Time and Cost-Efficient Modeling and Generation of Large-Scale TPCC/TPCE/TPCH Workloads

Christina Delimitrou[1], Sriram Sankar[2], Badriddine Khessib[2],
Kushagra Vaid[2], Christos Kozyrakis[1]

[1] Electrical Engineering Department, Stanford University, Stanford, CA, USA, 94305
{*cdel, kozyraki*}@stanford.edu
[2] Global Foundation Services, Microsoft, Redmond, WA, USA, 98052.
{*srsankar, bkhessib, kvaid*}@microsoft.com

Abstract. Large-scale TPC workloads are critical for the evaluation of datacenter-scale storage systems. However, these workloads have not been previously characterized, in-depth, and modeled in a DC environment. In this work, we categorize the TPC workloads into storage threads that have unique features and characterize the storage activity of TPCC, TPCE and TPCH based on I/O traces from real server installations. We also propose a framework for modeling and generation of large-scale TPC workloads, which allows us to conduct a wide spectrum of storage experiments without requiring knowledge on the structure of the application or the overhead of fully deploying it in different storage configurations. Using our framework, we eliminate the time for TPC setup and reduce the time for experiments by two orders of magnitude, due to the compression in storage activity enforced by the model. We demonstrate the accuracy of the model and the applicability of our method to significant datacenter storage challenges, including identification of early disk errors, and SSD caching.

Keywords: Workload, Modeling, Storage Traces, TPC benchmarks, Characterization, Storage Configuration, Datacenter.

1 Introduction

As cloud data-stores have emerged over the past decade, user data has started being increasingly stored in large-capacity and high-performance systems, which account for a significant portion of the total cost of ownership (TCO) of a datacenter (DC) [4]. Specifically for large-scale databases, data retrieval is often the bottleneck for application performance [1, 4], promoting efficient storage provisioning to a first-order design constraint. This makes the study of the TPC benchmarks (TPCC, TPCE and TPCH), which have traditionally been of fundamental importance for the configuration and evaluation of DC-scale systems, even more important. TPCC and TPCE especially, being clearly I/O-bound, are tailored for use in storage system studies, while TPCH also experiences significant challenges towards its optimal configuration. However, one of the main challenges when trying to evaluate storage system options using large instances of TPC is the *cost* in *time,* and *demand* for *expertise* for setting up and managing the workload itself. Applications like TPCC, TPCE and TPCH introduce high complexity in order to be correctly configured and setup to scale to hundreds of thousands of servers, while they demand in-depth knowledge of the structure and functionality of the workloads from the DC operators.

One of the main challenges when using large-scale applications to evaluate storage is the difficulty in replaying the entire application in all possible system configurations. The effort itself can be highly inefficient in both time and cost. Furthermore, applications like TPCC and TPCE differ from conventional desktop applications in that they cannot be approximated by single-machine benchmarking, therefore highly scalable experiments are required.

Despite the merit in understanding the characteristics of the workload in order to effectively provision its storage system, no in-depth, per-thread characterization of the I/O behavior of TPC applications exists. Previous work on workload generation [6, 10, 11, 13] lacks the ability to capture aspects of the workload like spatial and temporal locality which are critical for the accurate representation of the application's storage activity, while experiments are limited to small scales. This underlines the importance of investing in frameworks that enable extensive workload analysis, characterization and modeling, while permitting easy and fast setup for TPC applications, without requiring significant knowledge on the application's intricate details and functionality, therefore can be performed directly by storage experts.

In this work, we propose a framework that enables *fast* and *accurate* configuration and storage experiments for TPCC/E/H using a workload model and a tool that generates storage activity that is similar in I/O characteristics and performance metrics to that of the original application. This framework decouples performing large-scale storage experiments from the requirement to being an expert in the TPC workloads. This infrastructure includes probabilistic, state diagram-based models [1] that capture information of configurable granularity on the workload's access patterns. The models are developed from production traces of large instances of TPCC, TPCE and TPCH. We identify the optimal level of detail required for the model to accurately describe the storage activity of each application and design a tool that recognizes them and recreates access patterns with I/O features that resemble those of the original workloads. We have performed extensive validation on the accuracy of the infrastructure, in terms of the generated storage behavior and have verified the consistency and conciseness of the results [3]. Based on these models and the original traces we perform an in-depth, per-thread characterization of the storage activity of the TPC benchmarks and provide insights on their behavior.

The proposed framework can be used for research on large-scale storage systems. Specifically in this work we present two possible use cases, namely: identification of early errors in large-scale storage systems, and evaluation of incorporating SSD caching in the back-end servers to improve performance. The results demonstrate the accuracy of the modeling process, and the time and cost-efficiency in performing large-scale experiments for TPC benchmarks.

Succinctly, the main contributions of this work are:

- We perform an extensive, per-thread characterization of the access pattern and characteristics in the storage activity of the TPCC, TPCE and TPCH workloads and provide insights on what this means in terms of query correlation.
- We provide a framework for accurate modeling, of configurable detail, of the storage activity for TPCC, TPCE and TPCH and verify the resemblance between original and synthetic application in I/O features and performance metrics.

- We greatly simplify the setup and configuration procedure for evaluating storage using large-scale TPC benchmarks, and remove the requirement for expertise in the application’s structure, components and functionality, therefore enabling storage experts to independently perform storage configuration experiments.
- We greatly reduce the time required to perform large-scale storage experiments (e.g. SSD caching, from days in the original TPC setup, to minutes, when using the model i.e. a 150x time reduction), by compressing the storage behavior and removing the need to deploy the entire application in different storage configurations.
- We demonstrate the scalability of our methodology to a large number of servers, as well as the ability to perform accurate scaled-down experiments using fewer instances of the model.
- We show the applicability of this methodology to a wide spectrum of use cases, ranging from identifying early problems with storage (infant mortality), to storage configuration optimizations (e.g. use of SSDs, hybrid HDD/SSD systems).

This methodology enables studies previously impossible without a full TPC setup and without application deployment for every modification in the storage system. Thus it greatly reduces the overhead and complexity of performing large-scale studies, while offering the ability to setup the workloads to storage experts as well.

The rest of this paper is structured as follows. Section 2 discusses the motivation for this work. Section 3 presents related work on TPC characterization and storage modeling and generation. Section 4 provides a description of the model and an overview of the tool's implementation as well as an in-depth, per-thread characterization of the three applications. Section 5 discusses the validation of the methodology against the original TPC workloads, and a comparison of our toolset with a popularly used workload generator. Section 6 discusses the applicability of the tool in evaluating the important DC storage challenge of storage endurance and SSD caching. Finally, Section 7 presents topics for future work and concludes the paper.

2 Motivation

Datacenter applications are hard to model due to their varying user demand and large scale. Large enterprises typically use industry-standard TPC benchmarks for configuring their database systems prior to actual deployment. However, TPC benchmarks are extremely difficult to setup, sometimes taking multiple weeks to get a working configuration. In this work, we strive to reduce this setup time and the current need to have TPC as a full application deployment. In earlier large scale TPC setups, there used to be disks with intermittent read failures or bad sectors that would create latency profiles with few outliers that were very difficult to identify. In present day setups, SSDs (Solid State Drives) encounter intermittent write performance issues that are difficult to detect in a full application deployment. For all these cases, we need improved storage testing ability. The objective of our modeling and I/O regeneration work is to provide such a facility.

