Decoupling Datacenter Studies from Access to Large-Scale Applications: A Modeling Approach for Storage Workloads

Christina Delimitrou*, Sriram Sankar†, Kushagra Vaid† and Christos Kozyrakis*
*Electrical Engineering Department, Stanford University, {cdel, kozyraki}@stanford.edu
†Global Foundation Services, Microsoft, {srsankar, kvaid}@microsoft.com

***Abstract*—The cost and power impact of suboptimal storage configurations is significant in datacenters (DCs) as inefficiencies are aggregated over several thousand servers and represent considerable losses in capital and operating costs. Designing performance, power and cost-optimized systems requires a deep understanding of target workloads, and mechanisms to effectively model different storage design choices. Traditional benchmarking is invalid in cloud data-stores, representative storage profiles are hard to obtain, while replaying the entire application in all storage configurations is impractical both from a cost and time perspective. Despite these issues, current workload generators are not able to accurately reproduce key aspects of real application patterns. Some of these features include spatial and temporal locality, as well as tuning the intensity of the workload to emulate different storage system configurations.**

To address these limitations, we propose a modeling and characterization framework for large-scale storage applications. As part of this framework we use a state diagram-based storage model, extend it to a hierarchical representation and implement a tool that consistently recreates I/O loads of DC applications. We present the principal features of the framework that allow accurate modeling and generation of storage workloads and the validation process performed against ten original DC applications traces. Furthermore, using our framework, we perform an in-depth, per-thread characterization of these applications and provide insights on their behavior. Finally, we explore two practical applications of this methodology: SSD caching and defragmentation benefits on enterprise storage. In both cases we observe significant speedup for most of the examined applications. Since knowledge of the workload's spatial and temporal locality is necessary to model these use cases, our framework was instrumental in quantifying their performance benefits. The proposed methodology provides a detailed understanding on the storage activity of large-scale applications and enables a wide spectrum of storage studies without the requirement for access to real applications and full application deployment.

I. Introduction

With the advent of social networking and cloud data-stores, user data is increasingly being stored in large-capacity and high-performance storage systems. These systems account for a significant portion of the total cost of ownership (TCO) of a datacenter (DC) [12], [3]. Specifically, for online services, data retrieval is often the bottleneck to application performance [11], [12], making efficient storage provisioning a first-order design constraint. One of the main challenges when trying to evaluate storage system options is the difficulty in replaying the entire application in all possible system configurations. The effort itself can be highly inefficient from the time and cost perspective, given the scale of DC deployments (hundreds of TBs, over tens of thousands of servers). It is hence imperative to invest in frameworks that allow for extensive workload analysis, characterization and modeling.

Large-scale Online Services differ from conventional applications in that they cannot be approximated by single machine benchmarking, due to patterns that emerge from user behavior in a large-scale environment. Furthermore, privacy concerns make source code, user behavior patterns and datasets of DC applications rarely available to storage system designers. This makes the development of a representative model that captures key aspects of the workload's storage profile, even more appealing. Once such a model is available, creating a tool that reproduces the application's storage behavior via a synthetic access pattern will enable large-scale storage studies, decoupled from the requirement to access application code.

Despite the merit in this effort, previous work on I/O workload generation lacks the ability to capture the spatial and temporal locality of I/O access patterns, causing them to significantly deviate from the application's real characteristics. In this work, we provide a framework for research on large-scale storage systems that addresses these issues. This infrastructure includes probabilistic, state diagram-based models that capture information of *configurable granularity* on the workload's access patterns. We develop the models from production traces of real DC applications based on previous work [11]. We extend these models to a granular, hierarchical representation in order to identify the optimal level of detail for each application. Then we perform an in-depth, per-thread characterization of the storage activity of ten large-scale DC applications in terms of the functionality, intensity and fluctuation of I/O behavior. To the best of our knowledge this is the first study at a per-thread granularity for large-scale applications, including information on their spatial locality. Furthermore, we design a tool that recognizes these models and recreates synthetic access patterns with I/O features that closely match those of the original applications. We perform extensive validation of our methodology to ensure resemblance between original and synthetic loads in both I/O characteristics and performance metrics.

The main features we introduce for accurate I/O generation are:

1) The ability to issue I/Os with specified inter-arrival times, both static and following time distributions.
2) The ability to preserve the spatial and temporal locality of I/O accesses, as well as the features and weights of each transition in the state diagram.
3) The ability to modify the intensity of the generated I/Os by scaling the inter-arrival time of I/O requests. This enables high performance storage systems evaluations (e.g., Solid State Drives).

We use our methodology (model and tool) to evaluate two important DC storage design challenges. Firstly, we explore the applicability of Solid State Devices (SSD) caching in DC workloads. Using our tool, we show that for most of the examined DC applications, SSD caching offers a significant storage system speedup without application change (31% on average for a 32GB SSD cache). In the second use case, we motivate the need for defragmentation in the DC. We observe that user data gets accumulated over a period of time and files get highly fragmented. Using this information from tracing [5], we rearrange blocks on disk in order to improve the sequential characteristics of the workloads. Using the tool to run the defragmented traces we show that defragmentation offers a significant boost in performance (18% on average), in some cases even greater than incorporating SSDs.

Succinctly, the main contributions of this work are:

- We present a concise statistical model that accurately captures the I/O access pattern of large-scale applications including their spatial locality, inter-arrival times and type of accesses. It is also hierarchical, which allows configurable level of detail to accommodate the features of each application.
- We implement a tool that recognizes this model and recreates synthetic access patterns with same I/O characteristics and performance metrics as in the original application. No previous storage tool (e.g., IOMeter) can simulate spatial and temporal locality of DC workloads.
- This methodology enables storage system studies that were previously **impossible without full application deployment and without access to the real applications**. We demonstrate the applicability of our tool in evaluating SSD caching and defragmentation. These spatial locality-based studies have been unexplored due to lack of a tool that allowed their evaluation.

The rest of this paper is structured as follows. Section 2 discusses related work. Section 3 presents a description of the model and the tool's implementation. Section 4 includes a per-thread characterization of the storage activity of the DC applications, the methodology's validation and a comparison of our toolset with a popularly-used workload generator (IOMeter). Section 5 discusses the tool's applicability in evaluating two important DC storage challenges. Finally, Section 6 presents topics for future work and concludes the paper.

II. Related Work

Significant prior work [9] has studied how to efficiently provision DC storage systems. However, a necessary requirement towards efficiently configuring the storage system is studying DC workloads. A convenient approach for that should involve a model that captures the workload's representative features and a tool that accurately recreates its access patterns.

Despite this, most prior large-scale storage configuration techniques are mainly empirical, based on the workload's characteristics as derived from traces [7]. Kavalanekar et al [7], [8] use a trace-based approach to characterize large online services for storage system configuration and performance modeling. 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. Regenerating I/O workloads with high fidelity can offer far richer information towards understanding the behavior of workloads whose implementation remains largely unknown. It also enables addressing instrumental challenges in storage system design (e.g. incorporating SSDs, defragmentation, placement/migration of hot data) when optimizing for performance and efficiency.

