Synthesizing Representative I/O Workloads for TPC-H

Jianyong Zhang* Anand Sivasubramaniam* Hubertus Franke† Natarajan Gautam*
Yanyong Zhang‡ Shailabh Nagar†

*The Pennsylvania State University, University Park, PA 16802.
†IBM T. J. Watson Research Center, Yorktown Heights, NY.
‡Rutgers University, Piscataway, NJ.
Contact: *anand@cse.psu.edu*

Abstract

Synthesizing I/O requests that can accurately capture workload behavior is extremely valuable for the design, implementation and optimization of disk subsystems. This paper presents a synthetic workload generator for TPC-H, an important decision-support commercial workload, by completely characterizing the arrival and access patterns of its queries. We present a novel approach for parameterizing the behavior of inter-mingling streams of sequential requests, and exploit correlations between multiple attributes of these requests, to generate disk block-level traces that are shown to accurately mimic the behavior of a real trace in terms of response time characteristics for each TPC-H query.

1. Introduction

The data-centric nature of commercial applications/services and the growing disparity between processing and I/O speeds continue to motivate the need for designing and deploying scalable high-performance storage subsystems. The performance of these storage systems is highly dependent on the workload, and an indepth understanding of the demands imposed by the workload is critical for system design and implementation, and subsequent optimization once it is deployed. Workload characterization is thus at the core of any systems design and optimization process, and our focus here is on characterizing the demands imposed by an important commercial transaction processing workload (TPC-H) on the storage subsystem towards generating a synthetic trace of disk block-level requests.

The load on the disk subsystem can be envisioned as a sequence of block-level requests that are a result of the application-level requests filtering through the OS software stack. There are two ways of inducing this load on a target disk subsystem for evaluation: (i) using traces of the block level requests that have been collected on an alternate system, and (ii) using a synthetic workload generator that can closely mimic the behavior of the real load. Though attractive in terms of reflecting realistic workload behavior, traces have been widely pointed out [6, 18, 9] to have the following drawbacks:

- Traces may not be easily obtainable since system administrators may not want to slow down their production environment by tracing overheads, or they may not want to part with their traces (even if they have one) for security and/or privacy reasons.
- Traces can become very large with the systems being traced running for several days to get representative behavior. Storage space and distribution costs can make these long traces less attractive.
- Since each trace represents the execution of one workload on one system, it may be difficult to get statistical confidence in the evaluation. It is not easy to gauge if a performance glitch is an inherent feature of the underlying system or whether it is because of some anomalous behavior (e.g. an extraneous artifact imposed a transient load) in the trace.
- With a trace, it becomes very difficult to change workload behavior (in anticipation of future workloads that may be different from what has been traced). We may want to change arrival rates, burstiness in the load, or the pattern of sectors being referenced in order to stress the system under different conditions.
- Traces do not allow isolating the influence of any one parameter. When performance debugging/optimizing, one needs to understand what artifact (e.g. was it the burstiness or was it the randomness of sectors requested?) of the workload caused a problem.
- When characterizing the workload, we gain a lot more understanding about the workload that a blind use of a trace may not provide. Correlation (i) between the values of a particular parameter over the trace, or (ii) between parameters themselves, is one such important characteristic that is not readily available in a trace. These correlations can be extremely valuable in systems design, e.g. correlations between the sequence of sectors that is requested can give ideas for a prefetching mechanism, or correlations between arrival times and sectors requested may suggest better disk scheduling algorithms.

  This indepth understanding of workload behavior becomes extremely important for online tuning (or self-managing/optimizing systems that is gaining a lot of industrial interest recently due to the high costs of managing/tuning systems). If the characterization can provide good insights on what to expect in the near future, the underlying system can be adapted for the ideal performance when those workload conditions evolve.

Consequently, a lot of effort has been expended on developing synthetic workloads that can mimic the behavior of a real application. The main concern when synthesizing these workloads is to make them realistic and representative of the actual demands imposed on the system, while not taking away the above fundamental advantages

TPC-H is an important decision support workload supplied by the Transaction Processing Council [25] that contains 22 queries which analyze relational tables to aid in decision making for commercial enterprises. As is to be expected with database workloads, disk I/O is typically the most limiting factor in the performance of these TPC-H queries, which many prior studies [28, 23, 3] have pointed out. Our study uses traces collected from a TPC-H power test on a commercial IBM Netfinity 8-way SMP server with 15 disks running EE DB2 [12]. Using this trace, this paper conducts an extensive, and to our knowledge the first complete, analysis of TPC-H disk block-level requests to accurately synthesize the *arrival pattern* (inter-arrival times) and *access pattern* (location of the sector on disk and the request size in terms of the number of sectors) of each query in the workload.

The main areas in which this paper contributes to the state-of-the-art on this topic include:

- We study the suitability of different approaches to analyzing the arrival and access patterns for TPC-H. We investigate the correlations between the values of a single parameter in the requests, as well as the correlations across the parameters, and show how this can be useful in synthesizing the arrival and access pattern for the queries. An important contribution is a methodology for capturing the sector reference pattern of several inter-mixing (and possibly sequential) streams that can model correlations between multiple arrival and access pattern parameters, while being generic enough to capture diverse behaviors exhibited by these queries.

- The results of our analysis shows certain results that differ from those in previous studies. Specifically, we show that simple IID (independent and identically distributed) distributions can capture TPC-H I/O inter-arrival times, which is in contrast to more bursty and self-similar behavior observed in other workloads. We also show that despite there being some sequentiality in the disk blocks that are referenced, there is a fine grain inter-mixing of sequential streams making it imperative to conduct a smarter analysis of the requests to extract such patterns in some queries.

The rest of this paper is organized as follows. The next section summarizes related work in the area of I/O characterization, both in the scientific and production workload area. The trace collection mechanism, the simulation target used for validation, the metrics used to evaluate accuracy, and the overall synthesis methodology is outlined in section 3. Section 4 examines inter-arrival time characteristics. Subsequently, in section 5 we examine access pattern characteristics and use correlations between parameters to synthesize the requests for the queries. Finally, section 6 summarizes the contributions and outlines directions for future work.

2. Related Work

Workload characterization has long been recognized as a critical requirement for systems design with several prior investigations into synthesizing workloads for specific system artifacts. Our focus here is on the disk system, and the reader is referred to [6, 18, 9] for the motivation behind synthesizing representative I/O workloads, and why this is a non-trivial problem.

Several prior studies [16, 20, 21, 24, 26] have analyzed the I/O behavior of parallel scientific applications, for tuning - either offline or online - parallel file systems. When we move to non-scientific workloads, the impact of spatial reference pattern (particularly sequentiality) has been studied in [17, 14] for an email server. Ruemmler and Wilkes [22] observed burstiness in I/O requests in two actual environments - a time-shared system and a file server. Gomez and Santonja [8] confirmed the self-similarity in I/O arrivals for these traces suggesting reasons for such behavior. Further, [6] goes on to suggest that I/O arrivals may not match a Poisson process and may not be independent or exponentially distributed.

