Accurate Modeling and Generation of Storage I/O for Datacenter Workloads

Christina Delimitrou[1], Sriram Sankar[2], Kushagra Vaid[2], Christos Kozyrakis[1]
[1]Stanford University, [2]Microsoft

Abstract

Tools that confidently recreate I/O workloads have become a critical requirement in designing efficient storage systems for datacenters (DCs), since potential inefficiencies get aggregated over several thousand servers. Designing performance, power and cost optimized systems requires a deep understanding of target workloads, and mechanisms to effectively model different 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. Despite these issues, current workload generators are not comprehensive enough 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 behaviors.

To address these limitations, 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 design of the tool and the validation process performed against six original DC applications traces. We explore the practical applications of this methodology in two important storage challenges - 1) SSD caching and 2) defragmentation benefits on enterprise storage. In both cases we observe significant storage speedup for most of the DC applications. Since knowledge of the workload's spatial locality is necessary to model these use cases, our tool was instrumental in quantifying their performance benefits.

1 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, which account for a significant portion of the total cost of ownership of a datacenter (DC) [4, 14]. Specifically, for online services, data retrieval is often the bottleneck to application performance [1, 4], making efficient storage 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. It is hence imperative to invest in framework that allows for extensive workload analysis.

Large scale Online Services differ from conventional applications in that they cannot be approximated by single machine benchmarking. Furthermore, code, user behavior patterns and datasets of DC applications are 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, the next step is to create a tool that convincingly reproduces the application's storage behavior via a synthetic I/O access pattern.

Previous efforts on storage I/O workload generation lack the ability to detect and utilize the spatial and temporal locality of I/O access patterns, causing them to significantly deviate from the application's indigenous characteristics.

In this work, we provide a toolset for research on large scale storage systems. This toolset includes probabilistic, state diagram-based models that capture information of *configurable granularity* on the workload's access patterns. The models are developed from production traces of real DC applications based on previous work by [1]. We design a tool that recognizes these models and recreates synthetic access patterns with I/O characteristics that closely match those of the original application.

As part of this tool the following features have been implemented:

1. The ability to issue I/O requests with specified inter-arrival times, both static and following time distributions. Unlike all previous work, our model is based on inter-arrival times instead of outstanding I/Os making it more representative of an application's behavior [3].
2. The ability to generate requests from multiple threads with individual access characteristics (block size, rd/wr, seq/rnd), while preserving the thread weight (i.e. portion of accesses) for each type of I/O.
3. The ability to modify the intensity of the generated I/Os through an intensity knob which scales the inter-arrival time of I/O requests. This is especially useful in high performance storage systems (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 some DC applications, SSD caching offers a significant storage system speedup without application change (18% 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 [10], we rearrange blocks on disk in order to improve the sequential characteristics of the workloads. Using the tool shows that defragmentation offers a significant boost in performance (14% on average), in some cases 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 as in the original application. No previous storage tool (eg: IOMeter) can simulate spatial and temporal locality of DC workloads.
- This methodology (model and tool) 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 DC challenges have been relatively unexplored due to lack of a tool that allowed their evaluation.

The combination of the model, tool and use cases adheres to the optimal study technique for DC workloads and is distinguished from related previous efforts in its ability to correctly replay large scale applications.

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

2 Related Work

Exploring different storage system configurations is of great interest to hardware architects, especially when the target system is a large scale DC. Significant prior work [7] has gone into provisioning this part of the system. However, a necessary requirement towards efficiently configuring the storage system is studying DC workloads. An ideal way for that should involve a model that representatively captures the workload's features and a tool that accurately recreates its access pattern.

Despite this, most prior large scale storage configuration techniques are mainly empirical, based on the workload's characteristics as derived from traces [8]. Kavalanekar et al [2, 8] use a trace-based approach to characterize large online services for storage system configuration and performance characterization 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. Regenerating I/O workloads with high fidelity can offer far richer information towards understanding the behavior of workloads that remain largely unknown. It also enables addressing instrumental challenges in storage system design (e.g. SSD incorporation/migration of hot data) when optimizing for performance and efficiency.

IOMeter [6], SQLIO [12], Vdbench [11] are all open source generators of disk I/O loads. IOMeter allows for specific I/O characteristics to be defined, SQLIO simulates some aspects of the disk load of the Microsoft SQL Server, while Vdbench apart from the feature of I/O generation in disks and files 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 [13] captures information on disk I/O without significant CPU or latency overheads. However, as with IOMeter, all these workload generators lack the ability to exploit the temporal and especially spatial locality of DC applications. Also, 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 true behavior [3], decoupled from the system that hosts it. For our work, we extend the functionality of DiskSpd [5], an I/O workload generator in ways that permit us to recreate representative DC traces.

