Real-Time Characterization of Data Access Correlations

Bryan Harris, Michael Marzullo, and Nihat Altiparmak
Computer Science & Engineering Department
University of Louisville
{bryan.harris.1,michael.marzullo,nihat.altiparmak}@louisville.edu

2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS) | 978-1-7281-8643-6/21/$31.00 ©2021 IEEE | DOI: 10.1109/ISPASS51385.2021.00031

***Abstract*—Efficient and accurate detection of data access correlations can provide various storage system optimizations. However, existing offline detection techniques are costly in terms of computation and memory, and they rely on previously recorded disk I/O traces, thus wasting additional storage space and causing further I/O. Moreover, due to their offline nature, they are inadequate for allowing automatic optimizations. In this paper, we propose a real-time data access characterization framework that eliminates the drawbacks of offline analysis with a slight compromise in accuracy. The proposed framework continuously monitors I/O requests and forms correlations by applying single-pass data analysis techniques and maintaining a synopsis data structure to efficiently characterize access behavior in various dimensions including spatial locality, temporal locality, and frequency. Our evaluation using synthetic and real world storage workloads indicates that the proposed framework can detect over 90% of data access correlations in real-time, using limited memory. The proposed framework is crucial for enabling future self-optimizing storage systems that can automatically react to I/O bottlenecks.**

***Index Terms*—data access correlations; online i/o analysis; self-optimizing storage systems**

I. Introduction

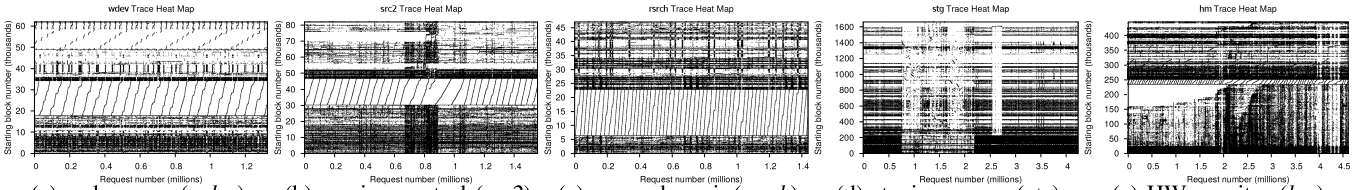
Modern applications have ever increasing demands for data storage, retrieval, and analysis that require predictable and continuously high disk I/O performance. Recent innovations in storage hardware resulted in low-latency Solid-State Drives (SSDs). Nevertheless, even the fastest secondary storage devices are still a few orders of magnitude slower than DRAM [1], and storage performance bottlenecks continue to be one of the major threats limiting the scalability of today's data intensive applications [2]–[4]. In addition to their slower performance compared to primary storage, due to their shared nature, dynamic workload intensity behavior, and the limitations of their internal physics, secondary storage devices are also highly unpredictable in their performance. For instance, garbage collection operation due to the physical limitations of flash storage is a notorious contributor of unpredictability in SSDs [5]. Another example is large scale shared storage systems, such as those used in cloud infrastructures, enterprise data centers, and scientific clusters, where unpredictability arises due to complex data access patterns [6].

Efficient and accurate data access characterization in real-time has the potential to eliminate unpredictable and inconsistent I/O performance, and ultimately generate self-optimizing storage systems. Traditionally, there is significant responsibility on application programmers to write file I/O accesses efficiently. They must consider file sizes, unbuffered versus buffered I/O, synchronous versus asynchronous with callbacks, all using a variety of supported interface libraries. However, even the most diligent application programmer cannot account for all interactions between applications and accesses throughout the storage stack. I/O patterns of many applications are known to be dynamic as their data access behaviors change over time [7]. In addition, multiple I/O intensive instances interacting and simultaneously accessing the same storage system increases the unpredictability of access patterns. Therefore, optimization of storage systems for "assumed" data access patterns of individual applications is no longer helpful, especially in dynamic and multi-tenant environments, where the storage system is shared by multiple applications, containers, virtual machines, etc., that can interact with each other at the storage level.

Data access correlations characterize both how individual applications access data and how multiple applications interact with each other at the I/O request level. In block storage, such correlations can be represented as file system blocks, by correlating data blocks that are frequently accessed together. Using data access correlations, it is possible to detect sequential patterns represented by adjacent blocks and random patterns that are commonly formed as a result of semantic relationships that are harder to infer. Characterizing data access behavior in real-time requires accurate and efficient data analysis techniques. Existing methods use off-the-shelf data mining algorithms that are costly in computation and memory, and operate in offline manner by relying on recorded disk I/O traces. This offline behavior prevents timely reaction to I/O bottlenecks, causes additional disk I/O for writing/reading trace files, and wastes storage space to store trace files.

In this work, we propose an online data access characterization framework that continuously monitors I/O requests and detects data access correlations from the block layer in real time, without relying on recorded disk I/O traces. These access correlations can be used to inform various optimization scenarios, leading to self-optimizing systems, which are beyond the scope of this paper, but some suitable applications are discussed in Section V. Our characterization framework maintains a synopsis data structure by applying "single pass" data analysis techniques to efficiently characterize data access behavior in various dimensions, including spatial locality (sequentiality), frequency, and temporal locality (recency). Once data access correlations are detected, they can enable various automatic storage system optimizations including but not limited to caching, prefetching, data placement, energy efficiency, garbage collection, I/O scheduling, and wear-leveling.

978-1-7281-8643-6/21/$31.00 ©2021 IEEE
DOI 10.1109/ISPASS51385.2021.00031



(a) web server (*wdev*) (b) version control (*src2*) (c) research proj. (*rsrch*) (d) staging server (*stg*) (e) HW monitor (*hm*)

Fig. 1: Storage heat maps of enterprise servers from Microsoft

II. Background and Related Work

In this Section, we first provide background information on data access correlation and frequent itemset mining, and then discuss the related work on existing frequent itemset mining techniques and correlation based storage system optimizations.

A. Background

Data access correlations derive from disk I/O requests that are frequently performed together, or within a very short time interval [8]. Such correlations are commonly encountered in storage workloads and it is possible to observe them in publicly available storage traces. Figure 1 shows storage heat maps of various enterprise servers from Microsoft [9], where the horizontal axis indicates the request sequence and the vertical axis shows the starting block in the block numberspace. Vertical patterns indicate data access correlations, and their horizontal repetition motivates the use of these correlations in storage system optimizations. Google also reveals the existence of data access correlations in their workloads and optimizes Spanner by co-locating correlated directories [10].

In block storage where disk I/O requests are represented as sets of file system blocks, access correlations can be characterized as *intra*-request and *inter*-request block correlations. While intra-request block correlations commonly exist among adjacent blocks of the same I/O request, indicating sequential access patterns, inter-request block correlations are formed between blocks of different I/O requests handled by the storage system at approximately the same time. Inter-request block correlations are especially more difficult to infer and detect in the block layer as they are commonly formed as a result of higher level semantic relationships, such as an inode block and its associated data blocks being correlated, blocks for a web server request being correlated with the blocks of a database table that it interacts with, or inter-tenant data access correlations in multi-tenant shared storage scenarios.