In the context of database/decision-support workloads, a more general investigation of the I/O throughput for TPC-H and TPC-C queries has been conducted in [3] on the IBM NUMA-Q system. But this does not give much insight on the temporal or spatial I/O properties of the queries, except reiterating that reads are dominant in the execution. TPC-D, a pre-cursor to TPC-H, has been analyzed in [15], at a higher level to examine request sizes and arrival phases. A very detailed evaluation of 10 production workloads, in addition to TPC-C and TPC-D, has been undertaken in [10] but the focus has been mainly on buffer management issues (caching, prefetching and write buffering). At the same time, the authors in [10] acknowledge that the TPC workloads are representative of the load on production environments, thereby underlining the importance of understanding (and synthesizing) their behavior.

Most previous studies for TPC-H have studied an isolated artifact at a time (possibly for a specific system optimization). The recent work by Kurmas et al. [18] is the only other study, to our knowledge, that has attempted to characterize several attributes of this workload at a time. Their goal, however, is to automate the process of generating synthetic workloads, and the TPC-H trace that they use as one of the examples is much more simple (with extensive sequentiality in the sectors referenced that the resulting model is much simpler) than the more irregular behavior that we observe for some of the queries in our system. Further, the methodology that we propose for addressing this behavior can capture different kinds of sequential interleaving, and can automatically degenerate to a model for strict sequentiality with appropriate parameter values.

3. Synthesis Methodology and Experimental Setup

Trace Collection: The TPC-H trace on which we conduct the analysis has been collected on an IBM Netfinity SMP server with eight 700 MHZ Pentium III processors, 2.5GB memory and fifteen disks. Of these fifteen, three are system disks, with the other twelve used to hold TPC-H data. The data disks are 36GB 7200RPM SCSI disks, and are connected to two SCSI controllers.

The database engine used for the study is IBM's DB2 UDB EE Version 7.2 [12] running on Linux 2.4.17. Typically, several kernel patches are needed beyond the vanilla Linux kernel to obtain good TPC-H performance. We used several patches including `lse04-rc1` (developed at IBM [19]), that integrates several patches such as the `highmem` patch, the `kio_pagesize` patch, the `smptimers` patch, light-weight `kiobuf` and a patch to remove the big I/O request lock. In addition, we also developed our patch to collect the I/O trace at the SCSI driver level. Each trace record consists of a timestamp (at mi-

crosecond granularity), the major and minor device numbers, the type of operation (read/write), the starting-sector and number of sectors requested, and the pid of the process issuing the request. We have compared the response times for each query with and without our patch, and found that there are negligible differences to verify that our patch has little interference on the workload. The resulting I/O trace consists of 18 million I/O references over all queries.

TPC-H Configuration: We use the TPC-H power test that measures the throughput/response times of a sequence of 22 queries as recommended by the Transaction Processing Performance Council (TPPC). A smaller version (30 GB) of the TPC-H implementation whose results for a 100 GB dataset were submitted by IBM [13] in February 2002 to TPPC is used in this exercise. All data disks are configured to use raw I/O.

The decision-support workload queries of TPC-H require the disks to hold 8 tables, and the names of these tables together with their populations (in brackets) are: Lineitem (6,000,000), Orders (1,500,000), Part (200,000), Partsupp (800,000), Customer (150,000), Supplier (10,000), Nation (25) and Region (5). The disk partitioning is based on these tables with each disk being partitioned into 7 logical partitions. With Lineitem and Orders being much larger than the other tables, we use 1 partition each (four in all) to hold the table Lineitem, the index of Lineitem, the table Orders, and the index of Orders. The other tables are grouped in two partitions, one for data, one for index. The last partition holds the Temp table, that the database engine uses for intermediaries.

In our trace, we found that the number of I/O requests for queries Q2, Q11, Q16 and Q22 is not very significant (typically less than 3000 requests to a disk). Even though we have conducted our analysis on these queries as well (and found our synthesis to match the response time characteristics of the original trace), the statistical confidence of our synthesis is not very high because of the fewer samples. Further, the accesses in Q13 are mainly to the Temp table. Hence, *we primarily examine the other 17 queries.*

Synthesis Methodology/Outline: We take the trace and conduct extensive analyses to completely characterize the (i) arrival pattern (*inter-arrival times*), and the (ii) access pattern (the other attributes of each request including the *start sector/location*, request *size* in terms of the number of sectors, and the request *type* as to whether it is a read or write). We first examine the arrival times in isolation and show that they can be assumed to be IID, and independently validate this assumption (by picking the other parameters from the original trace). We then examine the location parameter and show that queries fall into 3 categories based on the regularity/sequentiality in this parameter. We propose a new model of capturing these 3 location characteristics in a unifying way, and validate its accuracy in isolation (by taking the other parameters from the trace). Armed with all these intermediate results, we finally present a unified methodology that exploits correlations between attributes to generate a synthetic trace for each category. When validating the accuracy of a synthetically generated trace, we are specifically interested in matching its response time (for a request) characteristics with those of the original trace, by using them both to drive a disk subsystem simulator.

Simulator configuration: The simulator that we use for empirical validation of synthesis accuracy is DiskSim [7], that incorporates detailed models of the disk drives, controllers, caches and I/O channels. The parameters for the disks that are modeled (resembling Quantum Atlas disks) are given in Table 1 and we use two controllers that are based on the NCR 53C700. Please note that the parameters for this simulator itself are not very important since we are more interested in how close is the performance of the synthetic workload with that for the original trace, rather than the absolute performance issues. Still, in one set of experiments we also vary some of these hardware parameters to show that our synthetic workloads still produce similar performance characteristics as the real trace.

Capacity	22.4GB
Rotation Speed	7200 RPM
Data Surfaces	15
Cylinders	10042
512-Byte Sectors	44920468
Zones	24
Sectors/Track	229-334
Interface	Ultra-2 SCSI
4M Cache, 20 Segments, and 1 write segment	
Track Sparing/Reallocation	
Elevator Algorithm	

Table 1. Disk Parameters in the Simulator

Metrics: In addition to using different statistical error measures that can be associated with each of the models that are used when generating the synthetic workload from the trace, we also quantify the differences in the actual performance (response time of each request) characteristics between that for the synthetic model and the real trace on the simulated disk subsystem. Response time characteristics for the queries are very important from the user's perspective, and have been extensively used in related studies [6, 9, 22, 18] to evaluate the accuracy of synthetic workload generation. We use similar metrics in this study.

We track the response time for each I/O request (from the original and from the synthetic trace). We plot a cumulative density function (CDF) of these response times for the original and synthetic trace, and we can use the Root-Mean-Square (RMS) error of the horizontal distance between these two CDF curves [22], or the normalized Root-Mean-Square (nRMS) error which is $nRMS = RMS/m$, where m is the average response time for the original trace. We are specifically targeting a synthetic workload trace that has an nRMS of around 0.20 or lower.