In addition to reducing setup times and facilitating fast storage testing, we also use our framework to test new datacenter technologies, including tiered storage approaches. One recent development is the use of SSDs in traditional hard disk drive

(HDD) space. Since SSDs are significantly costlier per GB than HDDs, we need to configure the use of these devices such that the eventual cost-performance of the system is optimal. Such studies also require running the entire application setup, which takes a lot of time to test multiple configurations. Our framework is designed to address multiple test configurations in an efficient manner using intensity knobs.

3 Related Work

Performing scalable experiments using the TPC workloads is of great interest to hardware architects, especially when the target system is a large-scale DC. Specifically, because of their I/O-dominated behavior, TPCC and TPCE are of primary importance when configuring and evaluating the storage system of large DCs.

Significant prior work [7] has studied how to efficiently provision this part of the system, however, using the TPC workloads in this scope, introduces a large overhead, in terms of setup and maintenance for the application. An ideal way to overcome this is by using a model that captures the storage behavior of TPC benchmarks and a tool that recreates representative access patterns that resemble the original workload. Such a framework would provide insight on the storage behavior, and greatly reduce the time required for the setup and configuration of large-scale databases, as well as the time for storage configuration experiments.

Despite the obvious merit in developing such an infrastructure, most prior work on large-scale storage configuration is empirical, primarily relying on extracting the workload characteristics based on traces [8].

Kavalanekar et al. in [2] and [8] use a trace-based approach to characterize large online services for storage configuration and performance modeling respectively. Traces offer useful insight on the characteristics of large-scale workloads, but their usefulness is limited by the system upon which they have been collected. Generating TPC workloads with high fidelity can offer far richer information towards understanding their behavior and making experimenting with them easier and faster. It also enables addressing instrumental challenges in storage system design (error detection, SSD caching, data migration) when optimizing for performance, efficiency and/or cost.

Prior work on workload generators includes SQLIO [10], Vdbench [11], and IOMeter [6] which remains the most well-known open-source workload generator. The main disadvantage of these tools is that, as in the case of IOMeter, they lack the ability to represent the spatial and temporal locality in the I/O accesses, which for applications like TPC is critical in distinguishing between hot and cold tables in the database. This makes these tools impractical, if not impossible to be used for locality-aware studies, which are critical for many storage optimizations.

In this work we present a new framework for TPC workloads, which allows accurate modeling and generation of their I/O access patterns and greatly simplifies the setup of the infrastructure and the execution of large-scale experiments.

4 Modeling and Generating TPCC/E/H

Our framework consists of two main components. First, we train the model that captures the storage activity of the TPC workloads. Second, we have implemented a tool that recognizes these models and creates storage access patterns with similar characteristics as in the original applications.

4.1 Storage Workload Model

Our model is based on the Markov chain representation discussed in [1]. According to the model, states correspond to ranges of Logical Block Numbers (LBNs) on disk, and transitions, represent the probability of consecutive I/Os moving between states. The transitions are characterized by the following I/O features: *block size*, *type of request* (read, write), *randomness*, and *inter-arrival time* between subsequent requests. We present here the basic features of the state diagram. More details on the model can be found in [3]. The main insight behind the structure of the model is that spatial locality of I/Os can be represented by the clustering of requests in each state, and temporal locality by the transitions between states. The probability of each transition is calculated as the percentage of the requests that correspond to it. Figure 1(a) shows a simple state diagram with four states.

In order to include more detailed information on the I/O access pattern we have extended this model to a hierarchical representation with each state expanding to become a new state diagram. This way, with a simple sensitivity study we can identify the optimal amount of information required by each application to make the model accurate. Figure 1(b) shows an example of such a model using two levels. Increasing the number of states per level to reduce the number of levels is possible but augments the complexity of the model, as the transition count increases substantially.

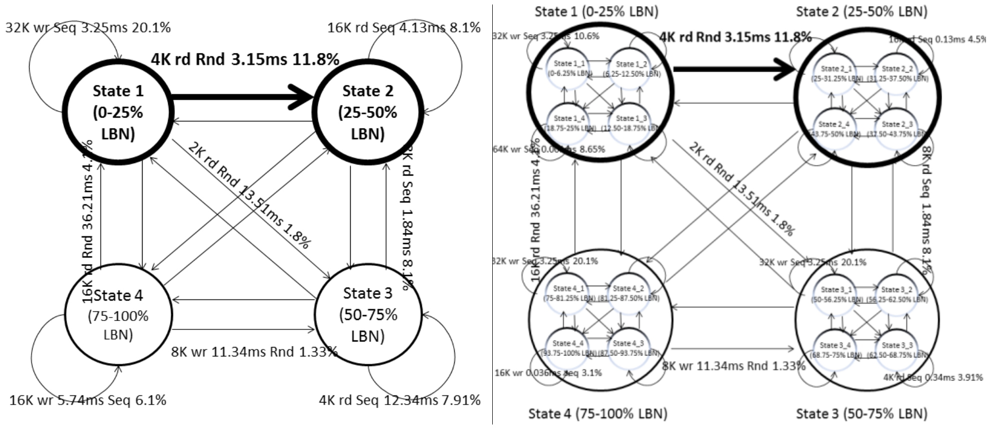


Figure 1: State Diagram Model: (a) One level, (b) Two levels

4.2 Generation Tool

The second part of the infrastructure consists of a tool that recognizes the storage workload model and creates a benchmark with activity similar to that of the original TPC application. For this purpose we have used and extended DiskSpd [5], a workload generator that works as a command line tool, initiating read/write requests in burst mode to disks and/or files. In order to account for the information captured in the model, we have implemented a set of additional features in DiskSpd, which briefly are:

- Initiating I/O requests with specified inter-arrival times, either static or time-varying to capture fluctuation in the intensity of storage activity.
- Maintaining transition probabilities between threads and guaranteeing that the generated workload is a compressed and accurate version of the original one.

- Varying the intensity of the generated workload using a knob that scales the inter-arrival times between subsequent I/O requests. This enables evaluation of alternative storage system configurations, e.g. SSD-based systems.

In Section 5 we present the comparison of the original TPC applications against the synthetic workloads to show the accuracy of the modeling and generation process.

4.3 Characterization of TPCC/TPCE/TPCH

We used the previously discussed models to characterize large-scale instances of TPCC, TPCE and TPCH workloads from real production DC systems. In order to reduce the dimensionality of the problem, we separate the application traces per thread and create a separate model for each thread. We observe the access patterns in the benchmark, along with the spatial and temporal locality characteristics captured by the model. We also evaluate the performance metrics (IOPS and average request latency) based on the information collected from the traces.