IOMeter [6], SQLIO [13], Vdbench [15] are all open-source generators of disk I/O loads. IOMeter allows for specific I/O characteristics to be defined, SQLIO simulates aspects of the disk load of the Microsoft SQL Server, while Vdbench apart from the feature of I/O generation is equipped with the capability of trace replay. Finally, a workload generator relying on online histograms in a virtual machine over VMWare's ESX Server [2] captures information on disk I/O without significant CPU or latency overheads. However, all these workload generators lack the ability to exploit the temporal and especially the spatial locality of DC applications. Spatial and temporal locality are extremely important for DC applications, due to the different behavior of storage devices with and without locality. Ignoring locality can result in greatly misleading results in performance, power and TCO.

Finally, where applicable, these tools are based on outstanding I/Os instead of inter-arrival times. However, the latter offers a better representation of the workload's behavior [10], decoupled from the system that hosts it. For our work, we extend the functionality of DiskSpd [4], an I/O workload generator, in ways that enable us to recreate representative DC loads.

III. Modeling and Generation Process

A. Basic State Diagram Model

Our approach requires a model that captures the I/O features and locality of storage activity from the application's point of view. This means that the model needs to cluster accesses based on their spatial locality and characteristics. For this purpose we use the Markov Chain representation proposed by Sankar et al. [11]. The models are trained based on real storage traces from production servers of a large-scale datacenter deployment, and can capture I/O accesses in one or multiple servers, and from one or multiple users. According to the model, states correspond to ranges of logical blocks on disk (LBNs) and transitions represent the probabilities of switching between LBN ranges. Each transition is characterized by a set of features that reflect the workload's I/O behavior and consist of the block size, randomness, type of I/O (read, write) and inter-arrival time between subsequent requests.

The insight behind the model's structure is that spatial locality is represented by the clustering of I/Os corresponding to the same state, and temporal locality (i.e., subsequent I/Os) is represented by the transitions between states in the model. Therefore, it provides a comprehensive and modular representation of the workload's I/O behavior. The probability for each transition is calculated as the percentage of I/Os that

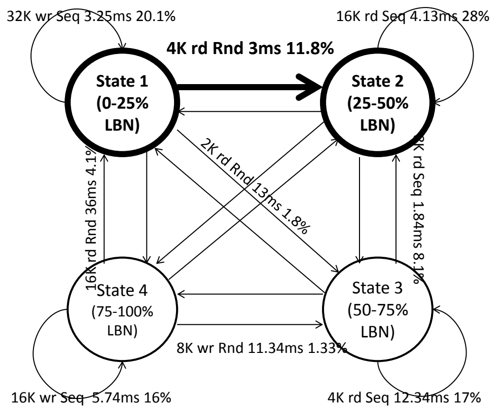


Fig. 1: One level State Diagram

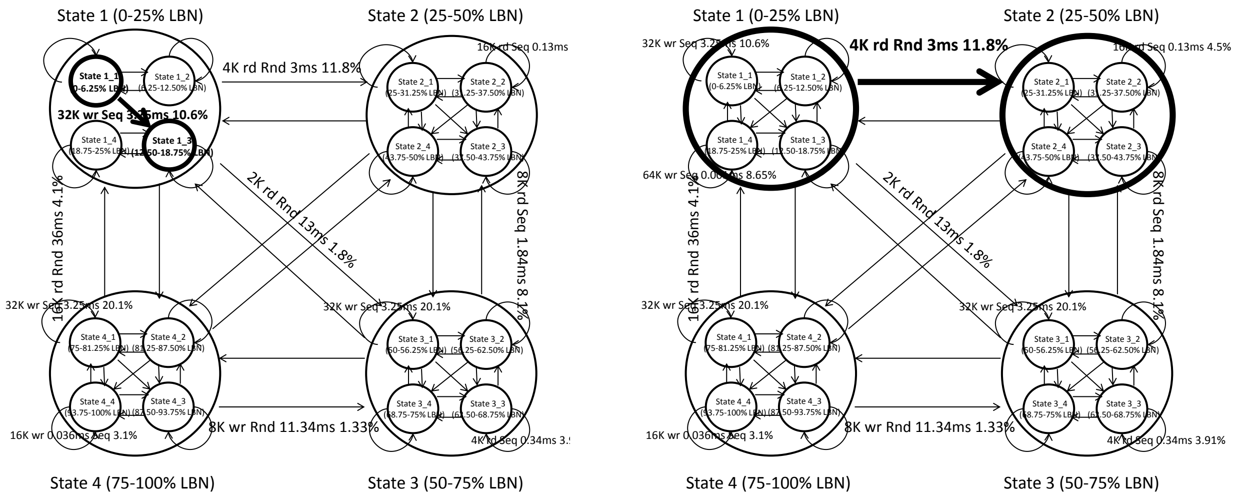


Fig. 2: Two level State Diagram: (a) Transition between minor states and (b) Transition between major states

correspond to it. Figure 1 demonstrates a simplified form of the state diagram with four states, each of which corresponds to 25% of the total LBNs. The model works as follows (highlighted part of the diagram): If an I/O corresponds to State 1, there is an 11.8% probability that the next I/O will be a 4K read, random access with an inter-arrival time of 3ms that corresponds to State 2.

B. Hierarchical State Diagram Model

Different applications have different access patterns, some requiring more detail than others to be accurately captured. To convey information of finer granularity, we have extended the previous model to a hierarchical representation. Figure 2 demonstrates one such model with two levels. To build a two-level model each state in the one level diagram is subdivided in four states and becomes a new state diagram. The two-level diagram will have 16 states. Here LBNs are divided in four states. In general, the number of states per level is chosen such that the probabilities of transitions are minimized.

Perhaps counter-intuitively, the number of transitions in the new diagram is not 256 but 76. As shown in Figure 2, level-two (fine-grained) transitions only exist within the large states but not across them. This means that a transition happens either between two minor states (Figure 2(a)) or between two major states (Figure 2(b)). This approach exploits the fact that spatial locality is mostly confined within states. The number of transitions for a given level is given by

$$4^{l-1}16 + \sum_{i=1}^{l-1} 4^{i-1}12 \qquad (1)$$

while for the flat model it is given by 16^l, where l is the number of levels.

Table I shows how the number of states and transitions scales for up to 10 levels. It becomes obvious that for the flat representation the number of transitions increases exponentially with the number of states, while the hierarchical model has a linear relation with the state count. This choice does not cancel the value of a flat model, but rather proposes that a hierarchical model is just as beneficial without making the number of transitions intractable. Comparing the throughput of models constructed with the hierarchical and the flat representations shows less than 5% difference in throughput.