Figure 2 illustrates an example in which two requests occur within a brief amount of time (what we refer to as the *transaction window*), the first request for four blocks (100–103) and the second for three blocks (200–202). Within each, every possible unique pairing of blocks forms an intra-request block correlation. In addition, each block in the first request is correlated with every block of the second request, forming inter-request block correlations, as they are part of the same transaction. The number of both types of block correlations is quadratic on the sizes of the two requests; there are $\binom{n}{2}$ unique

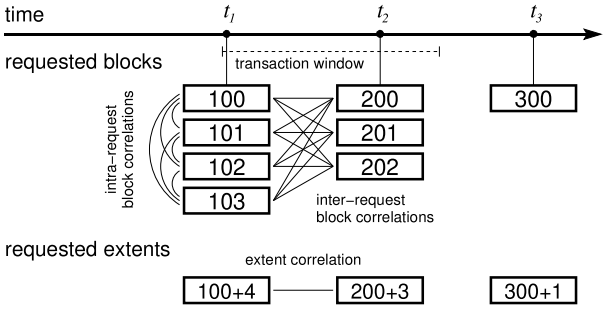


Fig. 2: Block based and extent based correlations

intra-request block correlations for a request of n adjacent blocks and nm unique inter-request block correlations for two requests of sizes n and m. The total number of inter-request block correlations becomes more complex (higher order polynomial) if three or more requests occur together in a transaction. As well as the correlations themselves, various additional information can also be extracted from storage workloads such as correlation strengths (frequency) and types (R/W), which can lead to better optimizations.

Data access correlations can be detected using frequent itemset mining (FIM) techniques. The original motivation of FIM was the need to analyze supermarket customer behavior in order to discover which products were purchased together and with what frequency. Guided by this information, marketers can place trivially correlated products (i.e. gin and tonic) and more importantly nontrivially correlated products (i.e. the myth of beer and diaper correlation [11]) next to each other on the shelves in order to boost sales, as the customer serendipitously picks up another item. FIM algorithms take a series of transactions as input, and output associated items with a frequency greater than a specified minimum support.

B. Related Work

Within the domain of FIM, there are offline and stream based algorithms [12], where many can be categorized as variants *apriori* [13], *eclat* [14], and *fp-growth* [15]. *apriori* performs a scan of the transactions to first filter all items that are not frequent and then finds the associated items from the filtered input, which generally provides faster results for the storage workloads that we investigated, but causes prohibitively high memory consumption. *eclat* uses a depth-first search of intersecting transactions, which reduces the memory consumption but significantly increases the running

time. And *fp-growth* is a tree based algorithm that provides a resource trade-off between *apriori* and *eclat*. As a result, the running time or memory usage (or both) of these offline FIM algorithms end up being unpractical for storage workload analysis. In addition, since they rely on previously stored traces, they waste storage space and cause additional disk I/O in order to store and retrieve the traces. Finally, due to their offline nature, they can only provide a delayed analysis and optimization opportunity. Nevertheless, offline algorithms establish a baseline for accuracy comparison.

Stream based FIM algorithms were explored more recently to apply association rule mining techniques to continuous data. A popular stream based FIM algorithm is *estDec+* [16], which implements a Compressible Prefix (CP) tree to dynamically accept items from a stream source. This tree structure adapts its performance to a configured memory size. The CP-tree merges and splits nodes to efficiently use the allocated memory space for the tree. The *estDec+* algorithm's accuracy is determined by the proportion of the allocated memory of the tree versus the size and throughput of the input stream. However, based on our experimentation using modern storage devices and real workloads, the performance of stream based FIM algorithms including *estDec+* is not adequate to handle the pace of disk I/O streams with a reasonable accuracy. This is mainly due to the focus of stream based FIM algorithms to generate frequent itemsets of maximum size rather than only pairs of items. Frequent pairs alone is sufficient for identifying data access correlations, therefore our proposed technique focuses on frequent pairs rather than maximum itemsets.

FIM was first applied to the storage domain in *C-Miner* [8], which is an offline mining algorithm that finds correlated data blocks from sources such as file systems or databases. A "gap" measurement is defined in C-Miner to limit the maximum distance between frequent subsequences. This creates a sliding window in which C-Miner processes items, which limits its spatial locality. In addition, temporal locality is not considered for the generated block association rules. In addition to C-Miner, two additional follow-up work also proposed offline data access correlation techniques, specifically using vectorized representation of file access [17] and deep learning methods [18]. However, similar to C-Miner, these are also offline techniques and suffer from the same insufficiencies of offline access characterization such as relying on previously stored workload traces and preventing timely optimizations.

Data access correlations from offline analysis were used in various storage system optimizations, such as caching [19], [20], prefetching [17], [21], data placement [4], [22], and energy-efficiency [23]. The wide optimization potential of data access correlations motivates us to develop an efficient and accurate real-time characterization framework enabling faster reaction to I/O bottlenecks and concept drifts in data access.

III. Real-Time Data Access Characterization

In this section, we present our proposed real-time data access characterization framework. Our framework is fundamentally platform independent; all that is required of the operating system is access to listen for block level I/O requests. We have chosen Linux and its *blktrace* tool for our evaluation, but similar tools, such as *perfmon* for Windows, can also monitor storage I/O requests for other operating systems.

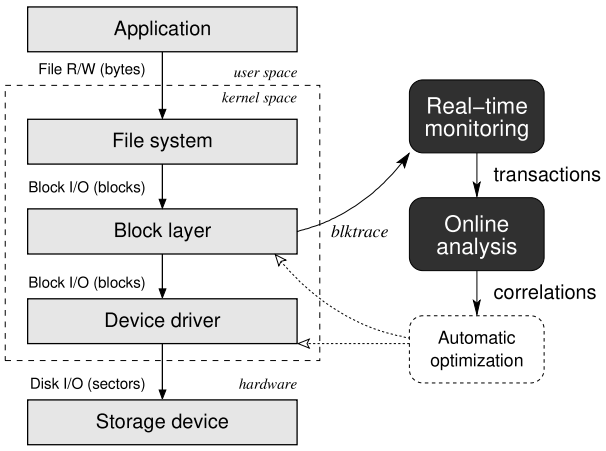


Fig. 3: Overview of the proposed framework

Our proposed framework is illustrated in Figure 3. The existing storage stack appears on the left and the proposed modules on the right. Our monitoring module groups I/O requests from the block layer into "transactions," or sets of requests that occur together, to be processed by the online analysis module. This analysis module collects information about data access correlations from the transactions. These correlations are then to inform an automatic optimization module, which is application specific and beyond the scope of this work; however, we provide insights on possible optimization scenarios as part of our discussion in Section V.

A. Block vs. Extent Correlations

Considering every possible pairing of individual blocks is too complex to be practical for real-time analysis. In the block layer, rather than list individual block numbers, I/O requests are specified as one or many adjacent blocks, given as a starting block number and size—what we refer to as an *extent*. In addition to *block* based data access correlations, it is also possible to form *extent* based correlations, which simplifies both the expression of requests and the consideration of correlations. In our example shown in Fig. 2, we have two requested extents in the same transaction forming only one extent correlation, 100+4 and 200+3, which is much simpler than the numerous block correlations. We can infer the $\binom{4}{2}+\binom{3}{2}=9$ intra-request and $4\times 3=12$ inter-request block correlations from this one extent correlation. Even though the number of extent correlations is also quadratic on the number of extents requested together, $\binom{N}{2}$ for N extents in the same transaction, expressing requests as extents prevents the number of pairing from exploding into higher order polynomials, as with block based correlations. Limiting the complexity of each transaction of correlated requests is important for scalability and real-time processing, and it is more consistent with the native form of how requests are made in the block layer.

Due to their reduced complexity, we propose extent based correlations instead of block based correlations, which leads to improved performance in real-time analysis. However, it is also important that considering only extent based correlations approximates the accuracy of entire block based correlations as it may cause a potential loss of overlooking some block correlations if requested extents are not requested repeatedly using the exact same shape (starting block and size). For example in Fig. 2, extent correlation 100+4 and 200+3 would be counted separately from another potential extent correlation of 101+3 and 201+2, which would potentially undercount the frequency of resulting block correlation between 101 and 201. Based on our analysis using real world Microsoft workloads shown in Figure 1, extents of same shape repeat themselves with very high frequency and therefore, this potential under-counting issue does not cause a significant accuracy loss while providing a significant performance improvement. At the end, we are interested in only *frequently* correlated data rather than all possible pairings of requested blocks.