Table 1 shows this characterization for the three workloads in a per-thread manner. This separation is performed perceptively at the moment; however, as part of future work, we plan to develop automatic ways to recognize and categorize storage activity in thread types. Each row in Table 1 corresponds to a different thread type in the application. We categorize the threads based on their functionality (i.e. a Log or a Data Thread), activity (low or high I/O request rate) and fluctuation (constant activity or experiencing activity spikes) characteristics.

The categorization in thread types is:

- *Log Thread with high activity and low fluctuation (#0)*
- *Log Thread with low activity and high fluctuation (#1)*
- *Data Thread (SQL Queries) with high activity and low fluctuation (#2)*
- *Data Thread (SQL Queries) with high activity and high fluctuation (#3)*
- *Data Thread (SQL Queries) with medium activity and low fluctuation (#4)*
- *Data Thread (SQL Queries) with low activity and high fluctuation (#5)*
- *Data Thread (SQL Queries) with low activity and low fluctuation (#6)*
- *Data Thread (SQL Queries) with very low activity and fluctuation (#7)*

Specifically TPCC and TPCE have the following thread types:

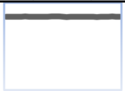
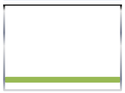
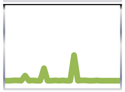
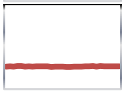
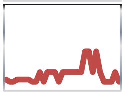
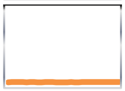
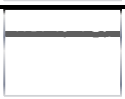
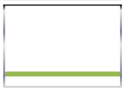
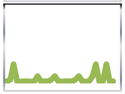
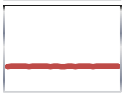
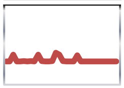
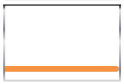
Log #0, ***Log #1***, ***Data #2***, ***Data #3*** and ***Data #6***

While for TPCH the thread types are:

Log #0, ***Data #2***, ***Data #3***, ***Data #4***, ***Data #5***, ***Data #6*** and ***Data #7***

Those are the only combinations of activity-fluctuation pairs observed for the specific applications. For each type of thread we show the characteristics of the I/O requests (rd:wr ratio, percentage of sequential I/Os, average inter-arrival time between requests and average request size), as well as the spatial locality of accesses as captured by the model. Each state corresponds to a portion of the disk space in LBNs, e.g. St1 corresponds to the first 25% of logical block addresses. The thread weight column shows the percentage of requests that correspond to the specific thread. Finally, we show the average performance metrics (throughput and latency) for each thread type. The number of threads that belongs to each thread type for each application is shown in Table 2.

Table 1. Per thread I/O characteristics and performance metrics for TPCC/TPCE/TPCH.

	Thread type	**Fluctuation (11min)**		***Rd:Wr* Ratio**	**% Seq. I/Os**	**Avg Int. Time(ms)**	**Avg Request Size (KB)**	***Spatial Locality***				***Thread Weight***	**Avg. IOPS**	***Avg Lat. (ms)***
								St1	**St2**	**St3**	**St4**			
TPCC	**Total**			**2.2:1**	**2.8**	**0.43**	**9.48**	**85.8**	**1.4**	**12**	**0.4**	**1.00**	**47,082**	**4.13**
			R	**2.2**	0.08	0.08	4.13	88.5	1.5	9.8	0.3	0.68	35,720	6.47
			W	**1**	6.49	4.13	12.31	79.1	1.2	19	0.6	0.32	11,362	0.47
	#0 (Log)			**1:99**	**1.8**	**2.56**	**2.09**	**88.3**	**11**	**0.7**	**0**	**1.2E-4**	**5.87**	**8.91**
			R	1	0.02	3036.1	0.5	0	0	100	0	1.2E-6	0.58	8.97
			W	99	1.8	2.55	2.09	88.9	11	0.2	0	1.2E-4	5.29	8.91
	#1 (Log)			**1:99**	**0.95**	**66.30**	**3.94**	**66.7**	**31**	**2**	**1**	**0.001**	**49.26**	**2.45**
			R	1	0.01	107.8	0.5	75	0	25	0	1E-5	0.49	6.13
			W	99	0.95	65.8	3.94	66.1	31	1.9	1	99E-5	48.77	2.30
	#2 (SQL Queries)			**2:1**	**25.0**	**2.60**	**14.625**	**86.4**	**0**	**14**	**0**	**0.17**	**8,403**	**3.01**
			R	2	28.5	2.57	20.625	87.2	0	13	0	0.119	5,602	4.89
			W	1	18.0	2.72	8.192	99.7	0	0	0.3	0.059	2,802	0.41
	#3 (SQL Queries)			**98:2**	**2.0**	**1.00**	**20.873**	**78.5**	**0**	**21**	**0.7**	**0.242**	**11,394**	**6.01**
			R	98	2.1	1.00	1.024	78.4	0	21	0.7	0.241	11,166	6.05
			W	2	0.3	10.96	65.526	99.4	0	0	0.6	0.001	228	0.56
	#6 (SQL Queries)			**100:1**	**14.7**	**5.27**	**32.893**	**100**	**0**	**0**	**0**	**0.02**	**945.42**	**5.47**
			R	100	14.7	5.27	32.893	100	0	0	0	0.02	945.42	5.47
			W	0	-	-	-	-	-	-	-	0	0	-
TPCE	**Total**			**10.5:1**	**8.15**	**0.78**	**8.41**	**89.8**	**8.4**	**1.7**	**0.3**	**1.00**	**24,694**	**5.55**
			R	10.5	6.04	0.59	8.02	92.5	5.4	2.1	0	0.913	22,546	6.01
			W	1	23.75	6.89	14.68	56.0	27	15	2.1	0.087	2,148	0.69
	#0 (Log)			**1:99**	**11.75**	**9.09**	**2.05**	**88.8**	**10**	**1.0**	**0.3**	**5E-4**	**12.37**	**0.48**
			R	1	0.01	909.14	0.5	76	16.3	7.4	0.3	5E-6	0.12	5.63
			W	99	11.75	9.08	2.05	88.9	11	0.1	0	5E-4	12.25	0.47
	#1 (Log)			**1:100**	**1.42**	**77.1**	**2.10**	**89.0**	**11**	**0**	**0**	**2.2E-5**	**0.54**	**0.56**
			R	0	-	-	-	-	-	-	-	0	0	-
			W	100	1.42	77.1	2.10	89.0	11	0	0	2.2E-5	0.54	0.56
	#2 (SQL Queries)			**7:1**	**20.6**	**7.25**	**6.1**	**92.4**	**7.4**	**0.2**	**0**	**0.077**	**1,897**	**4.63**
			R	7	15.3	3.81	3.8	96.0	4.0	0	0	0.067	1,659	5.35
			W	1	37.9	12.89	16.0	87.7	10	2.3	0	0.009	238	0.47
	#3 (SQL Queries)			**12.5:1**	**16.3**	**4.38**	**8.02**	**91.0**	**8**	**0.2**	**1**	**0.048**	**1,172**	**5.67**
			R	12.5	11.65	2.37	8.02	90.8	8	0.4	1	0.044	1,086	6.13
			W	1	50	12.68	8.01	92	0	0	8	0.004	86	0.89
	#6 (SQL Queries)			**70.4:1**	**15.5**	**15.32**	**8.00**	**91.2**	**6.2**	**2.4**	**0**	**0.035**	**859.3**	**5.35**
			R	70.4	15.6	12.46	8.00	90.6	7.2	2.4	0	0.034	847.3	5.41
			W	1	7.1	287.3	4.01	99	0	0	1	0.0005	12.0	0.65