The number of levels reflects the complexity of an application's access pattern and as shown in the validation section (V.C), finer granularity is instrumental to accurately represent

Levels	State Count	Transition Count	
		Hierarchical Model	Flat Model
1	4	16	16
2	16	76	256
3	64	316	4096
4	256	1276	65536
5	1024	5116	1048576
10	1048576	5242876	109951162776

TABLE I: Scalability of the model in terms of number of states and transitions with an increasing number of levels.

some applications. The proposed model structure guarantees scalability even for applications that require many levels.

C. Storage Activity Fluctuation

It is well known that DC applications experience high fluctuations in their activity, with peak activity usually occurring throughout periods of the day and low activity be present throughout the night. In order to generate representative storage workloads of real applications one must account for this effect. Studying the fluctuation of the storage activity for DC applications reveals that the main feature that changes is the intensity of the I/O requests, i.e. throughout specific periods of an application's lifecycle inter-arrival times decrease significantly. Other features however, like the spatial locality and size of the requests are self-similar throughout this lifecycle [10]. In order to account for this fluctuation, we calculate the inter-arrival times over shorter periods of activity and progressively switch between workload intensities. Essentially each model is composed by multiple models of different intensities that capture the transient features of the I/O requests.

D. Generation Tool Design - DiskSpd

The model, previously discussed, is the first step in recreating accurate DC I/O loads. The second step, involves a tool that recognizes the model and generates storage workloads with high fidelity, using some configuration knobs.

For this purpose we use DiskSpd, a tool that started as a means to measure disk I/O bandwidth and expanded to a complete workload generator [4]. It performs read and/or write I/Os in burst mode on either disks or files, given the I/Os' block size, randomness, and initial block offset. The former consist of a subset of the most relevant features of DiskSpd for the current study. Other features include controlling system parameters such as hardware or software (OS) caches, thread affinity, number of outstanding I/Os, etc.

To recreate a representative workload using the model previously discussed, we have introduced a series of features in DiskSpd. The following subsections describe these features.

1) Inter-arrival Times (Average and Distributions): Studying real application traces has shown that burst mode I/O accesses, though present for short periods of time, are not the norm and do not dominate an application's lifetime. Subsequent I/Os tend to have well-defined time margins between them. Narayanan et al [10] have shown that inter-arrival times are a critical feature of I/O behavior, especially in DC applications that experience high peaks and low troughs throughout their execution. Multiple studies quantify the magnitude of this metric and explore the differences among DC workloads [11], [12].

To demonstrate these margins between accesses of specific block ranges, we implement inter-arrival times in DiskSpd, which are calculated for each transition and measured in ms. Enabling inter-arrival times also means disabling the simultaneous tuning of outstanding I/Os since the two are incompatible, with the former ensuring an "idle" period of time between I/Os and the latter ensuring a defined number of on-the-fly I/Os in the queues.

The use of inter-arrival times instead of outstanding I/Os in a workload generator is *first proposed here*. Previous tools are based on defining the system's queue length. However, queued I/Os do not characterize an application as well as inter-arrival times, since queue length is a system feature, while I/O intensity a workload feature. The difference between the two becomes clearer in the case where we want to create a more intense workload as described in Section III.D.3.

Furthermore, in order to capture the variations in storage intensity we have added the feature of inter-arrival time distributions, i.e., during the workload's execution inter-arrival times can follow one of the following distributions: normal, exponential, Poisson and gamma. This permits a closer resemblance to the fluctuations of a workload's intensity throughout its lifetime, as well as the capture of burst I/Os.

In the default version of the tool, the mean for normal distribution (μ), the rate parameter for exponential (λ), the expected number of occurrences for Poisson (λ) and the scale parameter (θ) for gamma correspond to the mean inter-arrival time calculated from the traces.

2) Multiple Threads and Thread Weights: Access patterns of real applications have distinct per transition characteristics. In order to recreate an I/O load using the state diagram model, we have added the feature of executing threads with different I/O characteristics each (block size, randomness, type (rd/wr), target LBN range and inter-arrival times). Each thread corresponds to a transition in the state diagram and maintains the original access pattern via the notion of *thread weight*, i.e., the proportion of I/O accesses that correspond to each thread. During the threads' execution, we ensure that thread weights are satisfied with less than 0.05% deviation from the target weights by adjusting their "idle" time.

This mechanism might seem redundant if one considers thread weights as a straightforward translation of inter-arrival times. Although inter-arrival times are a strong indication of the proportions of accesses for a transition, there are cases where fast I/Os are not common, but are confined in a short period of the application's execution. In this case, simply maintaining inter-arrival time will not ensure the transition's weight. Although these events are rare, this mechanism ensures that thread weights are maintained in all cases.

Furthermore, in order to guarantee that thread weights are satisfied throughout the workload's execution we perform a Round Robin visit in states so that all threads are active in

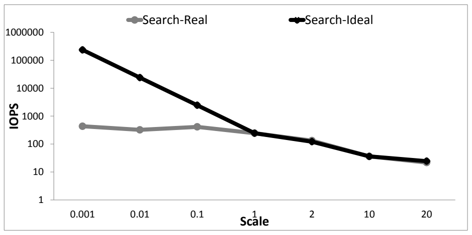


Fig. 3: Throughput Scaling for Different Workload Intensities.

different phases during the program's execution instead of limiting them in an arbitrary period of activity. This way the synthetic trace becomes a compressed version of the original workload. Although this does not cover all possible transition patterns, self-similarity tests [8] in original DC applications have verified that indeed the spatial characteristics of I/Os are consistent across time.

3) Intensity Knob: One of the main objectives behind developing this tool is to evaluate different storage system configurations. Although when referring to disks, inter-arrival times are within a few milliseconds or tenths of a millisecond, when switching to SSDs that number is expected to fall dramatically, since I/Os are expected to arrive at a higher rate. Current production traces do not have such intensity; however, we expect workloads to be tuned to faster storage systems using SSDs. In order to replay workloads compatible with such systems, we have added an intensity knob that scales inter-arrival times down or up to increase or decrease their intensity respectively. This feature clarifies the distinction between outstanding I/Os and inter-arrival times. Queuing more I/Os does not emulate a faster storage system, since in an SSD-based system, for example, I/Os do not simply get queued in larger numbers, they also get serviced faster. Maintaining the outstanding I/Os queue length in this case, stresses the storage system out of proportion and is not useful for DC scaling studies. Having this knob offers the opportunity to evaluate a storage system configuration based on the workload's expected intensity margins. It also enables studies that scale the number of users that initiate requests in the system.

For example, in a Hard Disk-based system when intensity exceeds the system's queues' capabilities, throughput levels-off. Figure 3 shows this inability of the HDD system to service high-rate requests. Smaller scale corresponds to a more intense workload. Although we assume that I/Os will not be dropped, unless timeouts are present in the system, the application can still not meet its intensified performance requirements. The use of SSDs for storage is motivated, among others, by this performance limitation of the HDD-based system.