B. Transaction Window

A "transaction" is a set of items (extents) that are coincident in time, or requested within a brief window of time such that they are considered correlated. Items that appear together in multiple transactions are *frequent correlations*. The size of this transaction window, or length of time, may be static or dynamic. For example, we can approximate I/O requests handled by the storage subsystem around the same time by simply using a static window duration of time t. However, this t may need to be adjusted over time depending on workload changes and storage system performance, which is affected by various factors such as the media type, operating system, server workload, etc. Rather than use of a static value, we propose to monitor the performance of the storage subsystem and adjust the transaction window accordingly, depending on the average access latency of the I/O requests.

In our evaluation framework, we used a transaction window size of double the average I/O latency, which provides a good performance experimentally and can easily be adjusted as needed. Using such a dynamic transaction window allows adjustment of the transaction window over time depending on workload and performance changes. The Linux kernel already maintains similar information for use in its recently implemented hybrid polling technique [24].

C. Real-Time Monitoring

The Linux kernel supports a tracing tool called *blktrace* that reports kernel block layer events to user space. The block layer, shown in Figure 3, lies below the application and file system layers. Applications send read or write requests to the kernel where the virtual file system (VFS) and file system translate them into block I/O requests. The block layer implements performance enhancements such as I/O scheduling and request merging before translating block I/O requests into physical sectors to be submitted to the storage device driver, and eventually the storage device controller itself.

The *blktrace* tool can be difficult for an average user or system administrator to use, as it provides numerous features and detailed information from multiple collection points in the block layer. It is typically used for creating a trace for offline analysis. This *blktrace* information is in a custom binary format that includes timestamp, event type, process ID that made the request, starting block number, and size of the request in bytes. This binary output is typically converted into a human readable text format using another tool, *blkparse*. The proposed real-time monitoring module uses the *blktrace* API to interpret trace events and gathers them into transactions, without using *blkparse* and thus reducing runtime overhead.

The real-time monitoring module launches and configures *blktrace* to listen for "issue" events from the block layer for a specific storage device. It groups issued requests into transactions based on their timestamps and the desired transaction window, and then passes these transactions to the online analysis module. For the purposes of our evaluation, the monitor filters events by process ID (PID) and process group ID so that only events generated by our workload processes are measured. Whether or not to filter events by PID may depend on the needs of the optimization desired.

D. Online Analysis

An online data access characterization technique needs to provide accurate results in a computationally efficient manner with limited memory usage. However, frequent itemset mining based techniques suffer from either excessive computation or excessive memory usage as they tend to perform more than what is required for access correlations. Instead, our design is inspired from fundamental cache replacement methods in order to achieve efficient results with minimal loss of accuracy. Various cache replacement techniques have been proposed in the literature [25]–[31]. Among these, Adaptive Replacement Cache (ARC) [31] seems to be the most suitable approach to be utilized for detecting data access correlations in real-time as it allows dynamic adaption to changing workloads while considering item frequency.

Inspired by ARC, our proposed online analysis module performs a single-pass data access characterization on received transactions by maintaining a synopsis data structure that incorporates spatial locality (*sequentiality*), frequency, and temporal locality (*recency*), where *sequentiality* is incorporated with the use of extent correlations, promotion from one tier to the next is performed based on *frequency*, and *recency* is incorporated based on an LRU-based eviction approach. Since we are "caching" only the metadata of requests and not the actual data stored on the storage devices, we refer to our synopsis data structure as a table rather than a cache.

Similar to ARC, we use a two-tier design where the first tier stores items seen infrequently (once) and items are promoted to the second tier upon a cache hit in the first. However, unlike ARC, rather than use an adaptive size scaling against a ghost cache, we use a fixed size. We also use two tables; the first "item table" stores individual extents and second "correlation table" stores pairs of extents seen together in transactions. Also

unlike ARC, rather than move evicted entries to a ghost cache, pairs in the correlation table are demoted to the end, making them next in line for eviction by LRU.

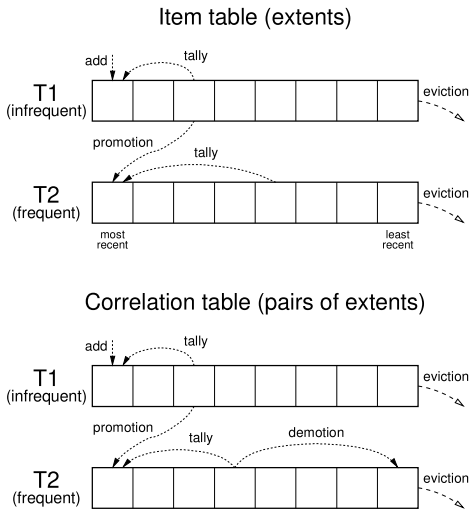


Fig. 4: Online analysis schematic

1) Proposed Synopsis Data Structure: The proposed online analysis module consists of two separate two-tier tables, as illustrated in Figure 4. The first table, *item table*, stores individual extents while the second table, *correlation table*, stores pairs of extents occurring together in a transaction. For each table, the first tier T1 maintains values seen "infrequently" while T2 maintains entries seen "frequently." An entry of each tier maintains an extent and a frequency counter or tally. Both tiers use an LRU approach; on each lookup hit, the entry is moved to the front of the queue and its frequency counter is incremented. On each miss, an entry is evicted from the least recently used position to make space at the front of the queue for the new entry. An entry is promoted from T1 to T2 if its counter reaches a given threshold. In order to reduce the relevancy of an entry without immediate eviction, we demote it to the end, marking it next for eviction. As a result, this two-tier design achieves a balance between recency and frequency.

2) Cost Analysis and Optimization: While processing a transaction, online analysis module inserts each extent in a transaction to the *item table*, while also inserting every unique extent pair within that transaction to the *correlation table*. Since frequent correlations must involve frequent extents, when an extent is evicted from the item table, we also demote it in the correlation table. For N extents in a transaction, there are $\binom{N}{2}$ combinations of pairs, requiring $\Theta(N^2)$ additions to the correlation table. This provides a challenge for an algorithm operating on a linear stream of transactions. However, in order to limit the number of operations at this stage, the monitoring module has a configurable limit to the size of a transaction. If a transaction exceeds this limit, the monitor will simply place additional items into a new transaction. This causes a potential loss in detection of frequent correlations, but helps to ensure stable stream processing. For our evaluation, we used a limit of eight I/O requests per transaction. In addition, not every request necessarily needs to be sent from monitoring to analysis module. We noticed that for some of our real world traces (*wdev* in particular), the same request was repeated more than once in a transaction window. Allowing repeated requests in a transaction can possibly distort the correlation frequencies, so there is a need for deduplication of requests in a transaction. This deduplication can easily be performed in $O(N^2)$ time for N items per transaction and does not add to the complexity of processing a transaction.

IV. Evaluation

In this section, we evaluate the performance of the proposed real-time data access characterization framework by comparing its efficiency and accuracy against existing offline techniques using synthetic and real world storage workloads.

A. Experimental Setup

For realistic experimentation, we replay synthetic and real world storage workloads on real hardware, with monitoring and analysis performed in the background in real-time. Our test system is a Dell PowerEdge R230 server with an Intel Xeon CPU E3-1230 v5 (3.40GHz) and 64 GB RAM. We replayed workloads with *fio* (the "Flexible I/O Tester") [32] on a Samsung 960 EVO SSD (500 GB).

For the proposed real-time characterization, while we replayed workloads while monitoring and analyzing in real-time, existing offline techniques stored the monitoring data to disk and performed offline analysis after the replaying of the trace, similar to how they would work in practice. As an offline analysis technique, we used Frequent Itemset Mining (FIM), specifically Borgelt's *eclat*, *apriori* and *fp-growth* implementations [33]. These three algorithms demonstrate a range of time-space tradeoffs.

B. Workloads

In order to test how well our proposed technique detects correlations, we use a range of both synthetic and real world workloads. While synthetic workloads provide us full control over the generated data access correlations, knowing what and what not to detect, real world workloads allow us to test the proposed framework using realistic storage workloads including real data access correlations and arrival patterns.