3 Modeling and Generation Tool

The purpose of this section is to describe the steps for the modeling and generation process. We present the state diagram-based model used, the extensions introduced to it in this work and the implementation key-points of the tool that creates synthetic I/O loads based on the model.

3.1 Basic State Diagram Model

For our model we use the Markov Chain representation proposed by Sankar et al. [1]. According to the model, states correspond to ranges of logical blocks on disk (LBN) 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: block size, randomness, type of I/O (read, write) and inter-arrival time between subsequent requests. The intent of the model is to provide a comprehensive representation where each transition has an LBN range and an inter-arrival time that reflect the spatial and temporal locality of the I/O accesses respectively. The probability for each transition is calculated as the percentage of I/Os that correspond to it. Figure 1(a) demonstrates a simplified form of the state diagram with four states, each of which corresponds to 25% of the total LBNs.

3.2 Hierarchical State Diagram Model

Different applications have different access patterns, some requiring more detail than others to be accurately captured. In order to convey information of finer granularity on the I/O access pattern we have extended the previous model to a hierarchical representation. Figure 1(b) demonstrates one such model with two levels. The hierarchical model is constructed as follows: each state in the one level diagram is subdivided in four states and becomes a new state diagram. The two-level state diagram will have 16 states.

Perhaps counter-intuitively the number of transitions in the new diagram is not 256 but 76. As shown in Figure 1(b) level-two (fine grained) transitions only exist within the large states but not across them. Between level-one states we maintain the previously demonstrated transitions (Figure 1(a)). This means that a hierarchical structure is preferred over a flat representation where all transitions are explored. The reason for that is that we expect spatial locality to be confined within states rather than across them. This does not cancel the value of a flat model, but rather proposes that a hierarchical model is just as beneficial as a flat model without making the number of transitions intractable. Comparing the throughput of models constructed with the hier-

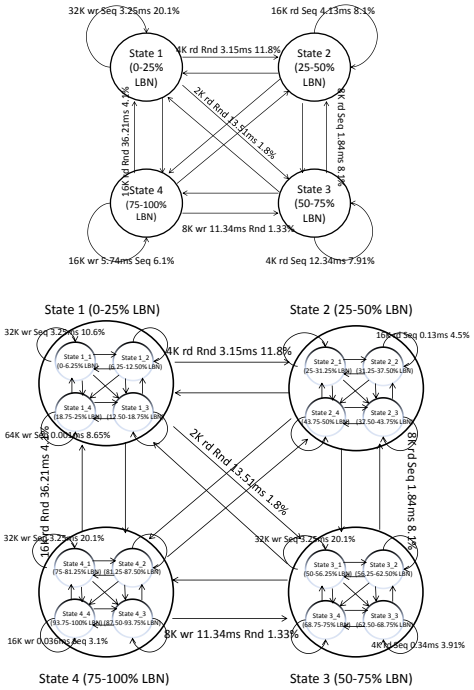

Figure 1: (a) One level and (b) Two levels State Diagram

archical and the flat representations shows less than 5% difference in throughput.

The numbers of levels reflects the complexity of an application's locality pattern and as will be shown in the validation section (4.3), finer granularity is instrumental for some applications to be accurately represented.

3.3 Generation Tool Design - DiskSpd

The model, previously discussed, consists of 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 [5]. It works as a command line tool, performing 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, system warmup and cool-down, number of outstanding I/Os, etc.

In order to recreate a representative workload using the model previously discussed, we have implemented a series of features in DiskSpd. The following subsections describe the major changes performed in the tool.

3.3.1 Inter-arrival Times (Static 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 certainly do not dominate an application's lifetime. Subsequent I/Os tend to have well defined time margins between them. Narayanan et al [3] 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 difference in inter-arrival times among DC workloads [1, 4].

To demonstrate these time margins between accesses of specific block ranges, we implement the notion of inter-arrival times in DiskSpd. Inter-arrival times 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 that a defined number of on-the-fly I/Os exists in the queues. The use of inter-arrival times instead of outstanding I/Os in a workload generator is first proposed here. Most previous tools are based on defining the number of I/Os in the queues of the system. However, queued I/Os do not characterize an application as well as inter-arrival times. The difference between the two becomes more clear in the case where we want to create a more intense workload as described in Section 3.3.3.

Furthermore, in order to capture the variations in the intensity of an application throughout its execution we have added the feature of inter-arrival time distributions, i.e. during the workload's execution the 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.