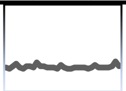
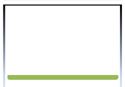
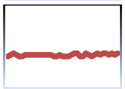
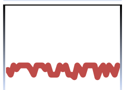
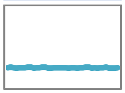
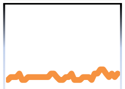
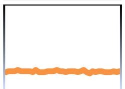
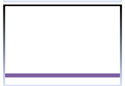
	Thread type	Fluctuation (11min)		*Rd:Wr* Ratio	% Seq. I/Os	Avg Int. Time(ms)	Avg Request Size (KB)	*Spatial Locality*				*Thread Weight*	Avg. IOPS	*Avg. Lat (ms)*
								St1	St2	St3	St4			
TPCH	**Total**			**29.4:1**	**35.5**	**0.327**	**16.13**	**92.7**	**6.2**	**1.3**	**0**	**1.00**	**3,296**	**5.92**
			R	29.4	35.6	0.317	16.19	89.8	10	0.1	0	0.967	3,188	6.01
			W	1	11.7	1.32	1.32	97.8	0	2.2	0	0.328	108.4	0.47
	#0 (Log)			**1:100**	**0**	**173.8**	**4.1**	**100**	**0**	**0**	**0**	**4.8E-6**	**0.016**	**0.44**
			R	0	-	-	-	-	-	-	-	0	0	-
			W	100	0	173.8	4.1	100	0	0	0	4.8E-6	0.016	0.44
	#2 (SQL Queries)			**100:1**	**74.8**	**10.8**	**524.2**	**97.8**	**2.2**	**0**	**0**	**0.012**	**40.4**	**5.31**
			R	100	74.8	10.8	524.2	97.8	2.2	0	0	0.012	40.4	5.31
			W	0	-	-	-	-	-	-	-	0	0	-
	#3 (SQL Queries)			**100:1**	**72.1**	**16.8**	**524.8**	**91.8**	**8.9**	**0.3**	**0**	**0.005**	**18.6**	**4.42**
			R	100	72.1	16.8	524.8	91.8	8.9	0.3	0	0.005	18.6	4.42
			W	0	-	-	-	-	-	-	-	0	0	-
	#4 (SQL Queries)			**99:1**	**63.8**	**48.63**	**128.8**	**95.4**	**4.6**	**0**	**0**	**0.002**	**7.20**	**5.31**
			R	99	63.9	48.5	128.9	95.4	4.3	0	0	0.002	7.19	5.46
			W	1	2.8	202.9	2.14	80	20	0	0	2E-5	0.01	0.46
	#5 (SQL Queries)			**100:1**	**50.9**	**315.8**	**524.8**	**80**	**0**	**20**	**0**	**0.0011**	**3.75**	**5.53**
			R	100	50.9	315.8	524.8	80	0	20	0	0.0011	3.75	5.53
			W	0	-	-	-	-	-	-	-	0	0	-
	#6 (SQL Queries)			**81:19**	**66.8**	**234.9**	**65.5**	**91.8**	**0**	**8.2**	**0**	**6.0E-4**	**1.99**	**5.42**
			R	81	70.2	123.7	65.5	100	0	0	0	4.8E-4	1.61	5.59
			W	19	12.8	869.4	4.01	80	0	20	0	1.2E-4	0.37	0.69
	#7 (SQL Queries)			**100:1**	**25.6**	**134.0**	**65.5**	**100**	**0**	**0**	**0**	**1.4E-8**	**4.6E-5**	**5.53**
			R	100	25.6	134.0	65.5	100	0	0	0	1.4E-8	4.6E-5	5.53
			W	0	-	-	-	-	-	-	-	0	0	-

Table 2. Number of threads and thread weights, for each thread-type for TPCC, TPCE and TPCH.

Workload	Thread type	Number of Threads	Thread Weight /Thread	Thread Weight /Thread Type
TPCC	**Total**	**26**	-	**1.00**
	#0 (Log)	**4**	1.2E-4	4.8E-4
	#1 (Log)	**12**	0.001	0.012
	#2 (Data)	**1**	0.17	0.17
	#3 (Data)	**3**	0.242	0.726
	#6 (Data)	**6**	0.02	0.12
TPCE	**Total**	**35**	-	**1.00**
	#0 (Log)	**6**	5E-4	0.003
	#1 (Log)	**9**	2.2E-5	1.9E-4
	#2 (Data)	**4**	0.077	0.308
	#3 (Data)	**9**	0.048	0.432
	#6 (Data)	**7**	0.035	0.245
TPCH	**Total**	**175**	-	**1.00**
	#0 (Log)	**13**	4.8E-6	6.3E-5
	#2 (Data)	**58**	0.012	0.696
	#3 (Data)	**52**	0.005	0.26
	#4 (Data)	**13**	0.002	0.026
	#5 (Data)	**6**	0.0011	0.006
	#6 (Data)	**11**	6E-4	0.007
	#7 (Data)	**22**	1.4E-8	3E-7

Examining all three TPC applications, we see that read requests dominate. For TPCC that feature is not as evident as for TPCE and TPCH where the rd:wr ratio is over an order of magnitude higher. In terms of I/O features per thread, we present the main insight for each of the three applications:

I.TPCC

TPCC threads are divided between Log and Data threads, with the first accounting for a very small percentage of the total storage activity, while Data requests dominate. In terms of sequential over random I/O characteristics per thread type, we observe that Data #2 (high activity, low fluctuation) has a significant percentage of Sequential I/Os, and experiences minimal fluctuation in its storage activity. On the other hand, random-dominated threads like Data #3 have high fluctuation in their throughput, which might be a result of accessing many different files, and performing more complex queries. Regarding spatial locality, most I/O requests are directed to the first 25% of the storage capacity, with a smaller percentage belonging to St3.

II.TPCE

While TPCC only had 26 threads, TPCE has a significantly larger number of threads in this instance (35). Most of these threads service SQL Queries in the Data partitions of the storage, while 15 threads service Log requests with considerably lower intensity. Log threads, as with TPCC, are write-dominated, and have low throughput requirements, while Data threads are read-dominated and account for the large majority of the workload's storage activity. The differences between the threads' activity are smoother for TPCE than for TPCC, partially due to their larger number, which contributes to better load balance in the system. The main difference between SQL threads, apart from their throughput requirements, is their rd:wr ratio, which is 7:1 for Data #2 and reaches 70.4:1 for Data #6 (low activity, low fluctuation). This difference is appointed to the different type of queries serviced by each type of thread, and is consistent across threads of the same type. TPCE has even higher spatial locality than TPCC, with I/O requests being more aggregated in space, especially when it comes to Data requests. Most of the I/O accesses belonging to higher states are initiated as part of Log threads. This has motivated the TPC community to start using SSDs in order to limit the overprovisioning with respect to spatial locality.