4. Synthesizing Arrival Characteristics

We begin by examining the synthesis methodology for arrival (inter-arrival time) characteristics of requests in the trace to all 12 disks (one can easily generate per disk arrivals from these results). In the following discussion, we first investigate the modeling issues (and errors) using statistical techniques that paints a fairly good picture about the accuracy of our models. Subsequently, we actually inject the workload generated by this model into the simulator and validate its accuracy further by comparing response time characteristics with those for the real trace.

Studying Correlations: Before deciding on a modeling strategy, it is first necessary to understand the correlations of the inter-arrival times using auto-correlation functions (ACF), since that can give a good indication of how to proceed. If there is very little correlation, then it may not be a bad choice to assume that inter-arrival times are independent identically distributed (IID), and proceed along the lines of fitting statistical distributions. On the other hand, if there are strong correlations, either near-term (with successive/close-by inter-arrival times) or longer-term correlations, then time-series or self-similar models may be needed. When we plotted the auto-correlation functions of the inter-arrival times, we found that there were not strong correlations between inter-arrival times, thereby suggesting

the possibility of IID assumption (the reader is referred to [27] for a detailed analysis).

Time-series Models: We next used the RPS toolkit [4] to fit different time-series models - auto regressive (AR), moving average (MA), ARMA, ARIMA - together with a fractionally integrated ARIMA model (ARFIMA) which is useful for modeling long-range dependence and self-similarity, a last value model (LAST), a windowed average model (BM), and a long-term average model (MEAN)- for the inter-arrival times. The reader is referred to [2] for an in-depth treatment of these models.

The results of this analysis reveal that the ARMA, ARIMA and ARFIMA models are quite inaccurate in modeling the inter-arrival times, ruling out the possibility of self-similar/long-range dependence models. While the other models are doing better than these, giving absolute errors that lower, the errors are still intolerable - in most cases they are comparable or even larger than the mean inter-arrival time. It is only in Q1, Q6 and Q15 do we find their errors tolerable. These results re-confirm our hypothesis from the previous correlation study that time-series or self-similar models may not be very suitable for generating the inter-arrival times for TPC-H queries (unlike their suitability to other workloads that was pointed out by others in [8] and [26]).

IID Distributions: The above observations from the previous results suggest the possibility of arrivals being independent and identically distributed. With these suggestive hints, we next attempt fitting different distributions for each query as follows:

- The first step is to identify the distribution family to use upon which the parameters of the distribution can be estimated. We mainly considered *hyper-exponential* (that can capture high variance behavior), *normal* (that can model lower variance), and *pareto* (that can model bursty/heavy-tailed behavior) distributions, though this is by no means a comprehensive list of all that we tried. Typically we tried all of them for each query, though in some cases we had some suggestive hints from previous results. For instance, in queries Q1, Q6 and Q15, the smaller errors for the time-series models suggest a distribution with smaller variance (achievable with an appropriate variance value for a normal distribution).
- The next step is to estimate the parameters for the model (e.g. mean and variance for a normal distribution, etc.). We used the Maximum Likelihood Estimation (MLE) [5] for fitting a distribution to a set of datapoints The idea behind this method is to derive the parameter values of the distribution that would produce the highest probability of sampling the given set of values. We use MLE to estimate the distribution parameters for this exercise. While the parameters are straightforward to obtain for the normal and pareto distributions, the hyper-exponential distribution requires more work since it is a mixed distribution. In this case, we use the Expectation Maximization (EM) method which is an iterative implementation of maximum-likelihood estimation. We used an implementation of the EM-algorithm [1] that can be used for fitting phase-type distributions.
- Finally, after finding the parameters for the distributions, we need to study how well they fit the original dataset. Since we are dealing with continuous time series data here, we use the Kolmogorov-Smirnov (K-S) test to evaluate the Goodness-of-Fit. For each distribution, we calculate the maximum distances between the CDF of the fitted distribution and the sample's empirical distribution with the K-S test, and then choose the fitted distribution that has the minimal distance. We ensured that this distance is less than 0.1 for all queries.

The reader is referred to [27] for the detailed results comparing the inter-arrival times of the original trace and that generated by our synthesis for each of the queries. Overall we found that the pareto distribution was not really doing well across the queries, that is perhaps another confirmation of the absence of burstiness and self-similarity in the requests mentioned earlier. As expected, in the queries with low variance (Q1, Q6 and Q15), the normal distribution gives the closest fit. Of the rest, the exponential distribution suffices in Q5. While exponential does a fairly good job in Q8, Q17 and Q20 as well, the 2-phase hyper-exponential gives a better Goodness-of-fit for these. The 2-phase hyper-exponential does a good job in the case of queries Q3, Q4, Q7, Q9, Q10, Q14, and Q18, while a 3-phase version is needed for Q12 and Q21. We also conducted an experiement wherein we substituted the inter-arrival times of the original trace with those from our synthesis, and compared the response time characteristics (CDF) with those of the original trace. The response times closely match [27], again reiterating the validity of such synthesis (suggesting there is some merit behind the IID assumption).

5. Synthesizing Access Characteristics and Trace Generation

We now move on to synthesizing the attributes of each access and generating the complete synthetic trace. These attributes include the (i) *type* of the operation (whether a Read or a Write), (ii) starting sector (which we will refer to as *location* henceforth), and the (iii) *size* of the request (in number of bytes or sectors to read or write from the starting sector). We refer to these, in addition to the arrival time, as the *primary attributes*, since these are directly the attributes of each trace record. At the same time, there could be *secondary attributes* that we can derive from these primary attributes, which may make it easier to synthesize the workload, and from these be able to generate the primary attributes themselves. For instance, sequentiality in the sectors that are accessed is a secondary attribute that can be gleaned by analyzing the trace, and may be easier to abstract/synthesize. Once we have a reasonable estimate of sequentiality (for instance, 30% of the requests are sequential), then one may be able to use such information in generating values for the starting sector (the primary attribute) that adheres to such sequentiality (the secondary attribute). The results from the previous section on the arrival pattern, and the characteristics of the access patterns to be discussed shortly, jointly provide the information that we need to generate the synthetic trace for each query.

We specifically focus the results from our study for a single disk (note that a request does not span multiple disks since there is no hardware RAID and DB2 is explicitly managing a set of individual disks). Further, DB2 spreads the load rather evenly across the disks, and we have found that the results are very similar across the disks. We also concentrate the experimental results on those for the lineitem partitions whose I/O accesses dominate over the rest of the operations.

TPC-H being a decision-support (OLAP) workload, is inherently read-dominated, with there being very few writes (primarily to the "Temp" partition), making write performance not that critical unlike other workloads [11]. Consequently, we fix the type (primary) attribute to "Read" in this exercise, and this simplification does not affect the accuracy of the results to be provided. The following discussions de-

tail our synthesis approach for the other two (primary) access pattern attributes - location and size.