An important note to make here is that our work is based on an open-loop approach, which means that applications are not retuned when we switch to an SSD-based system. This potentially underestimates the benefit from the use of SSDs, but offers a more concise comparison between the capabilities of the storage systems, since all other parameters remain constant. A second assumption is that, in order to use the same models as in the HDD-based system, we expect subsequent I/Os to be independent of each other, therefore scaling the inter-arrival times is a valid approximation of the workload's access pattern when run on a faster storage system. This assumption is justifiable for large-scale applications where most requests come from different users. Therefore, the intensity knob makes the application more compatible with a high service rate storage system, while retaining the previous spatial locality. Whether this locality and hence the model is subject to change in a faster system is deferred to future work.

IV. Characterization and Validation

A. Original DC Workloads

For all our experiments we use traces from production servers of ten popular large-scale DC applications. Messenger, Display Ads and User Content are the SQL portions of an Online Messenger, an Ads Display and a Live Storage application respectively. For each one, we study the part that maintains the SQL database with user information. These applications service thousands of users; therefore the data being accessed is typically spread across most of the provisioned disks.

Email, Search and Exchange are latency-critical online services. Email and Exchange are hosted in a much larger storage system than Search. Search has significant spatial locality, with some portions of the disk being frequently accessed and others heavily underutilized. TPCC, TPCE and TPCH are large-scale databases, part of the TPC benchmark suite [14]. Finally, D-Process is a highly-parallelized distributed computing application that resembles Map-Reduce [3], collecting and processing large amounts of information on applications such as Search. Its storage comprises of a large number of disks, partitioned between data and logs. These applications cover the majority of large-scale workloads in modern DCs.

B. Generating Models from Traces

The first step in order to create the workload models is collecting real, 24-hour long (unless otherwise specified) traces from production servers, hosting the applications previously discussed. The I/O traces are collected from individual servers, however due to load balancing in DCs the storage behavior is similar across multiple machines [10]. The length of the traces is sufficiently representative of an application's behavior, given the self-similarity of access patterns in DC workloads [10]. The traces are collected using ETW [5], which aside from information on I/O features (block size, type of request, etc.), tracks the file name, thread id and values of timestamps for each storage access. Having these traces, we create state-diagram models with different number of levels. The models are created by clustering I/Os in states based on their spatial locality and then categorizing them based on size, type (read/write) and randomness (random/sequential). Each distinct I/O category becomes a different transition and based on the number of I/Os that belong to each transition we calculate its inter-arrival time and probability. These models are then used to create the synthetic workloads.

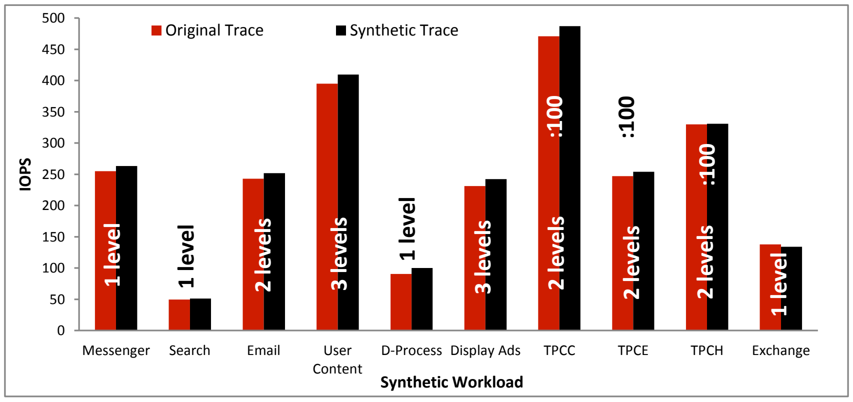


Fig. 4: Throughput Comparison between Original and Synthetic workloads for the 10 applications.

C. DC Application Characterization

We previously described a way to model the I/O characteristics of large-scale DC applications. Here we perform an in-depth per-thread characterization of the ten applications based on this model and provide insights on their behavior. To the best of our knowledge no such per-thread, storage activity categorization and characterization has been previously performed for DC applications. We separate the storage traces per-thread and define different thread types for each application based on activity fluctuation, intensity (I/O rate) and functionality of the thread (Data or Log - where applicable). The different thread types are shown in Table II. In Table III we show the per-thread storage characterization for one online service (Exchange), one SQL-based application (Messenger), and one of the TPC benchmarks (TPCE). We show the fluctuation for the aggregate workload and each thread type over the entire tracing period, as well as the I/O features in terms of average block size, inter-arrival time, read:write ratio, sequential I/Os and introduce here the study of ***spatial locality***. Spatial locality is estimated using one-level models, where each state corresponds to 25% of the machine's storage. The third to last column denotes the thread weight of the *individual thread* and of the *entire thread type*. We also show average performance metrics (throughput and latency). Examining *Exchange* reveals that the majority of storage activity comes from few, very I/O intensive threads (*Data #0*), while threads with no fluctuation, or low activity account for a considerably lower portion of the total throughput. *Exchange* is random, write I/O-dominated, while studying its spatial locality reveals that most accesses happen in the first half of the provisioned storage.
Email and Search and Exchange have similar behavior in terms of thread type classification and per-thread storage activity. Similarly, all the SQL applications (Messenger, Display Ads, User Content and D-Process) experience high resemblance in their storage activity. Finally, the I/O behavior of TPCC is very close to that of TPCE, with the DSS TPCH slightly deviating in terms of number of threads and intensity of storage activity.

D. Validation

Validating the accuracy of the model and the tool is necessary in order to ensure that original and synthetic workloads

Thread Type	Functionality	Intensity	Fluctuation
Data #0	Data	High	High
Data #1	Data	High	Low
Data #2	Data	Low	High
Data #3	Data	Low	Low
Log #4	Log	High	Low
Log #5	Log	Low	High

TABLE II: Per-Thread Classification for the ten examined applications based on *query type*, *intensity* of storage activity and *fluctuation* in the intensity of I/O requests.

are similar in their storage activity. Apart from that, we want to identify the optimal level of detail for each application. We perform the following steps:

1) Collect traces from production servers
2) Create workload models with a configurable number of levels
3) Run the synthetic workload and collect the new trace
4) Compare I/O characteristics and performance metrics between the original and synthetic storage workload.

For the validation experiments we use a server provisioned for SQL applications, like Messenger and User Content, with 8 cores, 5 physical volumes, 10 disk partitions and a total of 2.3TB of purely HDD storage.