1) Synthetic workloads: Data access correlations are highly dependent on I/O request workload. In order to understand how our online analysis handles access correlations, we examine its performance using three traces constructed with specific properties and known inter- and intra-request correlations of increasing complexity:

one-to-one — *a single block requested with another non-contiguous single block.* This smallest case models two variables or small data records that are associated at the application level, for which access of one occurs with access of the other.

one-to-many — *a single block correlated with a range of contiguous blocks.* A single block (512 B) is accessed whenever a range of blocks, chosen at random from 512 B to

1 MB, is accessed. One example of such a case is accessing the contents of a small file together with its inode.

many-to-many — *contiguous blocks correlated with other contiguous blocks.* The sizes of extents are chosen at random from 512 B to 1 MB. One example of this case may be a web server and associated database, in which an access to a file for a specific web resource is correlated with an access to a database table on disk.

In each of these three synthetic workloads, there are four constructed inter-request correlations of the specific type. These are ranked in popularity using a Zipf-like distribution [34], in which its probability of occurring is inversely proportional to its rank. With four correlations, the probability of each is 48%, 24%, 16%, and 12%. The arrival times of these correlated events are chosen at random such that the time between requests (interarrival time) follows an exponential distribution with mean 200 ms, which is large so that two sets of constructed correlations will not merge into the same transaction. In order to challenge the algorithm, we have inserted noise of random requests following an average interarrival time of 100 ms, ranging in size from 512 B to 8 KB. This noise introduces extents into transactions at random, therefore contributing to infrequent and "false" correlations.

2) Real world workloads: In addition to the synthetic workloads, we also evaluate our framework using five publicly available real world storage workloads provided by Microsoft [9]. These workloads include week-long block level I/O requests of various enterprise and production servers running at Microsoft Research Cambridge, including a web server (*wdev*), a version control server (*src2*), a research projects server (*rsrch*), a staging server (*stg*), and a hardware monitoring server (*hm*). These traces are widely used [35]–[37] and include a mix of applications with various I/O patterns as shown in Figure 1. Further statistics related to these traces, including total data amount accessed, unique data amount accessed, and the percentage of requests that have interarrival time of less than 100 µs are provided in Table I.

TABLE I: Microsoft workload statistics

Workload		Total Data accessed	Unique data accessed	Interarrival % $< 100\,\mu s$
wdev	test web server	11.3 GB	0.53 GB	78.4%
src2	version control	109.9 GB	26.4 GB	71.2%
rsrch	research projects	13.1 GB	0.97 GB	77.4%
stg	staging server	107.9 GB	83.9 GB	65.9%
hm	hardware monitor	39.2 GB	2.42 GB	67.0%
	average:	43.3 GB	7.58 GB	73.5%

Each Microsoft trace is composed of multiple disk IDs. In order to create the original workload on our single disk test system, for each Microsoft trace, we replayed the trace of the disk with the greatest number of requests directly on our test device (Samsung 960 EVO SSD) using original block numbers. Since these traces were recorded using HDDs with significantly greater access latencies than our NVMe SSD test device, for fair evaluation, we accelerated the traces according to their relative access latency, as shown in Table II. For this purpose, we first calculate the average latency as reported in the trace (second column). For a relative performance comparison of the device recorded in the trace and of our test device, we replayed the trace 10 times with *fio* as synchronous requests, ignoring trace timestamps (using the *replay_no_stall* option). We recorded the average read latency of our test device (third column) across the 10 replays. Since writes may be cached and reported as complete before actually writing in the device internally, we use only the read latency as a metric of its performance. Comparing the average latency recorded in the trace to our average replayed latency yields our replay speedup (fourth column), which we use to accelerate the request arrival rate in our evaluation. Therefore, the actual arrival rate of the requests in our evaluation is 61.2 to 473 times faster that the original interarrival rate reported in Table I, which forces the proposed framework even further.

TABLE II: Replay speedup of Microsoft traces

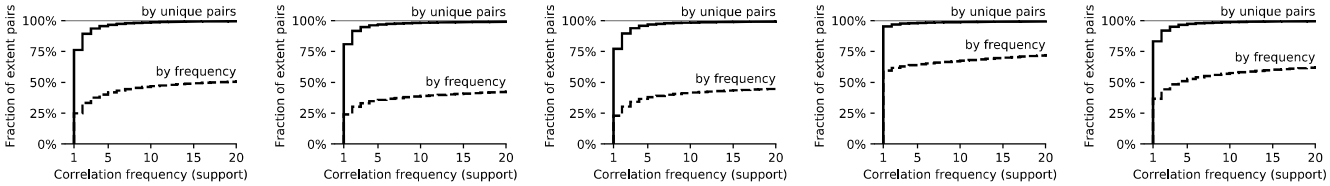
Workload	Mean trace latency	Mean measured latency	Replay speedup
wdev	3.65 ms	48.00 µs	76.0×
src2	3.88 ms	63.35 µs	61.2×
rsrch	3.02 ms	31.79 µs	94.9×
stg	18.94 ms	40.06 µs	473×
hm	13.86 ms	63.84 µs	217×

Transactions generated by our real-time monitoring module are both stored for offline analysis and also passed to the online analysis module in real-time. Using the offline FIM analysis data, we were able to examine properties of the workloads, transactions, and correlations discovered.

C. Experimental Results

1) Offline Characterization of Real World Workloads: Online characterization of real world storage workloads is crucial for achieving self-optimizing storage systems that can adjust themselves for their workloads. Most of this analysis should be performed in real-time, as in our proposed framework, so that the performance adjustments can be performed in a timely manner. However, it is also useful to perform some offline analysis and gain insights about the general characteristics of the workloads, which we use to design our proposed real-time characterization framework.

Figure 5 shows the cumulative distribution function (CDF) of extent correlations mined from the real world traces using offline FIM. The vertical axis shows the fraction of extent correlations counted by the number of *unique* extent pairs (solid line) and weighted by frequency (dashed line). The horizontal axis is the correlation frequency, or how often each extent correlation occurs in the transactions generated by the monitoring module. In all workloads, the solid lines rise quickly for small support (minimum correlation frequency) values, indicating that the majority of *unique* extent pairs are infrequent. For example, in the three traces on the left (*wdev*, *src2*, and *rsrch*), we can see that three quarters of the



(a) *wdev* (b) *src2* (c) *rsrch* (d) *stg* (e) *hm*

Fig. 5: Cumulative distribution of extent correlations by frequency

unique extent pairs occur only once (support 1). Since we are interested only in frequent correlations, this shows us that the number of extent pairs that must be stored by our synopsis data structure is actually significantly smaller as we should ignore this large majority of infrequent pairs. Despite their greater number, infrequent pairs contribute little to the total requests because of their low frequency. The solid line rises quickly, indicating that few pairs are frequent, and the dashed line rises slowly, indicating that the remaining frequent pairs represent a greater portion of the total trace, indicating a Zipf-like distribution [34]. This means that by raising the supported frequency even a little, we can significantly reduce the necessary size of the synopsis data structure while maintaining a valuable representation of extent correlations.

Using offline characterization, we can also gain insights into optimal conditions of the relationship between table size and hit rate. If we could record only one extent pair, then the most frequent pair would give us the greatest percentage of total frequencies, and the best possible hit rate. If we sort all extent pairs in decreasing order by their frequency, then the sum of frequencies for the n most frequent pairs is the optimal fraction of overall total frequency possible for any n pairs. Figure 6 plots this number n of unique pairs against the sum of frequencies of these n most frequent pairs for our real world traces. This relates the size of a correlation table to its optimal frequency, and inversely, the minimum table size necessary to represent any given fraction of total frequency. Looking at Fig. 6, it is possible to represent roughly 40% of all extent correlations for all traces using a small table size. On the other hand, roughly half a million entries is enough to represent all extent correlations in *wdev*, *src2*, or *rsrch*. Therefore in our evaluation, we use correlation table sizes of powers of two from 16K through 4M, as this size should be enough to capture all traces.

