target workload. Searching sectors in ascending order implicitly maintains temporal locality by considering the same sectors first. The performance of the workload based upon a descending sort order falls in the middle: Again, examining the same sectors first produces temporal locality; however, because the partial paths are shorter, the distribution of jump distance is not as accurate as the workload based on an ascending order.

The results above show how the generation of accurate synthetic workloads still requires some trial and error. In the case of our E-mail workload, two reasonable metrics (partial path length generated and average bin difference) suggested using the random and ascending absolute value sort orders, respectively. Further investigation, however, suggested that the ascending sort order produced the most accurate synthetic workload. This situation further supports the need for a tool like the Distiller, which can evaluate the effectiveness of several different generation techniques.

5. Future work

There are many aspects of our synthetic disk access generator that we have yet to investigate. Most immediately, we plan to investigate techniques for specifying several small graphs instead of one large graph, which will hopefully increase the length of the partial paths found as well as reduce the overall memory requirements. One idea, similar to analyzing each LU separately, is to analyze ranges of the storage system's address space separately, then use a transition matrix to choose the different address ranges when generating a synthetic access pattern.

We also plan to add our generator to the Distiller's library of workload analysis and generation techniques. Once it is a part of the Distiller's library, we can evaluate the usefulness of our jump distance generator when combined with other synthetic workload techniques, such as those that accurately reproduce arrival time patterns, or those that address correlations between I/O request parameters.

6. Conclusions

We have shown that, in a reasonable amount of time, we can generate a synthetic disk access pattern that maintains both a specified distribution of sectors accessed and a specified distribution of jump distance within a reasonable margin of error. Not only is the resulting synthetic workload "cosmeticly" similar to the workload it is intended to model, but it is also more accurate than synthetic workloads using more traditional generation techniques.

Acknowledgments

This research was begun as part of Hope College's summer REU program, and continued through a grant from the Research and Development office at Grand Valley State University. In addition to our supporters, we wish to thank our colleagues at Hope College for a wonderful summer, Carl Strebel for top-notch system support, and our friends in the Storage Systems Department at HP Labs, who provide encouragement and feedback.

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. 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.

[3] 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.

[4] 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.

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

[6] K. Keeton, A. Veitch, D. Obal, and J. Wilkes. I/O characterization of commercial workloads. In *Proceedings of 3rd Workshop on Computer Architecture Support using Commercial Workloads (CAECW-01)*, January 2001.

[7] 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.

[8] Z. Kurmas, J. Zito, L. Trevino, and R. Lush. Generating a jump distance based synthetic disk access pattern. Technical Report GVSU-CIS-2006-01, Grand Valley State University, 2006.

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

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

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