We maintain the configuration of the storage system the synthetic trace is replayed on, as close as possible to that of the original system by performing specific I/O requests in the appropriate disk partitions. For the SQL-based workloads, for example, Log I/Os are replayed in the Log partition while SQL queries are replayed in the data partition. For the remaining applications, the system varies in the real DC (Search and D-Process run on striped four-disk SATA systems); however, throughput does not greatly deviate from its expected value. Although this result might seem unexpected, for an incorrectly provisioned system, these applications have relatively low I/O throughput (IOPS) that is easily satisfied through a system engineered for SQL. Although the percentile difference is higher for these two applications, the absolute number of IOPS remains reasonably low.

For each one of the ten applications we evaluate the similarities in the features of the I/O requests (block size, rd/wr, rnd/seq, inter-arrival time and thread weight) as well

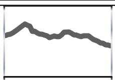
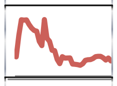
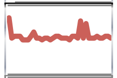
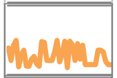
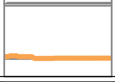
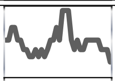
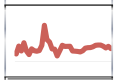
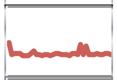
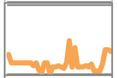
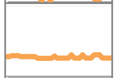
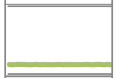
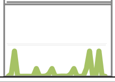
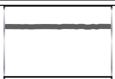
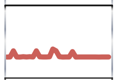
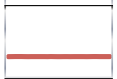
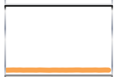
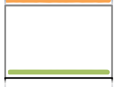
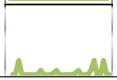
Workload	Thread Type	Fluctuation		Rd:Wr Ratio	%Seq. I/Os	Avg. Int. Time (ms)	Avg. Req. Size (KB)	Spatial Locality				Th W (one/type)	Avg IOPS	Avg Latency (ms)
								St1	St2	St3	St4			
Exchange (6h)	Total			**1:2.64**	**13.74**	**16.31**	**16.71**	**42.3**	**55.8**	**1.9**	**0.1**	**1.00/1.00**	**134.0**	**3.65**
			R	1	2.31	55.67	18.31	35.2	60.4	4.3	0.0	0.27	36.81	5.85
			W	2.64	22.67	28.65	12.41	45.3	54.5	0.1	0.1	0.73	97.19	0.48
	Data #0			**1:2**	**2.1**	**33.42**	**8.2**	**42.1**	**57.4**	**0.5**	**0.0**	**0.38/0.69**	**51.18**	**3.89**
			R	1	1.3	68.43	8.2	38.1	59.8	3.1	0.0	0.13	18.36	5.19
			W	2	2.8	44.27	8.2	48.9	50.1	1.0	0.0	0.26	34.12	0.48
	Data #1			**3.95:1**	**11.76**	**178.5**	**10.70**	**47.6**	**50.1**	**2.3**	**0.0**	**0.041/0.12**	**5.54**	**4.12**
			R	3.95	2.1	238.5	12.4	41.1	58.1	0.8	0.0	0.033	4.432	5.95
			W	1	17.85	498.7	4.00	31.9	48.7	19.4	0.0	0.008	1.108	0.46
	Data #2			**1:100**	**12.7**	**706.5**	**4.00**	**72.4**	**27.4**	**0.0**	**0.2**	**2E-3/0.13**	**0.32**	**0.50**
			R	0	-	-	-	-	-	-	-	0	0	-
			W	100	12.7	706.5	4.01	72.4	27.4	0.0	0.2	2E-3	0.32	0.49
	Data #3			**2.44:1**	**13.3**	**2836.6**	**48.2**	**21.0**	**68.8**	**10.2**	**0.0**	**2E-4/0.04**	**0.018**	**3.89**
			R	2.44	3.5	4676.3	65.5	43.0	58.4	2.6	0.1	14E-5	0.013	5.16
			W	1	13.7	3380.2	16.0	0.0	78.9	21.1	0.0	58E-6	5.2E-3	0.46
Messenger (6h)	Total			**2.8:1**	**9.32**	**4.63**	**9.17**	**31.5**	**22.8**	**45.5**	**0.2**	**1.00/1.00**	**255.14**	**8.09**
			R	2.8	7.32	6.89	8.46	31.0	38.6	30.8	0.2	0.737	194.02	10.01
			W	1	11.36	16.43	11.47	32.6	10.8	56.7	0.2	0.263	69.28	3.43
	Data #0			**1.8:1**	**11.36**	**53.9**	**4.85**	**31.6**	**32.4**	**36.0**	**0.0**	**0.052/0.42**	**13.8**	**9.36**
			R	1.8	9.43	81.22	8.10	51.5	10.1	38.4	0.0	0.643	8.87	11.84
			W	1	12.36	99.38	1.00	22.9	42.7	34.4	0.0	0.357	4.93	4.62
	Data #1			**3.35:1**	**5.36**	**112.34**	**8.96**	**65.8**	**21.9**	**12.3**	**0.0**	**0.021/0.27**	**5.53**	**9.83**
			R	3.35	4.26	134.56	5.36	71.8	19.6	10.6	0.0	0.016	4.25	10.83
			W	1	7.32	178.98	11.48	43.8	30.8	25.4	0.0	0.005	1.27	4.47
	Data #2			**1:2.1**	**13.67**	**456.32**	**14.56**	**43.5**	**36.6**	**19.0**	**1.9**	**0.004/0.13**	**1.00**	**8.48**
			R	1	8.42	731.68	10.01	41.2	23.8	35.6	0.5	0.0012	0.322	10.01
			W	2.1	13.65	531.11	18.86	46.3	41.8	8.8	3.1	0.0027	0.678	4.86
	Data #3			**10.5:1**	**10.80**	**483.12**	**8.98**	**43.5**	**38.9**	**17.6**	**0.0**	**0.004/0.18**	**1.00**	**8.77**
			R	10.5	8.78	493.15	7.43	48.9	31.5	19.6	0.0	0.0036	0.913	9.39
			W	1	14.56	1015.61	11.48	35.3	56.5	8.2	0.0	3.5E-4	0.087	3.89
	Log #4			**1:99**	**12.15**	**77.1**	**16.5**	**9.0**	**61.9**	**22.1**	**6.0**	**3E-4/5E-3**	**0.079**	**1.31**
			R	1	8.49	1014.6	4.00	79.0	21.0	0.0	0.0	3E-6	8E-5	10.83
			W	99	13.17	68.9	18.20	7.5	63.7	22.8	6.0	3E-5	0.078	0.46
	Log #5			**1:100**	**12.89**	**137.8**	**2.80**	**30.8**	**60.8**	**8.4**	**0.0**	**2E-5/4E-4**	**0.005**	**0.51**
			R	0	-	-	-	-	-	-	-	0	0	-
			W	100	12.89	137.8	2.80	30.8	60.8	8.4	0.0	2E-5	0.005	0.51
TPCE (11min)	Total			**10.5:1**	**8.15**	**0.48**	**8.41**	**89.8**	**8.4**	**1.7**	**0.3**	**1.00/1.00**	**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	26.9	15	2.1	0.087	2,148	0.69
	Data #0			**12.5**	**16.3**	**2.18**	**8.02**	**91.0**	**8.0**	**0.2**	**0.8**	**0.048/0.43**	**1,172**	**5.67**
			R	12.5	11.65	2.37	8.02	90.8	8	0.4	0.7	0.044	1,086	6.13
			W	1	50	12.68	8.01	92	0	0	8	0.004	86	0.89
	Data #1			**7:1**	**20.6**	**3.25**	**6.1**	**92.4**	**7.4**	**0.2**	**0**	**0.077/0.30**	**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
	Data #3			**70.4:1**	**15.5**	**12.32**	**8.00**	**91.2**	**6.2**	**2.4**	**0.2**	**0.035/0.25**	**859.3**	**5.35**
			R	70.4	15.6	12.46	8.00	90.6	7.2	2.4	0.1	0.034	847.3	5.41
			W	1	7.1	287.3	4.01	99	0	0	1	0.0005	12.0	0.65
	Log #4			**1:99**	**11.75**	**9.09**	**2.05**	**88.8**	**10**	**1.0**	**0.3**	**5E-4/3E-3**	**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
	Log #5			**1:100**	**1.42**	**77.1**	**2.10**	**89.0**	**11.0**	**0**	**0**	**2E-5/2E-4**	**0.54**	**0.56**
			R	0	-	-	-	-	-	-	-	0	0	-
			W	100	1.42	77.1	2.10	89.0	11.0	0	0	2.2E-5	0.54	0.56