We found using equal sizes for T1 and T2 to be appropriate for our framework; however, their ratio can be adjusted dynamically for specific applications. One important issue is that in order to process through a sufficient amount of noise of infrequent requests and collect valuable frequent information, the synopsis data structure needs to have a sufficiently large T1. Therefore, respecting to minimum fixed sizes for the T1 and T2 tables is important in a dynamic resizing technique, which would otherwise end up favoring T2.

One entry of both T1 and T2 for the item table contains an item and a frequency counter. Using a 64-bit block ID

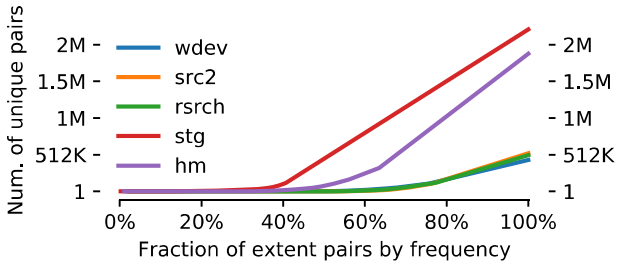


Fig. 6: Table size necessary to support real world traces

and 32-bit length, an extent is represented using 12 bytes. With a 32-bit frequency counter, one entry in the item table is therefore 16 bytes. A single correlation table entry contains two items and a counter, or 28 bytes. Since we used the same number of entries C in both T1 and T2, the size of our item table is $32C$ bytes and the size of our correlation table is $56C$ bytes, or $88C$ bytes for the total synopsis data structure. This is 1.44 MB for $C = 16$ K and 369 MB for $C = 4$ M.

2) Online Characterization of Synthetic Workloads: Figure 7 illustrates the synthetic traces and their correlations. The left column is a heat map of the block layer trace captured using the monitoring module. The second column shows *every pair* (support 1) of distinct blocks that occur together in the same transaction, representing every correlation that occurs at least once. For every pair of single blocks A and B that appear in the same transaction, points (A, B) and (B, A) are plotted. An extent is multiple consecutive blocks, such as $\{A, A + 1, A + 2\}$; these appear as squares on the diagonal line from lower left to upper right. The larger the square, the larger the extent. Rectangles away from the diagonal are two discontinuous extents requested in the same transaction, the width is the size of one extent and the height is the size of the other. An extent correlated with one discontinuous block (a one-to-many correlation) appears as a horizontal or vertical line. Background noise and occasional coincidence creates reflections and ripples throughout the plot.

In order to inform a possible automatic optimization module of the most valuable opportunities for performance improvement, the goal of online analysis is to extract only the *frequent* correlated pairs of blocks. Therefore, the third column of Fig. 7 shows the result of offline frequent itemset mining using *eclat* [33] with support 10, indicating minimum correlation frequency. The circles highlight single points (one-to-one

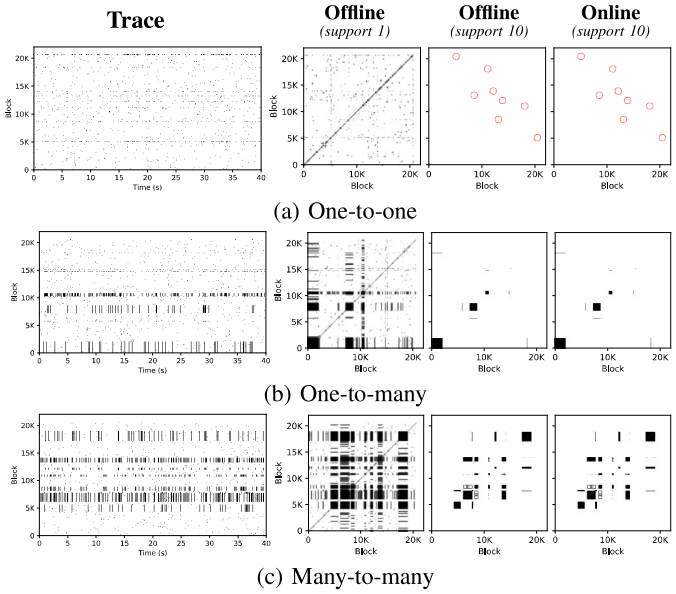


Fig. 7: Visualizations of offline and online analyses

correlations) for emphasis. The rightmost column of Figure 7 illustrates the results of our online analysis module for the three synthetic workloads. Now, when we compare the third and the fourth columns, it is clear from the figures that the proposed online framework captures a majority of important data access correlations by visually yielding a very similar shape with offline.

3) Online Characterization of Real World Workloads: The five real world workloads are far more sophisticated than our four synthetic workloads. They use a larger number space, they contain more correlations than the four we constructed for our synthetic traces, and rather than artificial random noise, they have all the background requests of a natural system. Unlike offline analysis, our online analysis can be performed while the workload is running and without storing a block level trace, thus requiring no additional wall time or storage on disk.

Figure 8 illustrates correlations for the real world traces. The left and middle columns show offline FIM data of *every pair* of blocks requested together (support 1) and only those occurring five or more times (support 5), respectively. The right column shows the extent correlations from our proposed online analysis module that also occur at least five times. We selected *support* $= 5$ for real workloads since this is past the "knee" of the unique pairs curve for all traces (solid line, Fig. 5), which yields a sufficiently high fraction by frequency. As it is clear from Fig. 8, the patterns are visually recognizably similar for offline and online analysis.

By removing infrequently correlated pairs, not only are there noticeably fewer unique correlations visible, but their patterns and structure can change; consider for example *hm* (Fig. 8e): there are many pairs correlated with blocks around number 5M that appear at least once (at left) but not at least five times (middle). Blocks in this region are requested frequently, but appear in transactions with other blocks only

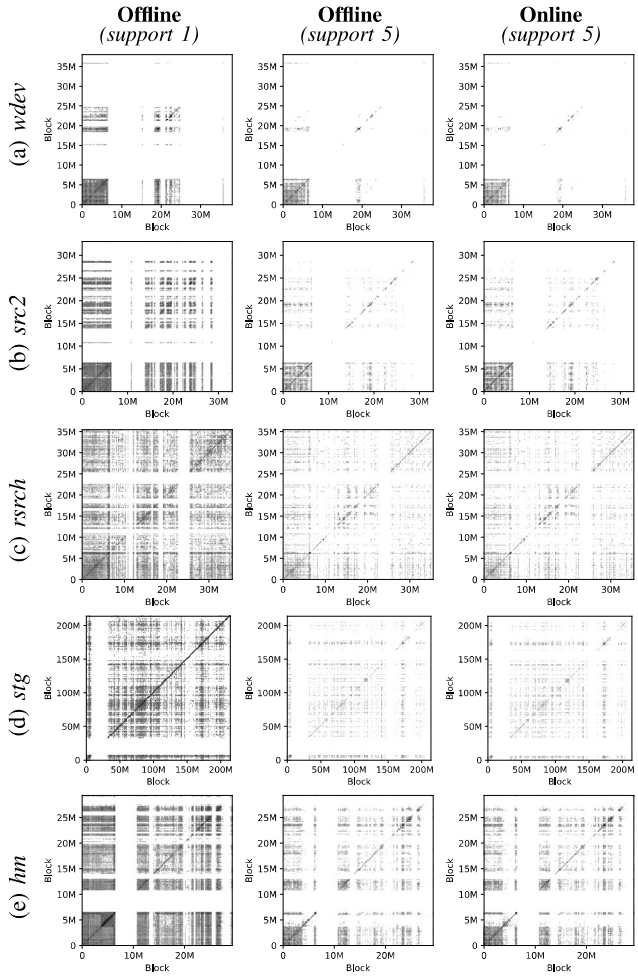


Fig. 8: Offline and online analysis of Microsoft traces

by coincidence. By refining correlated pairs to only those that appear frequently we remove such noise. We can see visually from Figure 8 that the correlated pairs for the online analysis are similar to the offline analysis.