III.TPCH

TPCH has been previously studied [9, 13] in terms of the large number of different queries it is comprised of. Here, we can correlate those types of queries with the underlying storage activity they initiate. In terms of aggregate I/O metrics, TPCH is significantly read-dominated, more than TPCE, and definitely more than TPCC. It is also a lot less intense than the previous two workloads, with average IOPS accounting for 5% of the average throughput of TPCC, and 10% of TPCE. However, the aspect where TPCH deviates most evidently from the I/O-dominated workloads is the number of threads it has, which is 175, almost an order of magnitude more than in either TPCC or TPCE. The interesting characterization of TPCH comes from studying the different types of these threads. Based on the storage activity and files visited, and without knowledge on the semantics of the application, we can deduce the type of query from the 22 TPCH query types [9].

This means that a workload model can represent one or multiple query types and that by assembling the correct mix of per-thread workload models one can create a benchmark with the corresponding combination of queries without the requirement to know the functionality and structure of the application. Table 3 shows this correspondence between thread types and TPCH query types [9].

Table 3. Correlation between Storage Activity and Queries for TPCH.

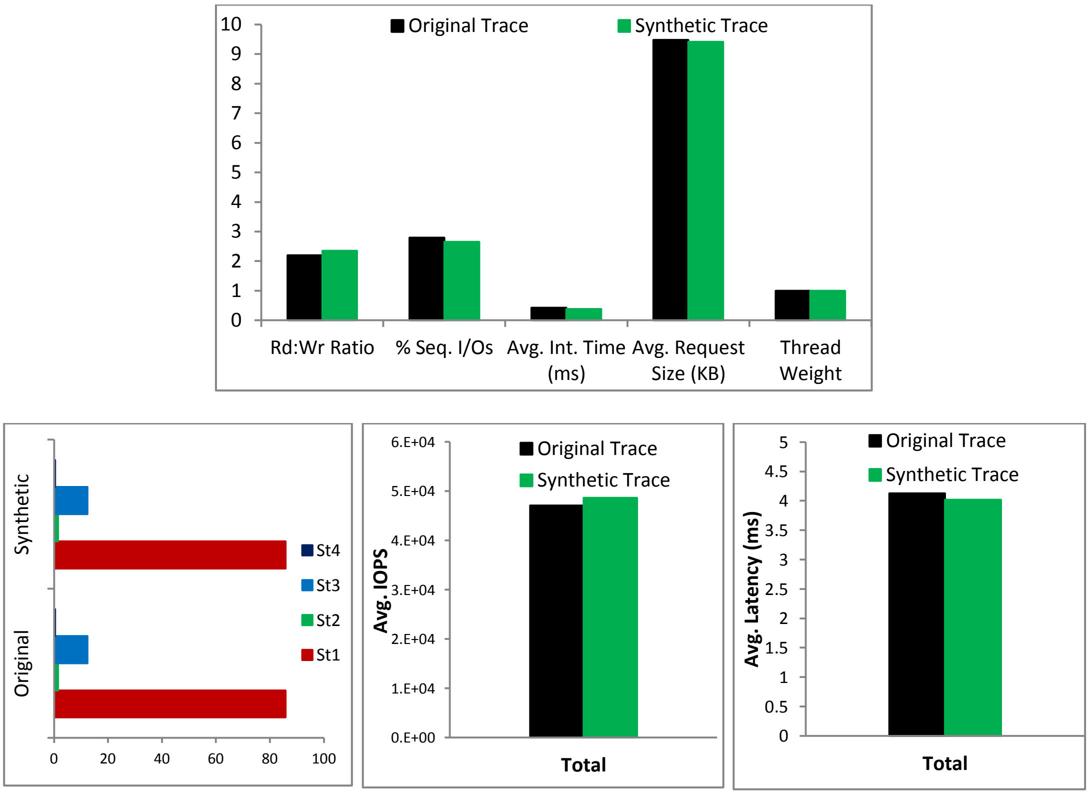
Workload	**Thread type**	**Query Type**
TPCH	**Total**	**Q1-Q8, Q10-Q22**
	#0 (Log)	**-**
	#2 (Data)	**Q1, Q5, Q7-Q10, Q16, Q18**
	#3 (Data)	**Q1, Q5, Q7, Q9, Q13, Q16, Q17**
	#4 (Data)	**Q3, Q5- Q8, Q17, Q19, Q21**
	#5 (Data)	**Q1, Q7, Q13-Q15, Q19, Q20**
	#6 (Data)	**Q2, Q4, Q6, Q14, Q15, Q22**
	#7 (Data)	**Q2, Q4, Q6, Q11, Q12, Q22**

5 Validation

For each thread type, we create a separate workload model and a synthetic workload that resembles its storage activity. We create a thread mix based on the ratios in Table 2 to recreate the aggregate activity of each TPC application. We then run the synthetic workload using our framework and compare it against the original TPC applications. For our experiments we use real DC traces from a large-scale deployment of the TPC benchmarks, running on MS SQL Server. The synthetic workloads are run on a TPC-provisioned server with two sockets, 8 cores, 16GB of memory and 28 disks (5.4TB total storage) organized in RAID1+0 configuration. The metrics of interest are I/O characteristics per thread, as well as performance metrics (throughput and latency).

Figure 2(a) shows the validation of I/O features (Rd:Wr Ratio, % of Seq. I/Os, Avg. Inter-arrival Time, Avg. Request Size, Thread Weight), and Figure 2(b) the comparison in spatial locality between original and synthetic application for TPCC. The x-axis in Figure 2(a) shows the different I/O characteristics and the y-axis the value of the corresponding feature. The different bars in Figure 2(b) show the LBN ranges (i.e. each bar is 25% of the disk capacity). As we showed in Section 4.3 the vast majority of requests happen in State 1, while State 2 and State 3 have significantly lower percentages of I/O accesses. Figures 2(c) and 2(d) show the comparison for throughput and latency between original and synthetic application. For all metrics, the deviation is marginal (less than 5%), which shows that the model accurately captures the behavior of the workload. Since the model is based on the logical level of the disk configuration, it is not dependent on the underlying physical layout. However, major changes in the database implementation (OS, database system) affect the model and will require re-training the state diagrams.

Figure 2: Validation of ***(a) I/O characteristics***, ***(b) Spatial Locality*** and ***(c, d) Performance Metrics*** between Original and Synthetic Trace for TPCC.



The per-thread results for all three applications are shown in Figures 3, 4 and 5. From left to right we show how the I/O features (Rd:Wr Ratio, %of Seq. I/Os, Avg. Inter-arrival Time, Avg. Request Size, Thread Weight), spatial locality and throughput and latency compare between original and synthetic application. For TPCC we show the comparison for all thread types and for TPCE and TPCH for the entire application, one Log thread and one Data thread. The results are similar for the other thread types as well. In all cases, the deviation is **less than 5%,** which demonstrates the accuracy of our modeling approach and generation process, even across threads with vastly different characteristics, like the low-activity, write-dominated Log threads and the high-activity, read-dominated Data threads. The error bars in each graph show the deviation between threads of the same type, in both original and synthetic applications.