5.1. Methodology for Location and Size Synthesis

Unlike the arrival characteristics, we encountered several more complications/difficulties when attempting to synthesize location and size information. We tried different approaches including distribution-fitting, correlation analysis, or even simple high level characterization such as sequentiality specification, etc. In the end, we found that a combination of (a) high-level categorization of the queries, (b) further identification and characterization of secondary attributes from the original trace, and (c) exploiting correlation information across attributes, was needed to get a fairly accurate synthesis. Before presenting the details of the synthesis, we briefly give a high-level overview of these issues to understand why each is important.

Categorization of Queries: I/O access patterns have often been abstracted at the high-level for easier understanding in terms of sequentiality or randomness in how the locations (starting sectors) are accessed. Such information can be very useful for devising prefetching mechanisms, buffer management algorithms, and other optimizations. When we examine the location attribute of TPC-H queries, we can broadly put them in 3 categories. Figure 1 shows representative examples from these 3 categories, by plotting the location (on the y-axis) for each access record in the trace in increasing time order (on the x-axis). We find that queries seem to be either very well-behaved adhering rigidly to sequential behavior (referred to as Category 2), or falling to the other extreme of having a very random access pattern (referred to as Category 3), or exhibiting some amount of regularity/sequentiality though not as rigidly (referred to as Category 1). Examining these graphs for the 17 queries, we find Q4, Q10 and Q14 falling in category 1, Q1, Q3, Q5, Q6, Q7, Q8, Q12, Q15, Q18, Q19, and Q21 falling in category 2, and Q9, Q17 and Q20 falling in category 3. Such a categorization helps us develop a different synthesis strategy for each (e.g. a sequentiality biased synthesis for category 2 versus a more random set of requests for category 3).

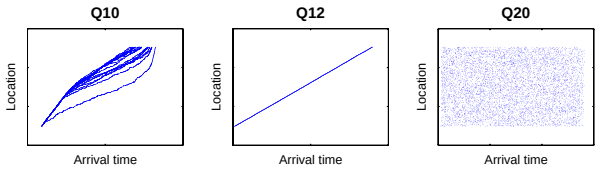


Figure 1. Access Pattern characteristics of the three categories (e.g. Q10 for Category 1, Q12 for Category 2, and Q20 for Category 3). The plots show the locations referenced by the sequence of requests for each query.

Secondary Attributes: As mentioned earlier in this section, identification of certain secondary attributes, which may be easier to synthesize since they can better capture the vagaries in the trace, can facilitate the generation of the primary attributes subsequently. For instance, despite the visual appearance of regularity/sequentiality in Figures 1, there are more subtle variations when one examines these characteristics at a much finer granularity. In other words, at a finer resolution, the requests may not be that sequential due to several reasons, one of them being that the database engine multiplexes several activities/threads that are concurrently accessing different I/O regions. For instance, while one thread of a query is scanning a table, another thread could be performing joins, or the database prefetcher could be bringing in blocks ahead of their need. Still, there could be sequentiality within an activity/thread before another activity takes over, and their inter-mingling can make location synthesis a lot harder. One can visually discern this behavior for Q10 in Figure 1, though it also happens at a finer granularity (and to a lesser extent) for queries in the second category. While there has been some brief mention ([10, 14]) of understanding such inter-mingling sequentiality, there is no previous study that has formally attempted to define this inter-mingling sequentiality, characterizing it for TPC-H, and using those results to generate a synthetic trace. To capture this inter-mixing sequentiality, we introduce two concepts - a *run* and a *stream*.

A run is a strictly sequential set of I/O requests until this sequentiality is broken by another request. More formally, a *run* can be defined as a sequence of I/O requests (s_i, l_i), with s_i and l_i denoting the start sector and number of sectors requested by the i-th request, where the next request in the sequence starts accessing from where the previous left off (i.e. $s_{i+1} = s_i + l_i$). For example, if we look at Table 2, which shows a sequence of requests from the trace for Q10, requests #12, #13, and #14 are part of a run, before that run is broken by request #15 which accesses a location (sector 108909) that is not adjacent to where #14 left off (i.e. sector 110253). The column denoted as RunID in this table, shows the start of the run and where each run ends - many of the runs in this case are just 1 request long, except for runs 8, 11 and 12. We denote the first request of a run as the *run start request*, the remaining requests of the run as *within a run requests*, and define the following attributes to characterize a run:

- *Run length*: The number of I/O requests in a run before it is broken by a non-sequential request.
- *Run Start Location*: This is the location (sector) accessed by the first request in a run.

ReqID	Location	Size	RunID	StreamID	Stream Jump Dist.		Interf.	Active
					inter	intra	Dist.	Streams
#1	102893	64	1	1				1
#2	123757	512	2	2	20800			1
#3	102957	64	3	1		0	1	1
#4	68653	64	4	3	-34368			2
#5	68781	64	5	3		64	0	2
#6	109933	64	6	4	41088			3
#7	68845	64	7	3		0	1	3
#8	103021	64	8	1		0	4	3
#9	103085	192	8	1		0	0	3
#10	109997	64	9	4		0	3	2
#11	68909	64	10	3		0	3	2
#12	110061	64	11	4		0	1	1
#13	110125	64	11	4		0	0	1
#14	110189	64	11	4		0	0	1
#15	108909	512	12	5	-1344			1
#16	109421	512	12	5		0	0	1

Table 2. A window of requests from Q10

While one may just proceed with the notion of a run, we want to point out that there are correlations between the run start locations, i.e. when a thread is interrupted/pre-empted from its sequential run by another, it is likely to get back to where it left off (or at least close to it) when it is scheduled again. If we ignore this artifact (i.e. assume run start locations are independent), then we may not be very accurate

in our synthesis. Consequently we introduce the notion of a stream wherein we try to check for sequentiality not just with the immediately previous requests, but go back further in the past. At the same time, we do not want to be too rigid on the sequentiality issue, i.e. we may want requests that are off by a few sectors to still belong to the same stream. Such a relaxation can (i) allow better stream formation/characterization (we do not want to end up creating as many streams as runs), and (ii) allow some vagaries in workload execution (e.g. an activity may be skipping alternate sectors but these still need to be in the same stream since they are correlated). More formally, when examining the trace we classify request req_i that reads s_i sectors starting from sector l_i to belong to the same stream as some other request req_j (reading s_j sectors starting from sector l_j) that has already been encountered in the trace (i.e. $j < i$), as long as

$$l_j - d_{backward} \leq l_i \leq l_j + s_j + d_{forward}$$

At the same time, we want to restrict how much backwards (j) in the trace that we need to go, in order to keep the problem tractable and meaningful. After extensive studies, we found that using a history of 32 requests (i.e. $i - j \leq 32$), and setting $d_{forward} = 256$ sectors, and $d_{backward} = 512$ sectors, served our purpose quite well across the queries.