3.3.2 Multiple Threads and Thread Weights

By default DiskSpd supports the execution of either one or multiple threads but all characterized by the same access parameters. However, access patterns of real applications have distinct per transition characteristics.

In order to recreate an I/O load using the model, we have added the feature of executing threads with different I/O characteristics (block size, randomness, type (rd/wr), LBN range, inter-arrival time), each representing a transition in the state diagram. We have introduced the notion of *thread weight*, i.e. the proportion of I/Os 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 the "idle" time for each thread. Furthermore, in order to guarantee that the thread weights are satisfied throughout the workload's execution we perform a Round Robin visit in states so that all threads are active in different periods during the program's execution instead of limiting them in an arbitrary period of activity. That way the synthetic trace becomes a compressed version of the original workload. Although this does not cover all possible transition patterns, self-similarity tests [2] in original DC applications have verified that indeed the spatial characteristics of I/Os are consistent across time.

3.3.3 Intensity Knob

One of the main incentives behind developing this tool is evaluating 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 all threads' inter-arrival times down or up to increase or decrease their intensity respectively. This feature clarifies the distinction be-

tween 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. Just maintaining outstanding queue length in this case, stresses the I/O system out of proportion and is not useful for DC storage scaling studies. Having this knob offers the opportunity to evaluate the applicability of a storage system configuration based on the workload's expected intensity margins, at a spatial locality granularity.

For example, in a Hard Disk-based system when intensity exceeds the system's queues' capabilities, throughput levels off. 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 Hard Drive-based system.

An important note to make here is that our work is based on an open-loop approach, which means that the application is not retuned when we switch to an SSD-based system. Therefore, the intensity knob makes the application more compatible with a high service rate storage system, while retaining the previous spatial locality information and assuming most requests are independent of each other. Whether this locality is subject to change in a faster system and thus should also change is deferred to future work.

4 Toolset Validation

4.1 Original DC Workloads

For all our experiments we use traces from production servers of six 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 and Search are latency critical online services. Email is hosted in a much larger storage system than Search. Unlike the three SQL-based applications, Search demonstrates significantly higher spatial locality, with some portions of the disk being frequently accessed and others heavily underutilized. Finally, D-Process is a distributed computing application that resembles Map-Reduce [14], 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.

4.2 Generating Models from Traces

The first step in order to create the workload models, is collecting real, 24-hour long traces from production servers, hosting the applications previously discussed. The length of the traces is sufficiently representative of an application's behavior, given the self-similarities of access patterns in DC workloads [3]. The traces are collected using ETW [10], which aside from information on I/O features (block size, type of request, etc.), offers tracking of the file name, thread id and timestamps' values for each storage access. Having these traces, we create state diagram models with different number of levels. These models are then provided as input for the tool to create the synthetic workloads.

Metrics	Original Load	Synthetic Load	Deviation
Rd:Wr Ratio	1.8:1	1.8:1	0%
Random %	83.67%	82.51%	-1.38%
Block Sizes	8K(87%) 64K(7.4%) 1K(1.6%)	8K(88%) 64K(7.8%) 1K(1.7%)	0.33%
Th. Weights	T1(19%) T2(11.6%)	T1(19%)T2(11.68%)	0% - 0.05%
Avg Int.Time	4.63ms	4.78ms	3.1%
IOPS	255.14	263.27	3.1%
Avg Latency	8.09ms	8.48ms	4.8%

Table 1: I/O - Performance Metrics Validation (Messenger)

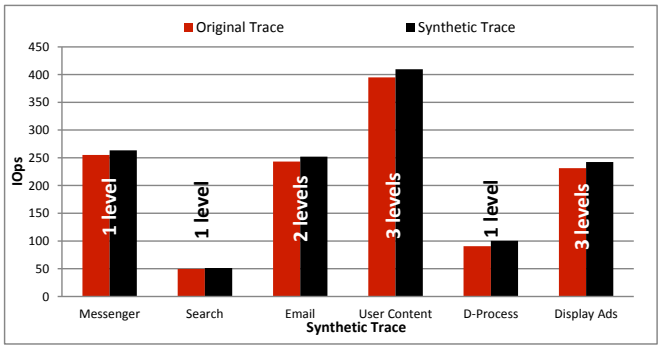

Figure 2: Throughput Comparison between Original and Synthetic Trace

4.3 Validation

Validating the accuracy of the model and the tool is necessary in order to ensure that original and synthetic workloads are similar in their storage activity. Furthermore, since we adopt an open-arrival approach (we do not guarantee a specific number of outstanding I/Os as in a closed system) I/O fidelity is not trivial. The process we are performing is as follows:

1. Collect traces from production servers as described in Section 4.2
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 this part of our experiments we use an SQL-provisioned server with 8 cores, 5 physical volumes, 10 disk partitions and a total of 2.3TB of purely HDD storage.