For these experiments, we have used **2 levels** in the workload model. In order to decide on the optimal number of levels, we perform a sensitivity study; we increase the number of levels until performance stabilizes (less than 2% difference in throughput). This offers the highest amount of information with the least necessary complexity in the model. Additional level of detail is possible, and can reveal more fine-grained access patterns for the TPC workloads, but its study exceeds the scope of this work.

Figure 3: Validation of ***I/O characteristics***, ***Spatial Locality*** and ***Performance Metrics*** between Original and Synthetic Trace for: ***Log #0*** (1st row), ***Log #1*** (2nd row), ***Data #2*** (3rd row), ***Data #3*** (4th row) and ***Data #6*** (5th row).

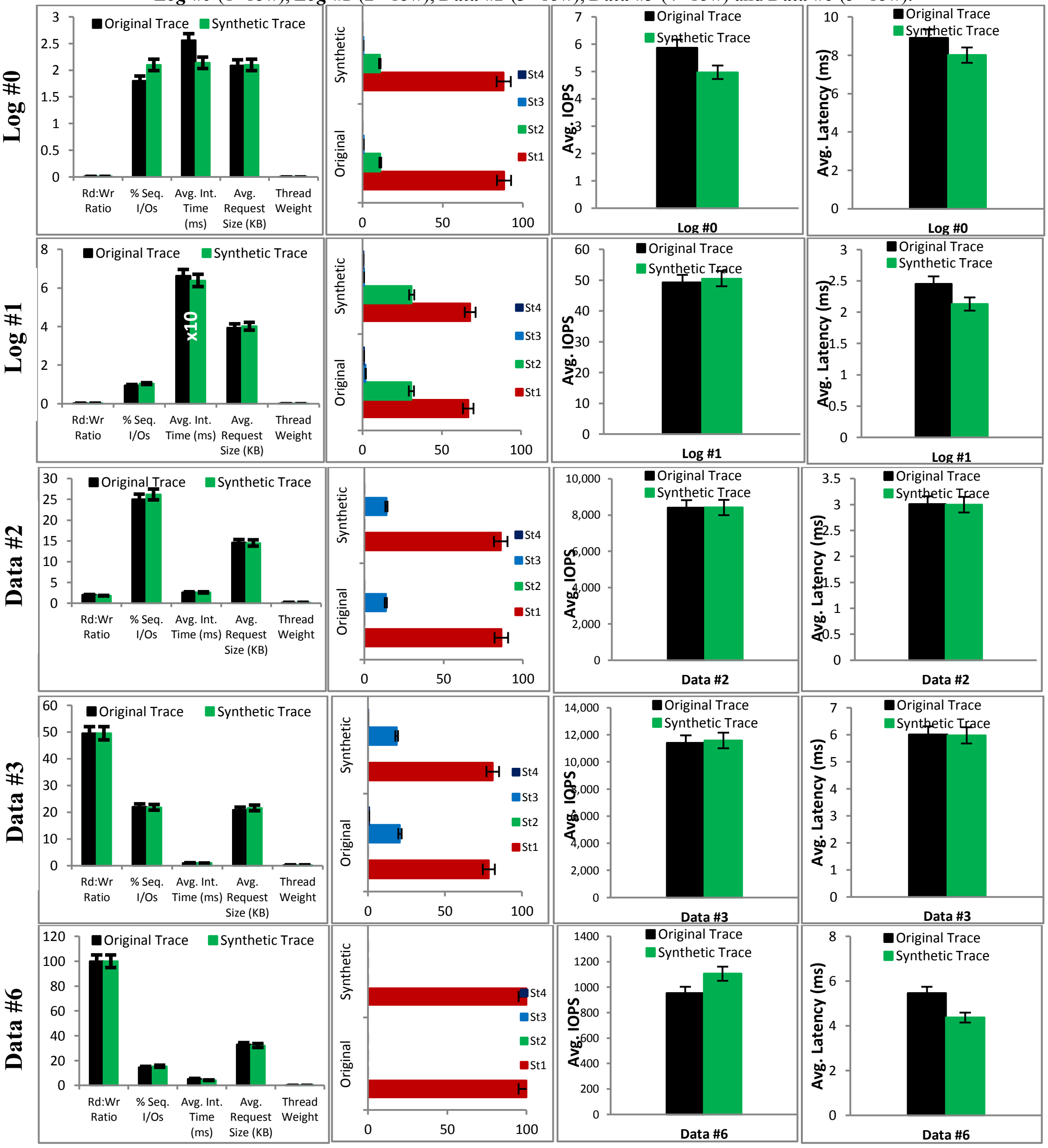


Figure 4: Validation of ***I/O characteristics***, ***Spatial Locality*** and ***Performance Metrics*** between Original and Synthetic Trace for: the ***entire TPCE*** *benchmark* (1st row), ***Log #0*** (2nd row), ***Log #1*** (3rd row).