TABLE III: Per-Thread Characterization for one Online Service (Exchange), one SQL-based application (Messenger), and one of the TPC benchmarks (TPCE). From left to right we show the fluctuation of storage activity, the I/O features in terms of rd:wr ratio, percentage of Seq. I/Os, average block size and inter-arrival time and spatial locality, as well as average performance metrics, in terms of throughput and latency for each thread type.

	Metrics	Original Workload	Synthetic Workload	Deviation
Messenger	Read:Write Ratio	2.8:1	2.8:1	0%
	% of Random I/Os	90.68%	89.43%	-1.38%
	Block Size Distribution	8K(87%) 64K(7.4%) 1K(1.6%)	8K(88%) 64K(7.8%) 1K(1.7%)	0.1%-1%
	Thread Weights Distribution	T1 (19%) T2 (11.6%) T3 (1.6%)	T1 (19%) T2 (11.68%) T3 (1.6%)	0% - 0.05%
	Average Inter-Arrival Time	4.63ms	4.78ms	1.02%
	Throughput (IOPS)	255.14	263.27	3.1%
	Average Latency	8.09ms	8.48ms	4.8%

TABLE IV: I/O Features - Performance Metrics Validation for Messenger.

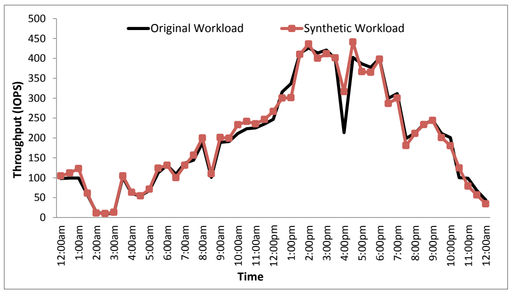


Fig. 5: Validation of Storage Activity Fluctuation over 24h (Messenger).

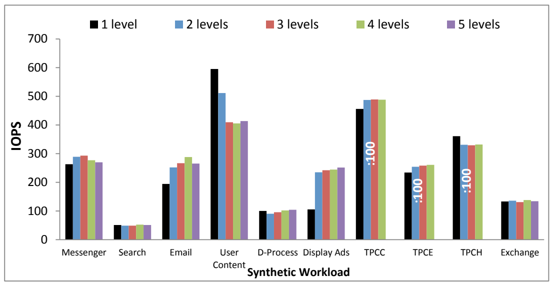


Fig. 6: Throughput for increasing number of levels.

as the performance metrics (throughput and latency) of the synthetic applications as opposed to the original ones. As far as the proportion of accesses is concerned, we verify that thread weights are satisfied with **less than 0.05%** deviation from their original values. Table IV shows the comparison for these metrics between original and synthetic workload for Messenger. The results are similar for the remaining applications. In all cases the deviation for the I/O features between original and synthetic load is less than 5%. Similarly for the performance metrics the deviation is at most 6.7% and on average 3.38%. Figure 4 shows the throughput comparison between original and synthetic load for all applications. The difference in IOPS is always **less than 5%**, verifying the accuracy of the modeling and generation process. Furthermore, in order to ensure the consistency of our results, we calculate the variance between different runs of the same synthetic workload and guarantee a difference in throughput **less than 1%** in all cases.

Figure 4 also plots the optimal number of levels per application, which is the one for which the synthetic trace resembles the original workload best. As optimal granularity we define the first number of levels for which the performance metrics stabilize (less than 2% difference in IOPS). That way, we convey the best possible accuracy with the least necessary model complexity. This methodology allows for a configurable level of detail in the model of each application. Figure 6 shows how the throughput changes for each application for an increasing number of levels. In most cases one to three levels are sufficient for the I/O characteristics to stabilize. Finally, we verify the resemblance in the fluctuation of storage activity between original and synthetic workloads. We perform a sensitivity study on the granularity at which the I/O request intensity changes, to choose the interval over which we calculate inter-arrival times. We observe that within a 30-minute period there is no significant fluctuation in the storage activity for the examined applications. Figure 5 shows the resemblance in storage activity fluctuation between the original and synthetic application for Messenger. The results are similar for the other applications as well. In all cases, peaks and troughs coincide for the two workloads, verifying that the activity fluctuation requirements are met.

E. Comparison with IOMeter

IOMeter is the most well-known open-source workload generator [6]. Although it offers many capabilities as far as access characteristics are concerned, it has limited information on the spatial locality of I/Os, making it unsuitable for several DC storage studies. Furthermore, IOMeter implements outstanding I/Os but cannot represent inter-arrival times, which seriously limits its intensity scaling capabilities. Finally, it does not allow specific file accesses, which as will be seen in Section VI.B, would make it impractical to evaluate the benefits of defragmentation. Table V summarizes the differences between the features supported by the two tools.

In this Section we compare the performance characteristics of IOMeter and DiskSpd. For the purpose of this comparison no change is conducted in IOMeter, and the parameters for the tests are defined using the tool's default knobs. We perform identical tests using both tools and quantify the difference in throughput and latency. The table below (Table VI) shows how the tools behave in a series of simple access patterns with the exact same parameters. All tests are run for 30 seconds, performing I/O requests to a simple file. In the interest of clarity, we do not demonstrate all possible parameter configurations, but some representative examples. Note that no notion of spatial locality is introduced in these simple tests. From the results we observe that both tools behave similarly with a maximum throughput deviation of 3.4%.