Offline FIM data provides the frequencies of all extent correlations. The similarity of this complete list to the contents of the online analysis synopsis tables gives the percentage of all extent correlations captured by online analysis. Figure 9 shows the percentage captured relative to the optimal percentage possible for the same number of entries (optimal shown in Fig. 6). Notice how, in general, the quality is low for a small table size and increases as the table size increases, eventually reaching 100% when the table is large enough to store every pair. The *stg* trace has the largest number space (an order of magnitude larger than the others) and the majority of extent pairs are infrequent. For *stg*, a very small correlation table (16–32 K entries) cannot keep enough entries that represent a majority of the correlations, since those pairs that eventually do become frequent and worth keeping are evicted by LRU and replaced with less valuable ones. For traces with a long tail of infrequent correlations, such as *stg* and *hm* (see figure

around table size of 2M), the analysis performs poorly versus the optimal solution.

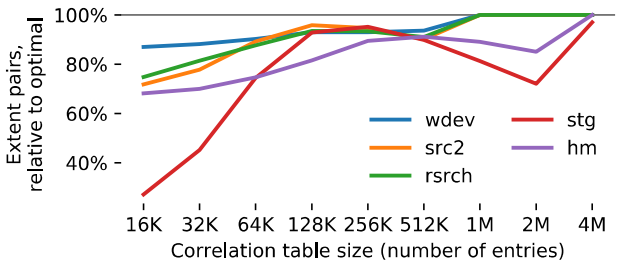


Fig. 9: Representability of extent correlations versus optimal

4) Overhead cost: The overhead cost of monitoring is minimal, since *blktrace* only exposes block layer events to userspace, which are delimited into transactions by the monitoring module. The overhead of the analysis module is $O(N^2)$ for N requests per transaction (see Sec. III-D2); however, this is controllable by enforcing a limit on transaction size (in our evaluation we use a limit of $N = 8$). Memory usage is also controllable by adjusting the synopsis table size of the analysis module. In Sec. IV-C3 and Fig. 9 we examine the trade off of table size versus accuracy with real world workloads.

5) Concept Drift: Data access patterns may change over time, as different applications take dominance of the storage system at different times, resulting in a change, or drift, in concept. Since online analysis techniques need to have a memory limit, they should also be capable of "forgetting" old patterns and adapting to newer patterns over time. In order to illustrate how our proposed method handles this concept drift, we replay parts of two different traces (*wdev* and *hm*) representing two different access patterns or "concepts" to the storage system as a single workload. First, we play the first 100 K requests of *wdev*, then the first 100 K requests of *hm* followed by the second 100 K requests of *wdev*, as illustrated in Figure 10. Suppose that the *hm* requests in the middle represents only a temporary drift in concept. If we were to perform an offline analysis, correlations from this temporary concept would appear in the analysis. However, since our online analysis has a limited knowledge (in Fig. 10 we used a correlation table of size $C = 32$ K entries), we remember only the most significant concept at the end (right).

The lower portion of the figure shows the correlations stored in the synopsis structure at three points in time: at the end of the first *wdev* section, after the inserted *hm* requests, and after more *wdev* requests. This allows our online analysis to adapt dynamically to changes in data access patterns. The pattern of *wdev* forming at the beginning is replaced by the pattern of *hm* in the middle, which begins to fade after more *wdev* requests. The correlation table size (32 K entries) is too small to store both patterns. If we were to instead completely switch to *hm*, the *wdev* correlations would eventually be forgotten.

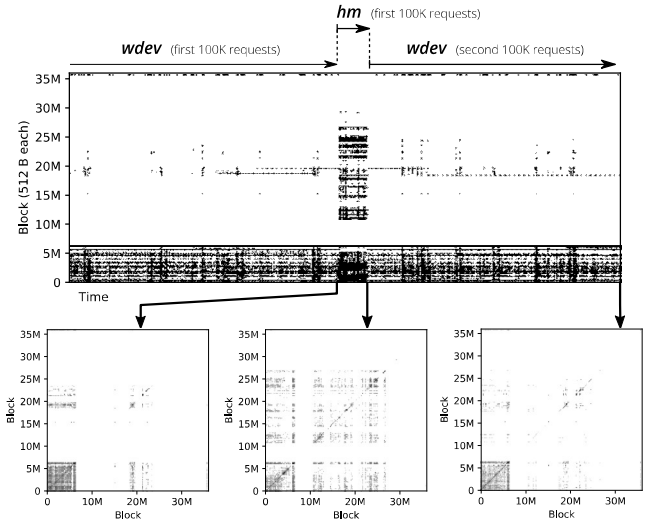


Fig. 10: Learning new concepts and forgetting old ones

V. Discussion: Automatic Optimization

Automatic storage system optimizations informed by the proposed workload characterization framework can take many forms based on application and environment, including but not limited to caching, prefetching, data placement, energy efficiency, garbage collection, I/O scheduling, and wear-leveling. Our ultimate goal in this work is to provide an efficient and accurate real-time data access characterization framework that is general enough to be applied to various optimization scenarios to achieve self-optimizing storage systems that are predictable, can continuously provide high I/O performance, can automatically react to I/O bottlenecks, and adapt to concept drifts in real time. In this section, we briefly discuss two automatic optimization scenarios that our proposed real-time data access characterization framework can enable, leading to a new generation of self-optimizing SSDs.

Existing SSDs are equipped with a vendor-specific Flash Translation Layer (FTL) implemented on the device as propriety firmware, which is mainly responsible for data placement and garbage collection. This "closed" design inhibits the device to be managed and controlled by the host operating system. In the past few years, host controllable SSDs (open-channel [38], [39], multi-stream [40], [41], zoned namespaces (ZNS) SSDs [42], etc.) together with their kernel support [43] have emerged as important technological enablers. Our proposed framework can be integrated into the host system and allow automatic optimizations in new generation SSDs, which could dynamically eliminate existing shortcomings of SSD performance, such as the notorious unpredictable performance issue due to heavy internal data movement of garbage collection or ill-mapped data layout, which consequently cause large tail-latencies [44]–[46].

1) Automatic Garbage Collection Optimization in Multi-Stream SSDs: Multi-stream SSDs (MS-SSD) recently introduced by Samsung have the aim of controlling data placement on the device for decreased Write Amplification Factor (WAF) [40], [41], the ratio of writes on the device to writes

by the host. Log structured designs used in modern SSDs cause new writes to be written to a single append point, even though their *death* times (the time a page is discarded or overwritten by the host [47]) may significantly differ and later cause heavy data movement during garbage collection. MS-SSDs accept write requests from multiple append points (open erase blocks), each called a separate *stream*, so that pages with similar death times can be grouped together. Data with the same stream ID is guaranteed by the device to be written together to a physically related NAND flash block, i.e. Erase Unit (EU). Using the provided multi-stream interface, the host can perform garbage collection optimization and reduce the WAF, which would improve performance and predictability.

Garbage collection optimization can be performed by predicting the *death times* of write requests and placing the data with similar death times in the same Erase Unit (EU) so that the number of valid pages in victim EUs are minimized during garbage collection. In this way, the WAF will be reduced, and consequently device endurance and predictability will also be improved. Death time prediction can be performed with the following assumption:

> *If two or more data chunks were frequently written together in the past, then there is a high chance that their death times will be similar.*

This prediction can continuously be performed using the proposed access characterization framework in real time, which can pass correlated *writes* to the garbage collection optimization module for EU determination. Correlations in this case indicates write extents that are predicted to have similar death times. Therefore, they should be placed in the same EU.