When we examine the example trace in Table 2, we find that request #3, starts accessing sector 102957 which is immediately after the last sector read by request #1, putting them in the same stream (note that they were not part of the same run). Similarly requests #8 and #9 became part of the same stream (denoted as StreamID 1). In essence, requests that are within a run, become part of the same stream, though a stream can capture requests that are not strictly sequential either in terms of their adjacency in the trace (they can be upto 32 positions apart) or in terms of the sectors that they access (they can be apart by $d_{backward}$ and $d_{forward}$).

As with the run, we denote the first request of a new stream as the *Stream Start Request*. We try to capture the following attributes for a stream:

- *Stream Length*: This is the number of requests in a stream. For example, if we look at StreamID 1 in Table 2, its length is 4 (requests #1, #3, #8, and #9). Note that this should be at least as large as the Run Length defined earlier, since every run is part of a stream.
- *Inter-stream Jump Distance:* This is the spatial separation in sectors between the sector read by the current stream start request and where the previous request (has to be of another stream or else this will not be the stream start request!) left off. For instance, in Table 2 request #6 is the start of stream 4 that reads from sector 109933, while its immediate predecessor (request #5) reads until sector 68845 (68781+64) making the inter-stream jump distance 109933 - 68845 = 41088, for stream 4. Intuitively, if we have two interleaving streams, this jump distance captures how much the disk head will have to move after serving a run of one stream before moving to another stream. Note that we need to calculate this only at the start of a particular stream (requests #2, #4, #6, #15). From the synthetic workload viewpoint, if we could use this information to generate the first request of a stream, then we could use the intra-stream characteristics (described next) to generate the requests within a stream.
- *Intra-stream Jump Distance:* This attribute captures the relation between successive references within a stream. Formally, for a request i this is calculated as $l_i - (l_j + s_j)$ where l_i and s_i are the start sector and number of sectors accessed for a request i in a stream and l_j and s_j are the start sector and number of sectors accessed for the immediately previous request j in the same stream. In Table 2, only request #5 (the 2nd request of stream 3) has a non-zero intra-stream jump distance. As is apparent, this attribute captures the sequentiality in the requests within a stream.
- *Number of Active Streams:* It is possible that over the duration of a query there may be numerous streams. However, it is more important to understand how many streams are active (i.e. their start requests have occurred but their last request has not yet arrived) at any given time. We call this the number of active streams at an instant, and this is given in the column denoted by Active Streams in Table 2 assuming that these 16 requests constitute the entire trace.
- *Interference Distance:* The previous stream attributes can help us generate the number of streams, the length of a stream and the requests within a stream. It is also important to understand how these streams interleave, and we capture that using the interference distance. This is formally defined as the number of requests between two successive requests j and i in a stream and is given by $i - j - 1$. The interference distance column in Table 2 gives these values for the example sequence. For instance, the two successive requests (#1 and #3) of stream 1 are separated by a request from another stream. Note that this parameter will be 0 for within a run requests.

The above run and stream attributes will help us better synthesize the spatial reference behavior of the queries. In fact, they are general enough to encompass a diverse range of workload behavior. For instance, we can get an intuitive understanding of their ability to capture the three categories of queries shown earlier in Figure 1 by examining the number of active streams in these queries. For the queries in Category 1, where there is very good sequentiality and little interleaving, the average number of active streams is very close to 1, with the stream and run lengths being relatively large. At the other extreme, with the queries in Category 3, streams are extremely short/rare, putting the average number of active streams close to 0. For the queries in category 2, we find the number of active streams is higher (between 1.5 and 3 in the trace data), with stream/run lengths in between these two extremes. These results also re-confirm our earlier categorization of the queries, and give evidence of possible utility of these attributes in the synthetic workload generation.

Note that as far as characterizing these secondary attributes from the trace is concerned, one could simply use a mean value of the observations from the trace, or we can resort to empirical distributions (or even curve fitting for the empirical distributions) to preserve more information about the original trace. There are trade-offs between accuracy and storage overheads (which we want to keep low since this is one reason why a synthetic workload may be preferable in the first place), that we always keep in mind. In our synthesis we ensure that the storage overhead is fairly small (we also quantify this towards the end of the paper), while still being fairly accurate in the synthesis.

Inter-attribute Correlations: We found that it is not only important to understand and exploit the correlations of values of an attribute across the requests (as we did in the run/stream concepts for location), but also the correlations across the attributes themselves. For instance, the starting sector (location) of a request can be highly correlated with the size+location of where the previous request (of that run) left off. Further, accesses with smaller sizes tend to come temporally closer to each other than those for larger sizes

(because it may take a lot more time to process that data before the next I/O requests).

We next go over the details for applying these methods and synthesize the queries in each category. We begin with category 1 which poses more challenges (this is not as regular as category 2, and not as random as category 3) to explain the overall process, which then become largely applicable to the other two categories.

5.2. Category 1: Q4, Q10, Q14

We will use Q10 as a representative example (in the interest of space) of Category 1 to walk over the intermediate steps of the analysis, though the final results of the synthesis are given for all queries.

Synthesizing Location: As in section 4 where we focussed only on the arrival time and used the other parameters from the trace, we start this exercise as well by attempting to synthesize one primary access parameter - location (starting sector) - and take the values for the other primary parameters directly from the trace.

We begin with a very simple model, called *LocIID*, where the starting sectors are assumed to be independent of each other and can be drawn from the same probability density function (which worked in the case of arrival times). Instead of trying to fit a curve (specific distribution) for this density function, we instead use an Empirical CDF, i.e. the CDF is obtained by simply histogramming the number of occurrences of each starting sector in the trace. Note that this ECDF may incur a high storage cost with the number of sectors becoming quite large. Our point here is to merely note that even a very accurate function for the probability density will not produce good results because the locations are not "independent". Figure 2 clearly clarifies this point, where the response time characteristics using the LocIID model is quite significantly different (nRMS=1.41) from that for the original trace on the simulator.

Our next improvement, called LocRUN, tries to capture correlations of locations between nearby requests using the concept of a run described in the previous section. In LocRUN, we calculate the run length distribution and the run start location distribution (this is obtained as an ECDF) for the original trace. As in LocIID, the ECDF for the run start location can again get exceedingly large, and we will show that despite such an accurate density function, just a run-based approach does not suffice in this case. To generate the trace, we randomly pick a run start location from the ECDF, and we randomly pick a run length using its distribution. Since the request sizes (and arrival times) are coming from the original trace, we can use these sizes to generate the start sector for the next request within a run (has to be spatially contiguous from the previous one). We illustrate the response time characteristics for LocRUN as well in Figure 2, wherein it shows better behavior than LocIID, but is still quite unacceptable (nRMS = 0.83).

Even if we use ECDF of run start sectors in LocRUN, we are assuming its independence when we start each run, which may not really be the case. This is where the concept of a stream comes in useful, i.e. if a stream got interrupted in its run by another stream, then it will probably start its next run close to where it left off previously. A brief summary of the steps in our next stream-based location generation model, **LocSTREAM**, is given below:

1. We use the stream length distribution obtained from histogramming the original trace, and we generate a length probabilistically from this distribution. A pictorial depiction of the stream length distribution for Q10 is given in Figure 3 (a).