The main difference in the two tools becomes evident when introducing the notion of spatial locality. To demonstrate that DiskSpd takes locality into account while IOMeter does not, we use an optimization technique that will be presented in more detail in the following section (Section VI.A). SSD caching takes advantage of frequently-accessed blocks, and thus improves performance by avoiding accessing the disk often. If a tool takes into consideration spatial and temporal locality we expect an improvement in performance when the synthetic trace is run using SSD caches. We run the synthetic traces for the ten applications and the corresponding I/O tests that best resemble their behavior using IOMeter. No notion of spatial locality is incorporated in the latter. Figure 7 shows how performance changes as we progressively add SSDs to the system for each of the workloads. The important point in these figures is not the precise speedup but the significantly different behavior of the tools. In all cases it becomes evident that IOMeter does not reflect the spatial and temporal locality of the original access pattern. For most applications there is no speedup for an increasing number of SSDs, due to incorrect caching of blocks, and for those that a speedup exists it is inconsistent with what would have been expected as caching becomes more intense (more SSDs - better speedup).

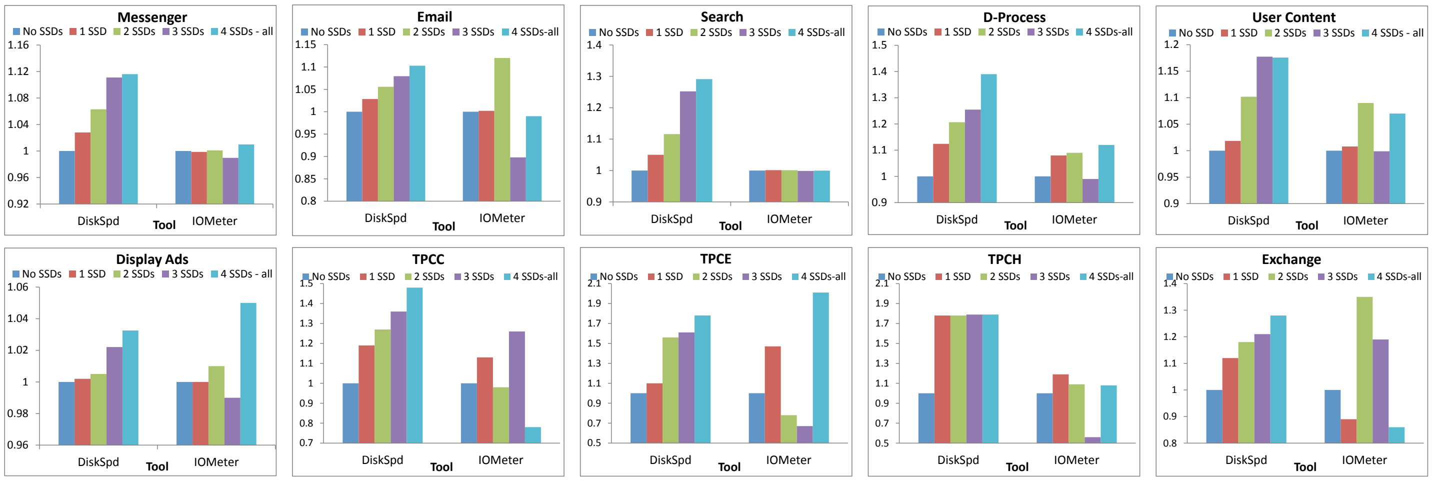


Fig. 7: IOMeter vs DiskSpd Speedup Comparison for (a) Messenger, (b) Email, (c) Search, (d) D-Process, (e) User Content (f) Display Ads (g) TPCC, (h) TPCE, (i) TPCH and (j) Exchange. The results for DiskSpd confirm the expected impact of SSD caching on workload performance. On the other hand, when the workloads are run using IOMeter there is either no performance speedup (e.g. Messenger) or inconsistent speedup (e.g. User Content) with an increasing number of SSD caches.

Features	IOMeter	DiskSpd
Inter-Arrival Times (Static or Distributions)	No	Yes
Intensity Knob	No	Yes
Spatial Locality	No	Yes
Temporal Locality	No	Yes
Trace Replay	No	Yes
Granular I/O Load Detail	No	Yes
Individual File Accesses	No	Yes

TABLE V: IOMeter vs DiskSpd Features Comparison

Test Configuration	IOMeter (IOPS)	DiskSpd (IOPS)
4K Int. Time 10ms Rd Seq	97.99	101.33
16K Int. Time 1ms Rd Seq	949.34	933.69
64K Int. Time 10ms Wr Seq	96.59	95.41
64K Int. Time 10ms Rd Rnd	86.99	84.32

TABLE VI: IOMeter vs DiskSpd Comparison

V. Use Cases

One of the main benefits from using a modeling and generation tool for DC workloads is enabling storage studies, which would otherwise require access to application code or full application deployment. In this section we evaluate two possible use cases for the tool: SSD caching and defragmentation. Both are spatial and temporal locality-dependent, and have been unexplored using workload generation tools.

A. SSD caching

Designing an efficient storage system configuration for widely-deployed applications is a great challenge and in terms of proper provisioning, a field that separates high-quality systems from the norm. Studying the spatial locality of the ten DC applications reveals that for most of them, I/Os are aggregated in a small LBN range. This motivates incorporating SSDs to improve performance. Estimating performance, however, is not easy since writes in SSDs are highly unpredictable, and often as slow as disk, which makes performance gains greatly dependent on the I/O access features. This impels using modeling and characterization to evaluate the potential benefit.

Workload	Spatial Locality - Level 1			
	State1	*State2*	*State3*	*State4*
TPCH	92.7%	6.2%	1.3%	0.0%
TPCE	89.8%	8.4%	1.7%	0.3%
D-Process	73.3%	18.8%	0.0%	0.0%

TABLE VII: Spatial Locality (SSD-caching study). Studying the spatial locality for the three applications with the highest benefit from SSD caching reveals that they have the highest I/O aggregation. This justifies the significant speedups from introducing SSD caching in the system.

Due to the fact that we use an open-loop approach, applications are not retuned when switching to the SSD-based system. The experiments are performed by running the previous models on an SQL-provisioned server with 4 SSD caches (8GB each) [1], which we progressively turn on.

Figure 7 shows the storage speedup when going from no SSD caches on (left bar) to all 4 SSD caches on (right bar). We observe that especially for the I/O intensive TPCH, TPCE and D-Process, the performance benefit from using a large number of SSDs is significant (31% on average across all workloads for 4 SSDs and 79% maximum for TPCH). Studying the clustering of accesses in the corresponding models reveals that these three applications have the highest aggregation of I/O requests (Table VII). Increasing the number of levels in this case, confines the accessed LBNs in a smaller range, thus better caching frequently-accessed blocks. An important note is that in Figure 7, we refer to storage speedup and not speedup for the entire application. These workloads are not necessarily limited by storage but their performance improvement, and the expected improvement in efficiency, are strong incentives towards the use of SSD caching nonetheless.