In addition to SSDs, the proposed framework can also allow similar optimizations to be performed in new generation HDDs based on Shingled/Interlaced Magnetic Recording (SMR/IMR) technologies [48], which have similar complexities as SSDs such as write amplification and garbage collection.

2) Automatic Parallel I/O Optimization in Open-Channel SSDs: Similar to MS-SSDs, Open-Channel SSDs (OC-SSD) are a new class of solid-state drives that expose their internal parallelism to the host system and allow the host to manage the devices' internals [38], [39], [49], as opposed to the traditional "closed" designs currently applied in most modern SSDs. OC-SSDs do not have a traditional firmware Flash Translation Layer (FTL) implemented on the device; instead they leave the management of generic FTL functions such as data placement and garbage collection to be implemented in the host. This design allows the host to perform workload-aware data placement decisions, which paves the way for automatic optimization of SSDs for dynamic and multi-tenant workloads, and efficient timing and execution of garbage collection for improved performance and predictability. The SSD device controller implements the remaining device management responsibilities including media-centric metadata and error handling. OC-SSD's internal structure is divided into a set of Parallel Units (PU), where accesses are fully independent of each other. Linux kernel 4.12 and later fully supports OC-SSDs that follow the NVMe specification, by providing a Physical Block Device (*pblk*) layer that implements the basic FTL functionality including mapping logical addresses onto physical addresses (4 KB granularity), guaranteeing the integrity and recovery of the mapping table in case of a crash, handling errors, and maintaining valid page counts per physical block (EU) to be used during garbage collection operations [43]. This subsystem allows a compatible block I/O target device to be controlled by the host and I/O commands to be issued to specific PUs. Several OC-SSD devices were recently announced or made available [38], including CNEX Labs OC-SSD, EMC Dragon Fire Board OC-SSD, LiteOn AD2 OC-SSD, and Radian Memory Systems RMS-325 OC-SSD [50]. In addition to garbage collection, another possible automatic optimization would be to reorganize data that is frequently accessed together to separate parallel units within the OC-SSD in order to improve performance through I/O parallelism.

Initial data placement in SSDs is commonly performed using RAID-0 like striping mechanisms over the Parallel Units (PU) of the device so that internal parallelism can be efficiently exploited [51]–[54]. However, RAID-0 like striping is only effective for large sequential accesses. In addition, due to dynamic logical-to-physical mapping (*out-of-place* updates) applied in modern SSDs, data layout changes over time and the initial striping may end up being largely skewed and inefficient, even for sequential I/O. Previous work indicates that an ill-mapped data layout can cause up to 4.2× higher latency for parallel accesses on an SSD [55].

Parallel I/O optimization can be performed with the following assumption:

> *If two or more data chunks were frequently read together in the past, then there is a high chance that they will be read together in the near future.*

This assumption considers all three important I/O characteristics: *spatial locality*, *frequency*, and *recency*. Using the proposed access characterization framework, it is possible to detect extent correlations in real- time and place correlated *read* extents into different Parallel Units (PU) automatically for improved internal parallelism.

VI. Conclusion

In this work, we propose an online storage workload characterization framework to find data access correlations in real time. Our proposed technique maintains a synopsis data structure using a limited memory footprint, and applies single pass data analysis techniques to efficiently characterize data access correlations by considering sequentiality, frequency, and recency. Once efficiently and accurately determined, data access correlations can inform various storage system optimization including caching, prefetching, data placement, energy efficiency, garbage collection, I/O scheduling, and wear-leveling, and enable self-optimizing storage systems.

Acknowledgments

This research was supported in part by the U.S. National Science Foundation (NSF) under grants CNS-1657296 and OIA-1849213.

REFERENCES

[1] B. Harris and N. Altiparmak, "Ultra-low latency ssds' impact on overall energy efficiency," in *12th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '20)*. USENIX Association, Jul. 2020. https://www.usenix.org/conference/hotstorage20/presentation/harris

[2] D. Singh and C. K. Reddy, "A survey on platforms for big data analytics," *Journal of big data*, vol. 2, no. 1, p. 8, 2015.

[3] H. M. Makrani, H. Sayadi, S. Manoj, S. Raftirad, and H. Homayoun, "Compressive sensing on storage data: An effective solution to alleviate I/O bottleneck in data- intensive workloads," in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2018, pp. 1–8.

[4] E. Tomes, E. N. Rush, and N. Altiparmak, "Towards adaptive parallel storage systems," *IEEE Transactions on Computers*, vol. 67, no. 12, pp. 1840–1848, 2018.

[5] M. Bjørling, J. Gonzalez, and P. Bonnet, "Lightnvm: The linux open-channel SSD subsystem," in *15th USENIX Conference on File and Storage Technologies (FAST '17)*. Santa Clara, CA: USENIX Association, Feb. 2017, pp. 359–374. https://www.usenix.org/conference/fast17/technical-sessions/presentation/bjorling

[6] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Workload analysis of a large-scale key-value store," *SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 1, pp. 53–64, Jun. 2012. http://doi.acm.org/10.1145/2318857.2254766

[7] A. Miranda and T. Cortes, "Analyzing long-term access locality to find ways to improve distributed storage systems," in *20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, Feb 2012, pp. 544–553.

[8] Z. Li, Z. Chen, S. M. Srinivasan, and Y. Zhou, "C-miner: Mining block correlations in storage systems," in *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, ser. FAST '04. Berkeley, CA, USA: USENIX Association, 2004, pp. 173–186. http://dl.acm.org/citation.cfm?id=1096673.1096695

[9] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *Trans. Storage*, vol. 4, no. 3, pp. 10:1–10:23, Nov. 2008. http://doi.acm.org/10.1145/1416944.1416949

[10] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford, "Spanner: Google's globally-distributed database," in *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 251–264. http://dl.acm.org/citation.cfm?id=2387880.2387905

[11] M. Whitehorn, "The parable of the beer and diapers: Never let the facts get in the way of a good story," https://www.theregister.com/2006/08/15/beer_diapers/, Aug 2006, the Register.

[12] C. Borgelt, "Frequent item set mining," *WIREs Data Mining Knowl. Discov.*, vol. 2, p. 437456, Nov. 2012. http://onlinelibrary.wiley.com/doi/10.1002/widm.1074/abstract

[13] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *SIGMOD '93*, ser. SIGMOD '93. New York, NY, USA: ACM, 1993, pp. 207–216. http://doi.acm.org/10.1145/170035.170072

[14] M. J. Zaki, "Scalable algorithms for association mining," *IEEE Trans. on Knowl. and Data Eng.*, vol. 12, no. 3, pp. 372–390, 2000.

[15] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 2000, pp. 1–12.

[16] S. J. Shin, D. S. Lee, and W. S. Lee, "CP-tree: An adaptive synopsis structure for compressing frequent itemsets over online data streams," *Information Sciences*, vol. 278, pp. 559 – 576, 2014. http://www.sciencedirect.com/science/article/pii/S002002551400365X

[17] P. Xia, D. Feng, H. Jiang, L. Tian, and F. Wang, "Farmer: A novel approach to file access correlation mining and evaluation reference model for optimizing peta-scale file system performance," in *Proceedings of the 17th International Symposium on High Performance Distributed Computing*, ser. HPDC '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 185196. https://doi.org/10.1145/1383422.1383445

[18] D. Dai, F. Bao, J. Zhou, X. Shi, and Y. Chen, "Vectorizing disk blocks for efficient storage systems via deep learning," *International Journal of Parallel Computing*, Apr. 2018.

[19] Z. Li, Z. Chen, and Y. Zhou, "Mining block correlations to improve storage performance," *ACM Trans. Storage*, vol. 1, no. 2, p. 213245, May 2005. https://doi.org/10.1145/1063786.1063790

[20] S. Lee, Y. Won, and S. Hong, "Mining-based file caching in a hybrid storage system," *J. Inf. Sci. Eng.*, vol. 30, pp. 1733–1754, 2014.