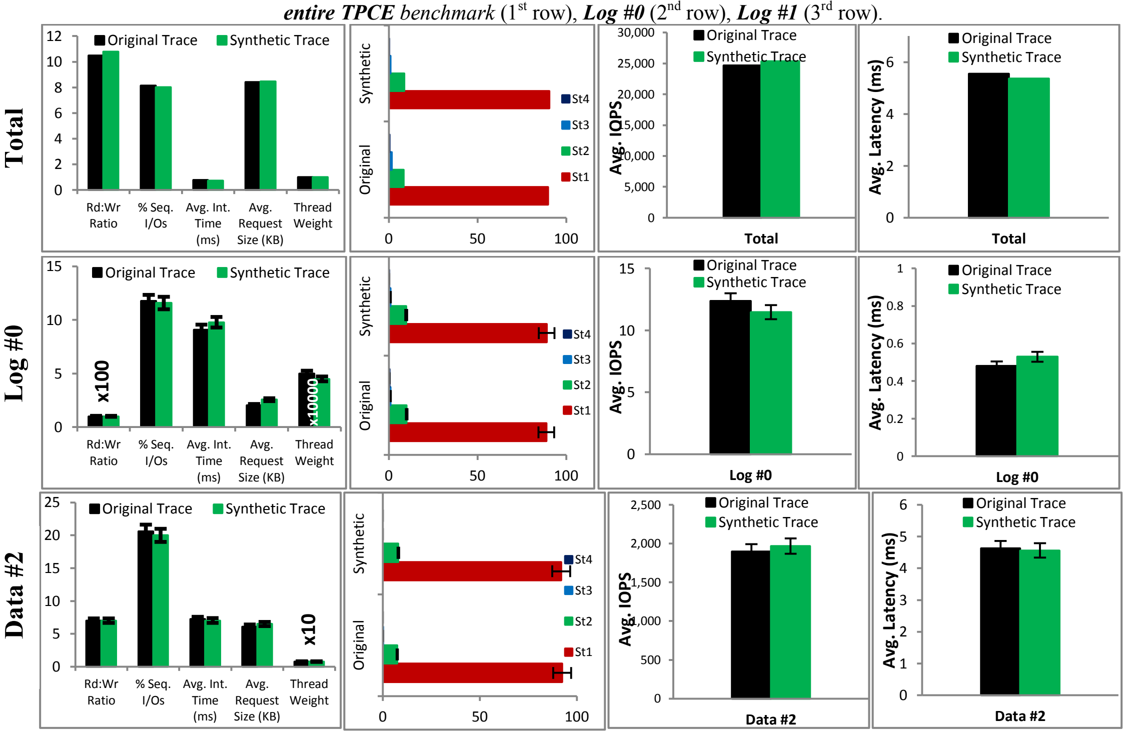
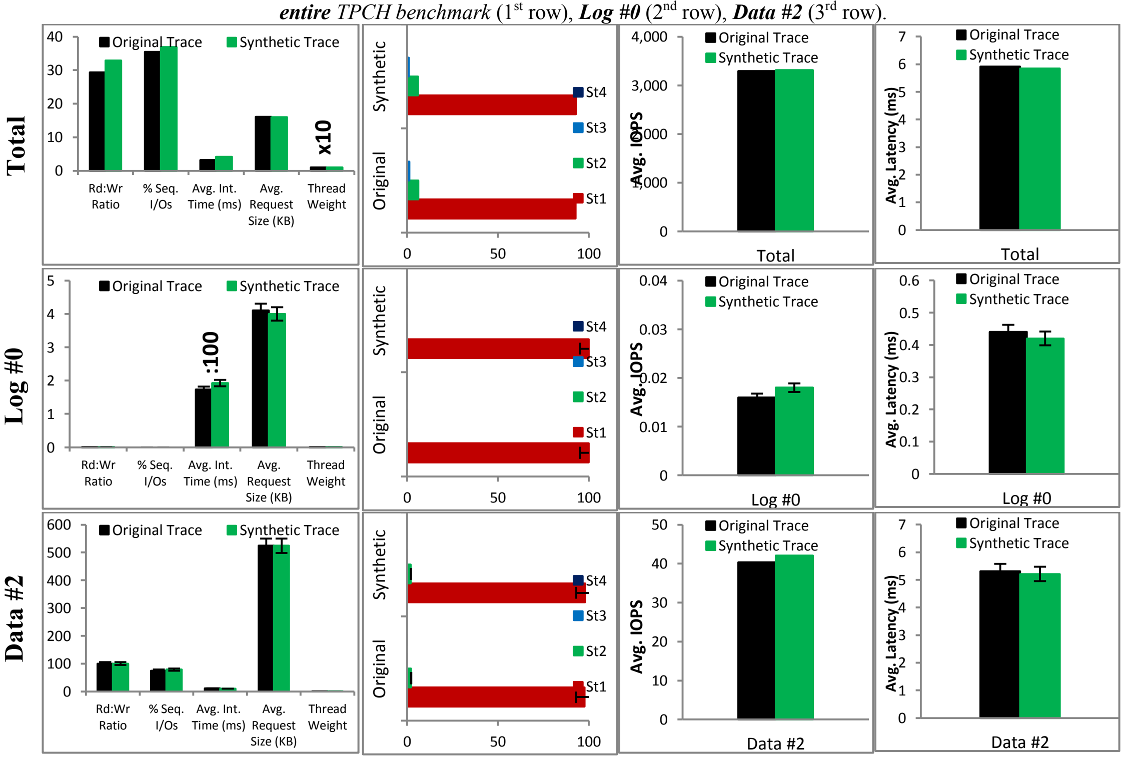


Figure 5: Validation of ***I/O characteristics***, ***Spatial Locality*** and ***Performance Metrics*** between Original and Synthetic Trace for: the ***entire*** *TPCH benchmark* (1st row), ***Log #0*** (2nd row), ***Data #2*** (3rd row).



6 Use Cases

6.1 Identifying Early Storage Problems

Providing a framework for characterization and replay of large-scale TPC workloads enables studies that can identify early problems with datacenter-scale storage systems. Infant mortality is a prominent issue in large-scale systems and results in significant cost in terms of infrastructure and effort to be detected and resolved. Running the models for TPCC/E/H, which offer a time-compressed version of the workload, can identify such problems very early on, thus saving significant portions of TCO and improving the reliability and availability of the system. It can also identify access patterns (i.e. thread types) that increase the risk of causing reliability problems.

Furthermore, the intensity knob included in the generation tool is helpful in that it can increase the I/O request rate and stress test the endurance of hard disks. More importantly it allows a fast identification of such errors, which eliminates faulty disks in the system and improves its longevity.

6.2 SSD Caching for TPC Workloads

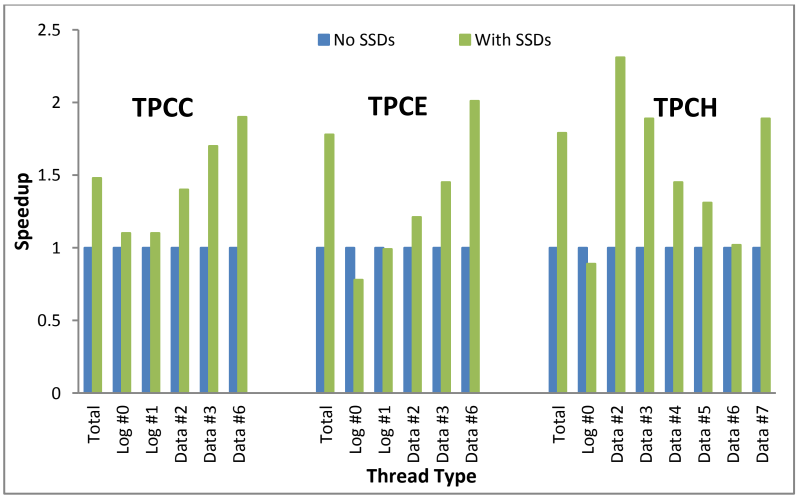
Studying the spatial locality of the TPC workloads, per thread, has revealed that most accesses are limited to a small range on disk, while the vast majority of the storage capacity remains unutilized. This justifies the attention that SSD-based or HDD/SSD hybrid storage systems have recently attracted. Using our framework, we can *predict* the performance gains from using an SSD-based storage configuration or a system with SSD caches, for TPC workloads. Furthermore, since the experiments are very fast to perform, one can examine many more options, than previously possible, in search of an optimal storage configuration for a workload.

We have used our models for TPCC, TPCE and TPCH to evaluate the performance gains in a system with SSD caching (4 SSD caches, 8GB each [12]). For this, we have used the same disk I/O traces as before, which implies that we assume no change in the intensity of the workload when we switch to the SSD-based system. This is not necessarily the case, since we expect I/Os to arrive at a faster pace when there are SSDs in the system. However, we have chosen to maintain the previous features for the application, to compare the performance metrics more consistently. We plan to evaluate a retuned version of the applications in the faster storage system, after determining that subsequent I/O request are independent, in which case simply increasing the intensity of the requests, is what we would see in the SSD-based system.