B. Defragmentation

Most DC applications experience high levels of fragmentation, as user-requests get accumulated over time, with Random I/Os often exceeding 80%. This motivates the use of defragmentation to improve performance and efficiency. From the information provided by ETW [5] we can extract the name of the file for each I/O access, estimate fragmentation levels,

Workload	Read	Write	Before		After	
			Rnd	Seq	Rnd	Seq
Messenger	73.7%	26.3%	90.7%	9.3%	63.2%	35.7%
Email	52.8%	45.2%	84.5%	13.7%	61.6%	33.7%
Search	49.8%	45.1%	87.7%	8.5%	70.9%	24.5%
UserContent	58.3%	39.4%	93.1%	5.5%	73.2%	25.0%
D-Process	30.1%	68.8%	73.2%	26.8%	45.4%	54.4%
DisplayAds	96.5%	2.5%	93.5%	4.3%	78.5%	19.2%
TPCC	68.8%	31.2%	97.2%	2.8%	71.1%	29.9%
TPCE	91.3%	8.7%	91.9%	8.2%	77.7%	22.4%
TPCH	96.7%	3.3%	65.5%	35.5%	52.8%	47.2%
Exchange	32.0%	68.1%	83.2%	16.8%	68.1%	31.9%

TABLE VIII: Rnd/Seq Characteristics before and after defragmentation

and perform a block rearrangement to improve the sequential characteristics as shown in Figure 8. Estimating performance after defragmentation, using the models, can act as an offline method to identify the benefits of defragmentation, as well as the optimal moment to perform defragmentation, without having to examine the entire application. These are usually latency-critical applications that cannot afford the overhead of a continuous online evaluation.

In most cases, the percentage of sequential accesses increases by 20% (Table VIII), which corresponds to a storage speedup of 8-45% as shown in Figure 9. This result implies that clustering I/Os is more beneficial than taking advantage of parallel spindles, due to faster completion of sequential accesses. Two applications that benefit from this are D-Process and Email, which have the highest write-to-read probabilities. Since these applications are random-write dominated, improving their sequential characteristics allows better utilization of the full disk rotation. Defragmentation also benefits the TPC benchmarks that access continuous entries in database tables.

A comparison between the benefits of SSD caching and defragmentation shows that for some workloads (Email and Display Ads) defragmentation offers better speedups without increasing the cost of the system. The decision on which method is most beneficial at a certain time is up to the storage system designer, given the application and system of interest.

VI. Future Work and Conclusions

In this work we have proposed a framework for modeling, characterization and regeneration of large-scale storage workloads. We have extended a probabilistic model to capture granular information on a workload's I/O pattern and implemented a tool that recreates DC workloads with high fidelity. This tool can be used for a wide spectrum of studies for large-scale systems, without the need to access DC application code or full application deployment.

In contrast with previous work, we take into account spatial and temporal locality of I/O accesses, a critical feature for DC applications. We have conducted detailed characterization of the storage activity of DC applications and performed extensive validation of the generated I/O traces against ten real workloads. Finally, we have evaluated two possible uses for the tool, SSD caching and defragmentation, and quantified the improvement in performance. We believe that, compared to previously available workload generators, this framework can be used to make confident design decisions for DC systems.

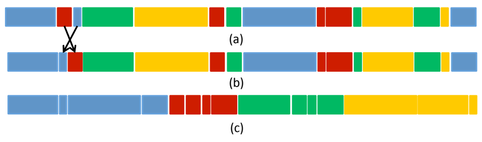


Fig. 8: File Mapping before (a), during (b) and after (c) defragmentation. Blocks of different colors correspond to different files.

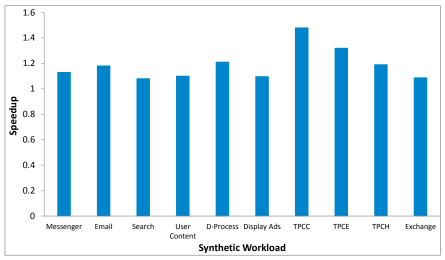


Fig. 9: Storage Speedup from Defragmentation

The main focus of our future work is the development of a model that aggregates all different parts of the system and a tool that recreates a synthetic load for complete DC applications, with possible applications in virtualization, application consolidation and DC performance and power modeling.

Acknowledgements

We sincerely thank Daniel Sanchez, Richard Yoo, and the anonymous reviewers for their useful feedback on earlier versions of this manuscript. Christina Delimitrou was supported by a Stanford Graduate Fellowship.

References

[1] Adaptec MaxIQ. 32GB SSD Cache Performance Kit. http://www.adaptec.com/en-US/products/CloudComputing/MaxIQ/SSD-Cache-Performance/
[2] I. Ahmad. "Easy and Efficient Disk I/O Workload Characterization in VMware ESX Server". In Proc. of IEEE IISWC, Boston, MA, 2007.
[3] J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". In Proc. of OSDI'04, SF, CA, December, 2004.
[4] 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/
[5] ETW: Event Tracing for Windows: http://msdn.microsoft.com/en-us/library/bb968803%28VS.85%29.aspx
[6] IOMeter, performance analysis tool. http://www.iometer.org/.
[7] S. Kavalanekar, D. Narayanan, S. Sankar, E. Thereska, K. Vaid, B. Worthington,"Measuring Database Performance in Online Services: a Trace-Based Approach". In Proc. of TPC TC, Lyon, France, 2009.
[8] S. Kavalanekar, B. Worthington, Q. Zha, V. Sharda "Characterization of storage workload traces from production Windows servers". In Proc. of IEEE IISWC, Seattle, WA, Sept. 2008.
[9] C. Kozyrakis, A. Kansal, S. Sankar, K. Vaid, "Server Engineering Insights for Large-Scale Online Services". In IEEE Micro, vol.30, no.4, July 2010.
[10] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron, "Migrating enterprise storage to SSDs: analysis of tradeoffs". In Proc. of EuroSys, Nuremberg, 2009.
[11] S. Sankar, K. Vaid, "Storage characterization for unstructured data in online services applications". In Proc. IEEE IISWC, Austin, TX, 2009.
[12] S. Sankar, K. Vaid. "Addressing the stranded power problem in datacenters using storage workload characterization". In Proc. of the first WOSP/SIPEW, San Jose, CA, 2010.
[13] SQLIO Disk Subsystem Benchmark Tool. http://www.microsoft.com/downloads/en/details.aspx?familyid=9a8b005b-84e4-4f24-8d65-cb53442d9e19&displaylang=en
[14] TPC Benchmarks. http://www.tpc.org/information/benchmarks.asp
[15] H. Vandenbergh. Vdbench: User Guide 5.00 Oct. 2008.