[21] G. Soundararajan, M. Mihailescu, and C. Amza, "Context-aware prefetching at the storage server," in *USENIX 2008 Annual Technical Conference*, ser. ATC '08. USA: USENIX Association, 2008, p. 377390.

[22] E. N. Rush, B. Harris, N. Altiparmak, and A. c. Tosun, "Dynamic data layout optimization for high performance parallel I/O," in *23rd IEEE International Conference on High Performance Computing, Data, and Analytics (HiPC 2016)*, Hyderabad, India, December 2016.

[23] Y. Deng, J. Cai, W. Jiang, and X. Qin, "Employing dual-block correlations to reduce the energy consumption of disk drives," *Computing*, vol. 99, no. 3, p. 235253, Mar. 2017. https://doi.org/10.1007/s00607-016-0488-7

[24] D. L. Moal, "I/o latency optimization with polling," https://events.static.linuxfound.org/sites/events/files/slides/lemoal-nvme-polling-vault-2017-final_0.pdf, Mar 2016.

[25] A. V. Aho, P. J. Denning, and J. D. Ullman, "Principles of optimal page replacement," *J. ACM*, vol. 18, pp. 80–93, 1971.

[26] P. J. Denning, "Working sets past and present," *IEEE Transactions on Software Engineering*, vol. SE-6, no. 1, pp. 64–84, 1980.

[27] J. T. Robinson and M. V. Devarakonda, "Data cache management using frequency-based replacement," *SIGMETRICS Perform. Eval. Rev.*, vol. 18, no. 1, p. 134142, Apr. 1990. https://doi.org/10.1145/98460.98523

[28] E. J. O'Neil, P. E. O'Neil, and G. Weikum, "The lru-k page replacement algorithm for database disk buffering," in *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '93. New York, NY, USA: Association for Computing Machinery, 1993, p. 297306. https://doi.org/10.1145/170035.170081

[29] T. Johnson and D. Shasha, "2Q: A low overhead high performance buffer management replacement algorithm," in *Proceedings of the 20th International Conference on Very Large Data Bases*, ser. VLDB '94. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994, pp. 439–450. http://dl.acm.org/citation.cfm?id=645920.672996

[30] S. Jiang and X. Zhang, "Lirs: An efficient low inter-reference recency set replacement policy to improve buffer cache performance," in *Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 3142. https://doi.org/10.1145/511334.511340

[31] N. Megiddo and D. S. Modha, "ARC: A self-tuning, low overhead replacement cache," in *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, ser. FAST'03. Berkeley, CA, USA: USENIX Association, 2003, pp. 115–130. http://dl.acm.org/citation.cfm?id=1090694.1090708

[32] J. Axboe, *Flexible I/O Tester*, https://github.com/axboe/fio.

[33] C. Borgelt, "Efficient implementations of apriori and eclat," in *Proc. 1st IEEE ICDM Workshop on Frequent Item Set Mining Implementations (FIMI 2003, Melbourne, FL). CEUR Workshop Proceedings 90*, 2003, p. 90.

[34] L. Breslau, Pei Cao, Li Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: evidence and implications," in *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)*, vol. 1, 1999, pp. 126–134 vol.1.

[35] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron, "Migrating server storage to ssds: Analysis of tradeoffs," in *Proceedings of the 4th ACM European Conference on Computer Systems*, ser. EuroSys '09. New York, NY, USA: ACM, 2009, pp. 145–158. http://doi.acm.org/10.1145/1519065.1519081

[36] T. Xie and Y. Sun, "Dynamic data reallocation in hybrid disk arrays," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 9, pp. 1330–1341, Sept 2010.

[37] A. Miranda and T. Cortes, "Craid: Online raid upgrades using dynamic hot data reorganization," in *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST '14)*. Santa Clara, CA: USENIX, 2014, pp. 133–146. https://www.usenix.org/conference/fast14/technical-sessions/presentation/miranda
[38] "Open-channel solid state drives." http://lightnvm.io/, 2017.
[39] "The openssd project." http://openssd.io/, 2017.
[40] J.-U. Kang, J. Hyun, H. Maeng, and S. Cho, "The multi-streamed solid-state drive," in *6th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 14)*. Philadelphia, PA: USENIX Association, 2014. https://www.usenix.org/conference/hotstorage14/workshop-program/presentation/kang
[41] J. Yang, R. Pandurangan, C. Choi, and V. Balakrishnan, "Autostream: Automatic stream management for multi-streamed ssds," in *Proceedings of the 10th ACM International Systems and Storage Conference*, ser. SYSTOR '17. New York, NY, USA: ACM, 2017, pp. 3:1–3:11. http://doi.acm.org/10.1145/3078468.3078469
[42] "Zoned namespaces (ZNS) SSDs," https://zonedstorage.io/introduction/zns/.
[43] M. Bjørling, J. Gonzalez, and P. Bonnet, "Lightnvm: The linux open-channel SSD subsystem," in *15th USENIX Conference on File and Storage Technologies (FAST 17)*. Santa Clara, CA: USENIX Association, 2017, pp. 359–374. https://www.usenix.org/conference/fast17/technical-sessions/presentation/bjorling
[44] F. Chen, D. A. Koufaty, and X. Zhang, "Understanding intrinsic characteristics and system implications of flash memory based solid state drives," in *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '09. New York, NY, USA: ACM, 2009, pp. 181–192. http://doi.acm.org/10.1145/1555349.1555371
[45] M. Bjørling and J. Gonzalez, "Multi-tenant i/o isolation with open-channel ssds," in *Nonvolatile Memory Workshop (NVMW '17)*, 2017.
[46] J. Huang, A. Badam, L. Caulfield, S. Nath, S. Sengupta, B. Sharma, and M. K. Qureshi, "Flashblox: Achieving both performance isolation and uniform lifetime for virtualized ssds," in *15th USENIX Conference on File and Storage Technologies (FAST '17)*. Santa Clara, CA: USENIX Association, 2017, pp. 375–390. https://www.usenix.org/conference/fast17/technical-sessions/presentation/huang
[47] J. He, S. Kannan, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "The unwritten contract of solid state drives," in *Proceedings of the Twelfth European Conference on Computer Systems*, ser. EuroSys '17. New York, NY, USA: ACM, 2017, pp. 127–144. http://doi.acm.org/10.1145/3064176.3064187
[48] F. Wu, B. Zhang, Z. Cao, H. Wen, B. Li, J. Diehl, G. Wang, and D. H. Du, "Data management design for interlaced magnetic recording," in *10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 18)*. Boston, MA: USENIX Association, Jul. 2018. https://www.usenix.org/conference/hotstorage18/presentation/wu
[49] "Open-channel solid state drives specification, revision 2.0," http://lightnvm.io/docs/OCSSD-2_0-20180129.pdf, january 29, 2018.
[50] "Radian memory rms-325 datasheet," http://www.radianmemory.com/edge-card-ssd-rms-325/.
[51] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for ssd performance," in *USENIX 2008 Annual Technical Conference*, ser. ATC '08. Berkeley, CA, USA: USENIX Association, 2008, pp. 57–70. http://dl.acm.org/citation.cfm?id=1404014.1404019
[52] C. Dirik and B. Jacob, "The performance of pc solid-state disks (ssds) as a function of bandwidth, concurrency, device architecture, and system organization," *SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 279–289, Jun. 2009. http://doi.acm.org/10.1145/1555815.1555790
[53] M. Jung and M. Kandemir, "An evaluation of different page allocation strategies on high-speed ssds," in *Proceedings of the 4th USENIX Conference on Hot Topics in Storage and File Systems*, ser. HotStorage'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 9–9. http://dl.acm.org/citation.cfm?id=2342806.2342815
[54] M. Jung, "Exploring parallel data access methods in emerging non-volatile memory systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 3, pp. 746–759, March 2017.
[55] F. Chen, R. Lee, and X. Zhang, "Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, Feb 2011, pp. 266–277.