We try to maintain the configuration of the storage system the synthetic trace is replayed on as close as possible to the one the original trace has been collected from by replaying specific types of 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 three applications, the system varies in the real DC (Search and D-Process run on striped four-disk SATA systems); however, the throughput does not greatly deviate from its expected value. Although this result might seem unexpected at first, 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 six 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 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 the thread weights are satisfied with **less than 0.05%** deviation from their original values. Table 1 shows the comparison for these metrics between original and synthetic workload. For the sake of clarity, we only demonstrate this data for Messenger. The results are similar for the other applications. For all metrics the deviation between original and synthetic load is less than 4.8%. Figure 2 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.

Note that a number of levels is mentioned for each application in Figure 2. This is the optimal number of levels per application and is the one for which the synthetic trace behaves in a way that resembles the original trace more closely. The decision on the optimal granularity is performed by choosing 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.

4.4 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 currently has limited information on the spatial locality of I/Os, thus making it unsuitable for several DC storage studies. Furthermore, IOMeter implements the concept of 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 5.2, would make it impractical to evaluate the benefits of defragmentation.

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 above (Table 2) 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 deviation of 3.4% in throughput. The main difference in the two tools becomes evident when introducing the notion of spatial locality. In order to demonstrate how DiskSpd takes into consideration spatial locality while IOMeter does not, we use an optimization technique that will be presented in more detail in the following section (Section 5.1). SSD caching takes advantage of frequently accessed blocks and thus improves performance by avoiding visiting the disk often. If a tool takes into consideration spatial and temporal locality we expect to see an improvement in performance when the synthetic trace is run using SSD caches. We run the synthetic traces for the six 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 3 shows how the performance changes as we progressively add SSDs to the system for Messenger (3.a) and User Content (3.b). 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 locality of the original access pattern. For most applications there is no speedup for 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).

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 2: IOMeter vs DiskSpd Comparison

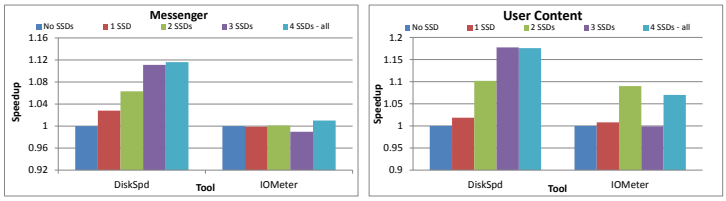

Figure 3: IOMeter vs DiskSpd Comparison for (a) Messenger, (b) User Content.

5 Use Cases

One of the main benefits from using a representative model and a corresponding robust tool to create DC workloads, is the opportunities it offers in evaluating storage studies that would otherwise require access to the application code or full application deployment. Especially when targeting large scale systems, such practicality can be proven critical in improving the Quality of Service (QoS) and TCO of the system. In this section we evaluate two such possible uses for the tool: SSD caching and the Benefits of Defragmentation. Both these studies are spatial locality-dependent and have thus been relatively unexplored using workload generation tools. Given the information on spatial locality distilled in the state diagrams we now have a tool that can evaluate storage system optimizations, in order to improve the performance and efficiency of the system.

5.1 SSD caching

Defining an efficient storage system configuration for widely deployed applications is a great challenge and in terms of proper provisioning, a field that often separates high quality systems from the norm. It is clear from studying the spatial locality of the six DC applications that for most of them, I/O accesses are aggregated in a small LBA range. This creates incentive towards incorporating SSD caches to take advantage of the frequently accessed blocks.