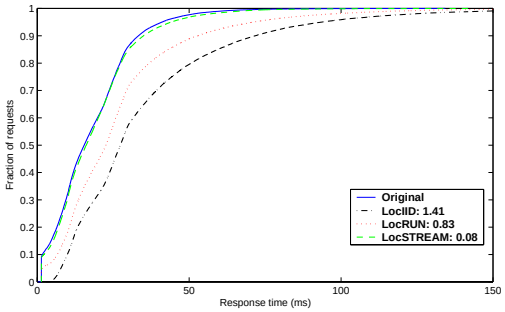


Figure 2. Location model validation for Q10.

2. We obtain a conditional distribution of the run length for each stream length (shown in Figure 3 (b)). From this distribution we generate run lengths for the stream length picked in step 1 - note that we need to pick several run lengths till the sum total of these lengths gets to the stream length.

3. Having identified the number of runs and their lengths for each stream, we need to next generate the start location for each of these runs. This in turn takes 2 steps: (a) We use the inter-stream jump distance distribution (given in Figure 3 (c)) to find the start location of the first run for that stream, and (b) We use the intra-stream jump distance distribution (given in Figure 3 (d)) to find the start location for each run in this stream. Note that since we are getting the size of each request from the original trace, we can use this and the strict sequentiality property within each run to determine the location parameter for each successive request in a run once the run start location is determined.

4. The final step is to take the different runs/requests generated until now and intermix them to get a single trace. We use the interference distance distribution (given in Figure 3 (e)) for this purpose, i.e. if we pick a number x using this distribution, we need to place x other requests between successive runs of this stream. It is possible when making these assignments to get a linear ordering of requests, we can have conflicts (i.e. two requests may want to occupy the same slot in the synthetic trace). In such cases, we simply use the next (closest) empty slot for one of them.

In Figure 2, when we observe the response time characteristics of a synthetically generated trace where we use LocSTREAM for starting sector determination, and use the inter-arrival times and sizes from the original trace, we are very close to the performance of the original trace (nRMS=0.08), giving us good confidence in the stream+run based modeling strategy for spatial access pattern.

Synthesizing the Complete Trace: While one could use a similar investigation to synthesize and validate request sizes (in sectors), which is the final access pattern attribute, we do not explicitly go in that direction in the interest of space. Further, we find that even if we are synthesizing one parameter independently, it is more important to understand inter-parameter correlations in the final trace synthesis and we directly proceed to that issue. There are not too many distinct request sizes, which is typically an artifact of the database engine, and Figure 4 (c) shows the probability of occurrence of 8 different sizes, in sectors, for Q10. We consequently use this empirical CDF of request sizes, which is reasonably small.

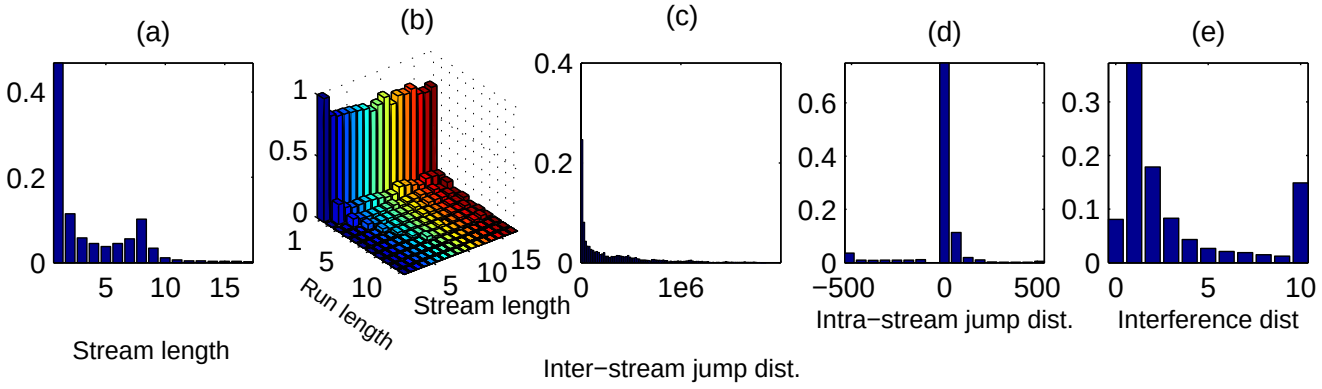


Figure 3. Characteristics of secondary attributes used in LocSTREAM (given here for Q10).

As a simple approach for trace generation, we could use the results from each of the previous exercises to generate the inter-arrival times, request start locations and request sizes independently, and put them together to compare the response time characteristics with that for the original trace. We have tried this but found its accuracy to be poor (nRMS = 0.30) - its response time CDF is not explicitly given due to space constraints.

The reason for this disparity, even if the individual parameters have been rather accurately modeled, is that we have not considered the correlations between the parameters:

- Typically one can expect correlations between inter-arrival times and start sector, i.e. if the gaps are shorter, then the start locations may also be closer. For instance, inter-arrival times of requests within a run (that touch successive locations) can be expected to be shorter than the time gap between run start requests and their predecessors (which are more spatially separated). Evidence of this expectation can be found in Figure 4 (a) where we show the inter-arrival time CDFs for the run start requests and the requests within a run separately (which can be fit as IID exponential distributions with means 22.497 and 12.472 milliseconds respectively). We refer to Figure 4 (a) as the arrival time distribution conditional on run start and within a run requests.
- We can expect correlations between the location and size attributes as well, e.g. if we are within a run, the next location is definitely a function of the size of previous request. The last two rows of the table in Figure 4 (c) shows this important correlation, wherein we find that the requests within a run are more biased towards smaller sizes than the starting request of a run. We refer to these two rows as the request size distribution conditional on run start and within a run requests.
- Finally, we can also expect correlations for the final pair: time and size. In the case of requests within a run we already observed that their inter-arrival times were much lower, and the size was more heavily biased towards the smaller (64 sector) sizes. When we examine the sizes used by the run start requests and their inter-arrival times in Figure 4 (b), we find that requests with smaller inter-arrival times are more biased towards the smaller (64 sector) size, compared to the larger inter-arrival times. Note that the x-axis in Figure 4 (b) represents 30 buckets of inter-arrival time ranges (and not absolute inter-arrival times). We refer to Figure 4 (b) as the size distribution conditional on inter-arrival time.

These correlations are ignored when we assume independence between the parameters, causing significant inaccuracies, and we consequently employ the following steps to address these correlations.