Figure 6 shows the results of this experiment in terms of per thread and aggregate speedup when we enable SSD caching, for each of the three applications.

We observe that for threads that are more intense in I/O requests, the improvement from adding the SSD caches is more significant. This is in agreement with the fact that SSDs are more beneficial in very intense workloads, while disks are preferred for less frequent I/O requests. This also takes into account the fact that SSDs mainly benefit read-dominated thread types like *#2, #3* and *#7* for TPCH, while the write dominated Logs (*#0*, *#1*) and Data *#6* do not experience as high an improvement, with performance even worsening for Log #0 and #1 in TPCE. We expect that tuning the intensity of the workload to resemble the behavior in an SSD-based system, would accentuate even more the difference in performance benefit between ***intense, read-dominated*** and ***not intense, write-dominated*** thread types.

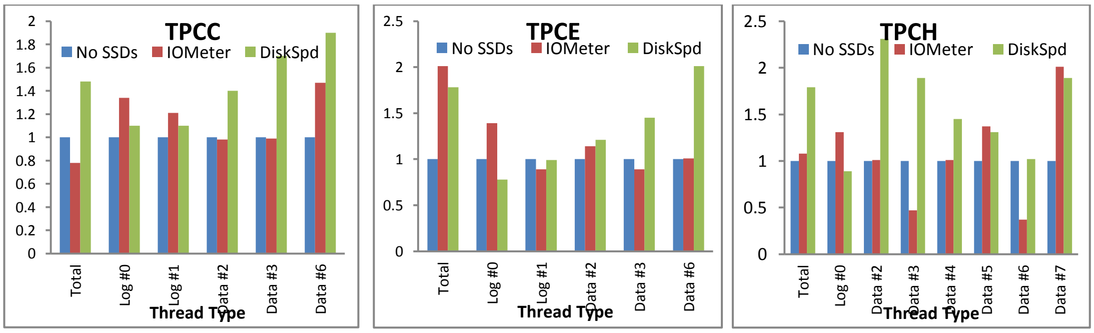
Figure 6. Performance before and after the use of SSDs for TPCC/E/H.



To show the need to capture spatial and temporal locality in the model we compare our results with those obtained by the most well-known open-source workload generator, IOMeter. IOMeter, does not have any notion of locality of I/O requests, in which case it is oblivious as to which blocks should be cached in SSDs. Therefore using it to study SSD caching should not demonstrate an improvement in performance. Indeed, as shown in Figure 7 there is a significant difference in the behavior of the two tools. DiskSpd knows which blocks experience temporal locality, therefore caches them in the SSDs, while IOMeter does not. The performance improvement is evident when using DiskSpd for the storage system evaluation, while IOMeter is not helpful in identifying the speedup from adding the SSDs to the system.

This experiment can also help with identifying the reasons behind the variability in write performance observed in systems with SSD storage. Adding SSDs in the system, has consistently made predicting write performance a lot more difficult than with HDDs, and as SSDs become more affordable, this problem is expected to aggravate. Characterizing the application, as performed in Section 4.3, can help with identifying the threads that are write-dominated and thus cause the performance variability, while running experiments using the workload generator can replicate and identify the reasons for this variability. It can also help with documenting the sensitivity of SSDs in write I/O intensity by increasing the rate at which write I/Os are issued and quantifying the impact on performance, and performance variability.

Figure 7. Comparison between IOMeter and DiskSpd for TPCC/E/H when using SSD caching.



7 Conclusions and Future Work

In this work we perform a detailed, per-thread characterization of the storage activity of the TPCC, TPCE and TPCH workloads, and provide insight on their behavior and access patterns. We propose a modeling and generation framework that greatly reduces the time to setup and perform storage experiments in large-scale instances of the TPC benchmarks. We demonstrate the accuracy of our methodology against the original applications, compare it to previous schemes, and present two possible use cases for the framework, early disk error detection and SSD caching. We believe that this framework offers a detailed understanding on the storage activity of TPCC/E/H, while decoupling performing datacenter storage studies from the overhead of setting up the application and the requirement to deploy it in all possible storage system configurations. We plan to demonstrate the applicability of our methodology by evaluating additional use cases for large-scale storage systems.

References

1. S.Sankar, K.Vaid. ”Storage characterization for unstructured data in online services applications”. In *Proceedings* of the IEEE International Symposium on Workload Characterization (IISWC), Austin, TX, 2009.
2. S. Kavalanekar, B. Worthington, Q. Zha, V. Sharda ”Characterization of storage workload traces from production Windows servers”. In Proceedings of IEEE International Symposium on Workload Characterization (IISWC) 2008, Seattle, WA, Sept. 2008.
3. C.Delimitrou, S.Sankar, K.Vaid, C.Kozyrakis. "Accurate Modeling and Generation of Storage I/O for Datacenter Workloads". In *Proceedings of the 2nd Workshop on Exascale Evaluation and Research Techniques (EXERT)*, Newport Beach, CA, March, 2011.
4. S. Sankar, K. Vaid. ”Addressing the stranded power problem in datacenters using storage workload characterization”. In Proceedings of the first WOSP/SIPEW International conference on Performance Engineering, San Jose, CA, 2010.
5. DiskSpd: File and Network I/O using Win32 and .NET API’s on Windows XP http://research.microsoft.com/en-us/um/siliconvalley/projects/sequentialio/
6. IOMeter, performance analysis tool. http://www.iometer.org/
7. C. Kozyrakis, A. Kansal, S. Sankar, and K. Vaid, "Server Engineering Insights for Large-Scale Online Services". In IEEE Micro, vol. 30, no. 4, July 2010.
8. S. Kavalanekar, D. Narayanan, S. Sankar, E. Thereska, K. Vaid, B. Worthington, ”Measuring Database Performance in Online Services: a Trace-Based Approach”. In Proceedings of the First TPC Technology Conference on Performance Evaluation & Benchmarking (TPC TC), Lyon, France, 2009.
9. TPC BENCHMARK-H. (Decision Support). Standard Specification. Revision 2.14.0. TPC Council. San Francisco, 2011.
10. SQLIO Disk Subsystem Benchmark Tool. http://www.microsoft.com/downloads/en/details.aspx?familyid=9a8b005b-84e4-4f24-8d65-cb53442d9e19&displaylang=en
11. H. Vandenbergh. Vdbench: User Guide. Version: 5.00 October 2008. http://garr.dl.sourceforge.net/project/vdbench/vdbench/Vdbench%205.00/vdbench.pdf
12. Adaptec MaxIQ. 32GB SSD Cache Performance Kit. http://www.adaptec.com/en-US/products/CloudComputing/MaxIQ/SSD-Cache-Performance/
13. J. Zhang, A. Sivasubramaniam, H. Franke, N. Gautham, Y. Zhang, S. Nagar. “Synthesizing Representative I/O Workloads for TPC-H”. In Proceedings of the International Symposium on High Performance Computer Architecture (HPCA), Madrid, Spain, February 2004.