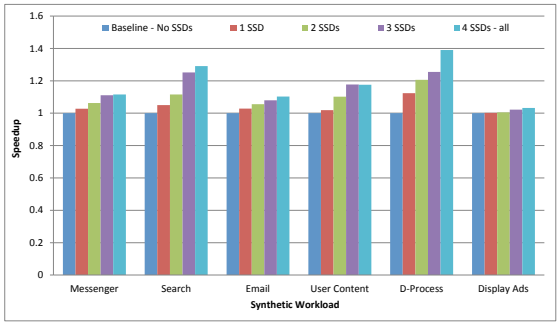

Figure 4: Storage Speedup from SSD caching

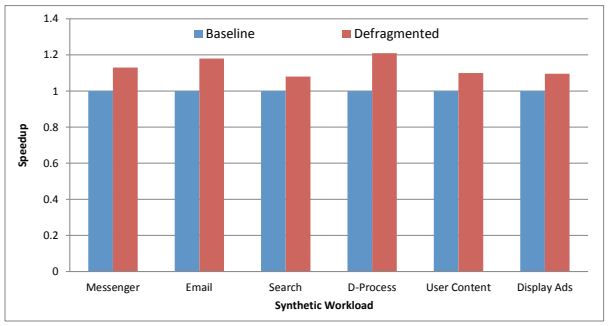

Figure 5: Storage Speedup from Defragmentation

As previously noted, we use an open-loop approach, which means that the applications are not retuned when switching to the SSD-based system. This might underestimate the benefits of the faster storage system but offers more concrete performance metrics. The experiments are performed by running the previously created models on an SQL provisioned server with 4 SSD caches (8GB each) [9], which we progressively turn on.

Figure 4 shows the storage speedup achieved when going from no SSD caches on (left bar) to all 4 SSD caches on (right bar). We observe that especially for Search, User Content and D-Process, the performance benefit from using a large number of SSDs is significant (18% on average for 4 SSDs and 38% maximum for D-Process). Studying the clustering of accesses in the corresponding models reveals that these three applications demonstrate the highest aggregation of I/O requests. An important note is that in Figure 4, 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.

5.2 Benefits from Defragmentation

Most DC applications experience high levels of fragmentation, with their Random proportion often exceeding 80%. This motivates the use of defragmentation to improve performance and efficiency.

From the information provided by ETW [10] we can extract the name of the file for each I/O access, estimate fragmentation levels, and perform a block rearrangement to improve the sequential characteristics. Evaluating the new workload's performance using the state 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 critical applications that cannot afford the overhead of a continuous online evaluation.

In most cases, the sequential characteristics improve on average by 20%, which corresponds to a storage speedup of 8-20% as can be seen in Figure 5. This result implies that clustering I/Os together is more beneficial for performance than taking advantage of parallel spindles, due to faster completion of sequential accesses. The applications that benefit from this process the most 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 them to better utilize the full rotation of the disk.

6 Future Work and Conclusions

In this work we have extended a probabilistic model that captures granular information on a workload's I/O patterns and implemented a tool that recreates DC workloads with high fidelity. We differentiate from previous work in that we take into account spatial and temporal locality of I/O accesses. We have performed extensive validation of the synthetic I/O traces against six real DC applications and verified the consistency of our results. Finally, we have evaluated two possible uses for the tool, SSD caching and defragmentation, and quantified the improvement in storage performance. We believe that, compared to previously available workload generators, this toolset can be used to make confident design decisions for DC scale systems.

As part of future work, we plan to address the limitations of this methodology, one of which is substituting single transitions with paths in the state diagram to preserve history of accesses. We are planning to evaluate the energy efficiency of the two uses presented here and quantify the full-system benefit. Finally, having a model that aggregates all different parts of the system and a tool that recreates a synthetic load for complete DC workloads is the main focus of our future work, with possible applications in virtualization, application consolidation and energy efficient DC designs.

References

[1] S. Sankar, K. Vaid, "Storage characterization for unstructured data in online services applications". IEEE International Symposium on Workload Characterization (IISWC), 2009.

[2] S. Kavalanekar, B. Worthington, Q. Zha, V. Sharda "Characterization of storage workload traces from production Windows servers". In Proc. IISWC 2008, Seattle, WA, Sept. 2008.

[3] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron, "Migrating enterprise storage to SSDs: analysis of tradeoffs". In Proceedings of EuroSys 2009, Nuremberg, 2009.

[4] S. Sankar, K. Vaid. "Addressing the stranded power problem in datacenters using storage workload characterization". Proceedings of the first WOSP/SIPEW, 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, K. Vaid, "Server Engineering Insights for Large-Scale Online Services". In IEEE Micro, 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 First TPC TC, France, 2009

[9] Adaptec MaxIQ. 32GB SSD Cache Performance Kit.

[10] ETW: Event Tracing for Windows: `http://msdn.microsoft.com/en-us/library/bb968803%28VS.85%29.aspx`

[11] H. Vandenbergh.Vdbench: User Guide. Version:5.00 October 2008

[12] SQLIO Disk Subsystem Benchmark Tool.

[13] I. Ahmad. "Easy and Efficient Disk I/O Workload Characterization in VMware ESX Server". VMware Inc. In IISWC 2007.

[14] J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". OSDI'04. CA, December, 2004.