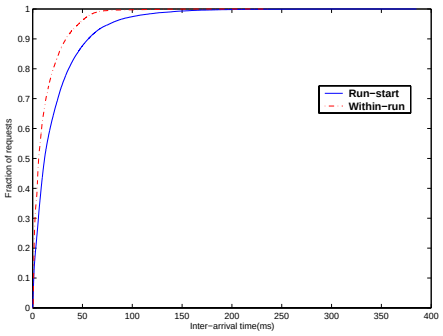
Synthesis Methodology for Cat. 1

1. LOCATION: We first use LocSTREAM to generate the start location for each request, which will also classify a request to be a starting request of a run, or a request within a run. Note that we also allow runs to be of length 1, in which case there is only a starting request, and no request within that run.
2. TIME: We use the inter-arrival time distribution (Figure 4 (a)) conditioned by run start requests and within a run requests to independently generate arrival times for the two sets of requests obtained from step 1.
3. SIZE: At this step, we have the location and times synthesized for the two sets of requests: run start requests and within a run requests. We next generate the requests sizes for each of these two sets as follows:
   - For the run start requests, we use its inter-arrival time (synthesized from step 2 above) to generate a size from the conditioned size distribution for the run start requests (Figure 4 (b)).
   - For within a run requests, we simply use the density function of the size for such requests (row 3 of Figure 4 (c)) to generate their sizes.

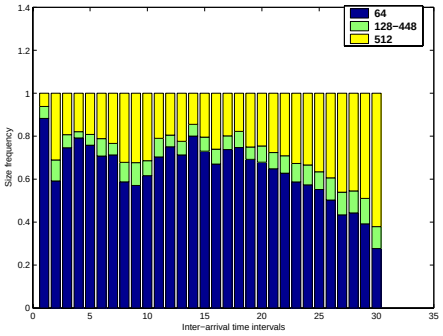
The synthetic trace generated by this methodology has been directed to the simulator and its response time characteristics have been compared with those for the original trace in Figure 5. We find that the response time CDFs of the synthetic and original trace for Q10 are quite close, with nRMS = 0.16. We give the final accuracy result for Q10's synthesized trace in Figure 5. Q4 and Q14's results can be found in [27]. Our results show that this methodology works in those cases as well.

5.3. Category 2: Q1, Q3, Q5, Q6, Q7, Q8, Q12, Q15, Q18, Q19, Q21

The basic characteristics differentiating this category from the earlier one is that the number of active streams



(a)



(b)

Size(Sectors)	64	128	192	256	320	384	448	512
All req.	0.716	0.009	0.010	0.009	0.009	0.011	0.011	0.225
Run start	0.577	0.012	0.013	0.012	0.013	0.015	0.016	0.342
Within run	0.916	0.004	0.004	0.004	0.004	0.005	0.005	0.057

(c)

Figure 4. Correlations between attributes (for Q10) used in the synthesis methodology. (a) Arrival time distribution conditional on run-start/within-run requests, (b) Size distribution conditional on inter-arrival time for run-start requests, (c) Size distribution conditional on run-start/within-run requests.

is typically 1 (there are no interleaving streams), with very good sequentiality and the request sizes are typically much larger. For instance, the 512 sector request size constitutes 99% of the requests in Q19, 90% of the requests in Q12, 85% of the requests in Q21, and 79% of the requests in Q3.

We can use our trace generation methodology discussed in the previous section here as well. The only difference that arises is that in some of these queries, with a lot of sequentiality, runs can get very long making the number of run start requests quite low. In fact, in Q19 and Q21, the number of run start requests is less than 3%, making it difficult to undertake the steps that involve separately dealing with run start requests and within a run requests. In such cases, we can simplify the methodology even further, i.e. step 1 of the methodology remains as is, step 2 does not condition the arrival time by run start/within a run (instead we have just one arrival time distribution), and step 3 does not specifically deal with run start requests.

Figure 5 compares the response time characteristics of the trace synthesized by applying our methodology to Q12

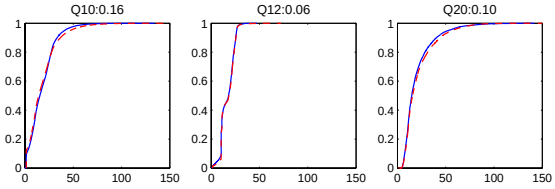


Figure 5. Accuracy of the final synthetic traces shown for a representative query from each category (Q10 for Category 1, Q12 for Category 2, and Q20 for Category 3). The plots for other queries can be found in [27]. x-axis is the response time, and y-axis is the CDF. The solid line is the response time CDF for the original trace, and the dotted line is for the synthetic workload. nRMS values are given at the top of each plot.

with that for the original trace. The results for the other 10 queries can be found in [27]. We find that our generic methodolog, adapts automatically for this category (which is a specialization of category 1 in that there is 1 active stream and a lot more sequentiality), giving nRMS values that are even less than 0.10.

5.4. Category 3: Q9, Q17, Q20

When we move to category 3 queries, we find that these queries are randomly (uniformly) distributed in terms of their starting locations, and there is very little run/stream behavior. The request sizes are also dominated by the smaller 64 sector value (for example, all requests in Q20 are for 64 sectors), making it redundant to study any correlations between size and other parameters. These characteristics help us simplify/adapt our described trace generation methodology in the following ways. Step 1 is as before, except now the run/stream lengths are invariably of size 1, and the start sectors are randomly (uniformly) distributed. In terms of Step 3, we can simply use the ECDF of sizes (which is dominated by 64 sector accesses) to generate a size without worrying about runs and other correlations. In terms of step 2, while we can assume independence between inter-arrival times themselves, we need to however consider the correlation between these times and the start location - with these sectors randomly distributed, seek overheads are expected to be higher, and this will consequently affect the time of injection of the next request. We exploit the correlations between jump distance (note that in this case every request is a an individual stream by itself, and the inter-stream jump distance defined earlier automatically captures this) and the inter-arrival times to generate the arrival characteristics for this category (which is similar to how we exploited the correlations between arrival times and runs in category 1).

The accuracy result for Q20 is given in Figure 5. As can be seen, our synthesis provides very good results, and so is the accuracy for Q9 and Q17 whose results can be found in [27].

6. Concluding Remarks

Synthesizing representative I/O workloads for designing and optimizing disk subsystems is an important and chal-

lenging area of research. This paper has focussed on one important decision-support commercial workload, TPC-H, towards completely characterizing and synthesizing its disk block-level requests.

The main contributions of this work are two-fold: (i) a synthesis methodology that captures correlations between the primary attributes of the requests, and consequently between the secondary attributes we track as well, and (ii) application of this methodology to the TPC-H queries to characterize its behavior and synthesize the I/O request pattern of its queries. An application of this methodology to I/O requests of TPC-H suggest that hyper-exponential distributions can capture inter-arrival times At the same time, it is possible to capture the regularity/non-regularity in the sequential behavior of requests from several inter-mingling streams, by incorporating correlations between/across the attributes of requests.

We have also examined the sensitivity of our methodology in terms of the disk subsystem target that is used for the validation, as well as the sensitivity to the system where the trace is collected and the size of the dataset. Our methodology is fairly resilient to these factors, giving good accuracy across the spectrum (please refer to [27] for detailed results).

	Q1	Q3	Q4	Q5	Q6	Q7
Storage Fraction ($\times 10^{-3}$)	3.46	3.64	2.76	3.43	3.46	3.47
nRMS	0.10	0.09	0.20	0.07	0.01	0.04
	Q8	Q9	Q10	Q12	Q14	Q15
Storage Fraction ($\times 10^{-3}$)	3.66	0.004	2.79	3.73	6.49	3.46
nRMS	0.05	0.15	0.16	0.06	0.19	0.01
	Q17	Q18	Q19	Q20	Q21	
Storage Fraction ($\times 10^{-3}$)	2.03	3.54	3.44	4.57	2.95	
nRMS	0.05	0.06	0.03	0.10	0.07	

Table 3. Storage costs for synthetic generator as a fraction of the space taken by the original trace for each query. Note that the fractions need to be multipled by 10^{-3}

It is important to strike a good balance between the storage overheads needed for the synthetic generator (relative to the original trace) and the resulting accuracy. Table 3 quantifies the size required to store the characteristics of the primary/secondary attributes and their correlations for our synthetic workload in each query, relative to the storage for the original trace. These numbers clearly illustrate the storage benefits of our generator while providing response time characteristics that closely mimic the original trace. This is in addition to the numerous benefits of a synthetic workload generator explained earlier.

We would like to point out that we have been able to present only a portion of the interesting results. At the same time, we have also shown some of the negative results along the way in order to better motivate our intermediate steps. Our ongoing work is examining the use of the developed techniques in not only evaluating them for other workloads (both commercial and non-commercial) but also in on-line workload characterization where we want to predict requests as the workload evolves for autonomic performance tuning.

Acknowledgements: This research has been supported in part by NSF grants 9988164, 0097998, 0325056, 0130143, an IBM Faculty award, and an IBM SUR Equipment grant.

References

[1] S. Asmussen, O. Nerman, and M. Olsson. Fitting phase-type distribution via the EM algorithm. *Scandinavian Journal of Statistics*, 23:419–441, 1996.

[2] G. Box and G. Jenkins. *Time Series Analysis Forecasting and Control*. Holden-Day, 2nd edition, 1976.

[3] D. DeSota. Characterization of I/O for TPC-C and TPC-H Workloads. In *Proceedings of the Workshop on Computer Architecture Evaluation Using Commercial Workloads*, January 2001.

[4] P. Dinda and D. O'Hallaron. An Extensible Toolkit for Resource Prediction In Distributed Systems. Technical Report CMU-CS-99-138, School of Computer Science, Carnegie Mellon University, July 1999.

[5] A. Downey and D. Feitelson. The elusive goal of workload characterization. *Performance Evaluation Review*, 26(4):14–29, 1999.

[6] G. Ganger. Generating Representative Synthetic Workloads: An Unsolved Problem. In *Proceedings of the Computer Measurement Group (CMG) Conference*, pages 1263–1269, December 1995.

[7] G. Ganger, B. Worthington, and Y. Patt. *The DiskSim Simulation Environment Version 2.0 Reference Manual*. http://www.ece.cmu.edu/ ganger/disksim/.

[8] M. E. Gomez and V. Santonja. Analysis of Self-Similarity in I/O Workload Using Structural Modeling . In *Proceedings of International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems(MASCOTS)*, March 1999.

[9] M. E. Gomez and V. Santonja. A New Approach in The Modeling and Generation of Synthetic Disk Workload. In *Proceedings of International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2000.

[10] W. W. Hsu, A. J. Smith, and H. J. Young. Analysis of the Characteristics of Production Database Workloads and Comparison with the TPC Benchmarks. *IBM Systems Journal*, 40(3), 2001.

[11] Y. Hu, T. Nightingale, and Q. Yang. RAPID-Cache — A Reliable and Inexpensive Write Cache for High Performance Storage Systems. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):290–307, March 2002.

[12] IBM DB2. http://www-3.ibm.com/software/-data/db2/.

[13] IBM TPC-H Disclosure Report. http://www.tpc.org/-results/FDR/tpch/x350_100GB_16proc_FDR.pdf.

[14] K. Keeton, G. Alvarez, E. Riedel, and M. Uysal. Characterizing I/O-intensive Workload Sequentiality on Modern Disk Arrays. In *Proc. Workshop on Computer Architecture Evaluation Using Commercial Workloads*, 2001.

[15] K. Keeton, A. Veitch, D. Obal, and J. Wilkes. I/O Characterization of Commercial Workloads. In *Proceedings of the Workshop on Computer Architecture Evaluation Using Commercial Workloads*, January 2000.

[16] D. Kotz and C. S. Ellis. Practical prefetching techniques for parallel file systems. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems*, pages 182–189, December 1991.

[17] Z. Kurmas, K. Keeton, and R. Becker-Szendy. Iterative Development of an I/O Workload Characterization. In *Proceedings of the Workshop on Computer Architecture Evaluation Using Commercial Workloads*, January 2001.

[18] Z. Kurmas, K. Keeton, and K. Mackenzie. Synthesizing Representative I/O Workloads Using Iterative Distillation. http://www.cc.gatech.edu/ kurmasz/.

[19] Linux Scalability Effort: File List. http://lse.-sourceforge.net/project/showfiles.php?group_id=8875.

[20] T. M. Madhyastha and D. A. Reed. Input/output access pattern classification using hidden Markov models. In *Proceedings of the Workshop on Input/Output in Parallel and Distributed Systems*, pages 57–67, November 1997.

[21] J. Oly and D. Reed. Markov model prediction of I/O request for scientific application. In *Proceedings of the 2002 International Conference on Supercomputing*, June 2002.

[22] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27(3):17–28, 1994.

[23] J. Schindler, A. Ailamaki, and G. R. Ganger. Lachesis: Robust Database Storage Management Based on Device-specific Performance Characteristics. In *Proceedings of the 29th International conference on Very Large Data Bases*, 2003.

[24] E. Smirni and D. A. Reed. Workload characterization of input/output intensive parallel applications. In *Proceedings of the Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, volume 1245, pages 169–180. Springer-Verlag, June 1997.

[25] TPC-H Benchmark. http://www.tpc.org/tpch/.

[26] N. Tran. *Automatic ARIMA Time Series Modeling and Forecasting for Adaptive Input/Output Prefetching*. PhD thesis, University of Illinois at Urbana-Champaign, 2002.

[27] J. Zhang, A. Sivasubramaniam, H. Franke, N. Gautam, Y. Zhang, and S. Nagar. Synthesizing Representative I/O Workloads for TPC-H. Technical Report PSU-CSE-03-018, Department of Computer Science and Engineering, Pennsylvania State University, October 2003.

[28] Y. Zhang, J. Zhang, A. Sivasubramaniam, C. Liu, and H. Franke. Decision-Support Workload Characteristics on a Clustered Database Server from the OS Perspective. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS)*, May 2003.