On the Performance Variation in Modern Storage Stacks

Zhen Cao[1], Vasily Tarasov[2], Hari Prasath Raman[1], Dean Hildebrand[2], and Erez Zadok[1]
[1]*Stony Brook University* *and* [2]*IBM Research—Almaden*

Appears in the proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST'17)

Abstract

Ensuring stable performance for storage stacks is important, especially with the growth in popularity of hosted services where customers expect QoS guarantees. The same requirement arises from benchmarking settings as well. One would expect that repeated, carefully controlled experiments might yield nearly identical performance results—but we found otherwise. We therefore undertook a study to characterize the amount of variability in benchmarking modern storage stacks. In this paper we report on the techniques used and the results of this study. We conducted many experiments using several popular workloads, file systems, and storage devices—and varied many parameters across the entire storage stack. In over 25% of the sampled configurations, we uncovered variations higher than 10% in storage performance between runs. We analyzed these variations and found that there was no single root cause: it often changed with the workload, hardware, or software configuration in the storage stack. In several of those cases we were able to fix the cause of variation and reduce it to acceptable levels. We believe our observations in benchmarking will also shed some light on addressing stability issues in production systems.

1 Introduction

Predictable performance is critical in many modern computer environments. For instance, to achieve good user experience, interactive Web services require stable response time [9, 20, 22]. In cloud environments users pay for computational resources. Therefore, achieving predictable system performance, or at least establishing the limits of performance variation, is of utmost importance for the clients' satisfaction [37, 48]. In a broader sense, humans generally expect repetitive actions to yield the same results and take the same amount of time to complete; conversely, the lack of performance stability, is fairly unsatisfactory to humans.

Performance variation is a complex issue and can arise from nearly every layer in a computer system. At the hardware level, CPU, main memory, buses, and secondary storage can all contribute to overall performance variation [9,22]. At the OS and middleware level, when background daemons and maintenance activities are scheduled, they impact the performance of deployed applications. More performance disruptions come into play when considering distributed systems, as applications on different machines have to compete for heavily shared resources, such as network switches [9].

In this paper we focus on characterizing and analyzing performance variations arising from benchmarking a typical modern storage stack that consists of a file system, a block layer, and storage hardware. Storage stacks have been proven to be a critical contributor to performance variation [18, 33, 40]. Furthermore, among all system components, the storage stack is the cornerstone of data-intensive applications, which become increasingly more important in the big data era [8, 21]. Although our main focus here is reporting and analyzing the variations in benchmarking processes, we believe that our observations pave the way for understanding stability issues in production systems.

Historically, many experienced researchers noticed how workloads, software, hardware, and the environment—even if reportedly "identical"—exhibit different degrees of performance variations in repeated, controlled experiments [7, 9, 11, 22, 23]. We first encountered such variations in experiments using Ext4: multiple runs of the same workload in a carefully controlled environment produced widely different performance results. Over a period of two years of collecting performance data, we later found that such high performance variations were not confined to Ext4. Over 18% of 24,888 different storage stack configurations that we tried exhibited a standard deviation of performance larger than 5% of the mean, and a range value (maximum minus minimum performance, divided by the average) exceeding 9%. In a few extreme cases, standard deviation exceeded 40% even with numerous repeated experiments. The observation that some configurations are more stable than others motivated us to conduct a more detailed study of storage stack performance variation and seek its root causes.

To the best of our knowledge there are no systematic studies of performance variation in storage stacks. Thus, our first goal was to characterize performance variation in different storage stack configurations. However, measuring this for even a single storage stack configuration is time consuming; and measuring all possible stack configurations is time-prohibitive. Even with a small fraction of selected parameters, it could take more than 1.5 years of evaluation time (see Table 1). Therefore, in this study we combined two approaches to reduce the configuration space and therefore the amount of time to run the experiments: (1) we used domain expertise to select

the most relevant parameters, and (2) we applied a Latin Hypercube Sampling (LHS) to the configuration space. Even for the reduced space, it took us over 33 clock days to complete these experiments alone.

We focused on three local file systems (Ext4, XFS, and Btrfs) which are used in many modern local and distributed environments. Using our expertise, we picked several widely used parameters for these file systems (e.g., block size, inode size, journal options). We also varied the Linux I/O scheduler and storage devices, as they can have significant impact on performance. We benchmarked over 100 configurations using different workloads and repeated each experiment 10 times to balance the accuracy of variation measurement with the total time taken to complete these experiments. We then characterized performance variation from several angles: throughput, latency, temporally, spatially, and more. We found that performance variation depends heavily on the specific configuration of the stack. We then further dove into the details, analyzed and explained certain performance variations. For example: we found that unpredictable layouts in Ext4 could cause over 16–19% of performance variation in some cases. We discovered that the magnitude of variation also depends on the observation window size: in one workload, 40% of XFS configurations exhibited higher than 20% variation with a window size of 60s, but almost all of them stabilized when the window size grew to 400s. Finally, we analyzed latency variations from various aspects, and proposed a novel approach for quantifying the impacts of each operation type on overall performance variation.

Our paper has three key contributions: ■ **(1)** To the best of our knowledge, we are the first to provide a detailed characterization of performance variation occurring in benchmarking a typical modern storage stack. We believe our study paves the way towards the better understanding of complex storage stack performance variations, in both experimental and production settings. ■ **(2)** We conducted a comprehensive study of storage stack performance variation. Our analysis includes throughput and latency, and both spatial and temporal variations. ■ **(3)** We offer insights into the root causes of some performance variations, which could help anyone who seeks stable results from benchmarking storage systems, and encourage more follow-up work in understanding variations in production systems.

The rest of the paper is organized as follows. §2 explains background knowledge. §3 describes our experimental methodology. We list our experimental settings in §4. §5 evaluates performance variations from multiple dimensions. §6 covers related work. We conclude and discuss future directions in §7.

2 Background

The storage stack is an essential part of modern computer systems, and critical to the performance of data-intensive applications. Often, the storage stack is the slowest component in a system and thus is one of the main contributors to the overall variability in a system's performance. Characterizing this variation in storage-stack performance is therefore essential for understanding overall system-performance variation.

We first define common performance metrics and notations used in this paper. *Throughput* is defined as the average number of I/O operations completed per second. Here we use a "*Throughput-N*" notation to represent the throughput within the last N seconds of an observation. There are two types of throughput that are used most frequently in our analysis. One is *cumulative* throughput, defined as the throughout from the beginning to the end of the experiment. In this paper, cumulative throughput is the same as *Throughput-800* or *Throughput-2000*, because the complete runtime of a single experiment was either 800 or 2,000 seconds, depending on the workload. The other type is called *instantaneous* throughput, which we denote as *Throughput-10*. Ten seconds is the smallest time unit we collected performance for, in order to avoid too much overhead (explained further in § 4).

Since our goal is to characterize and analyze collected experimental data, we mainly use concepts from *descriptive statistics*. *Statistical variation* is closely related to *central tendency*, which is an estimate of the *center* of a set of values. *Variation* (also called *dispersion* or *variability*), refers to the spread of the values around the central tendency. We considered the most commonly used measure for central tendency—the *mean*: $\bar{x} = \sum_{i=1}^{N} x_i$.

In descriptive statistics, a measure of variation is usually a non-negative real number that is zero if all readings are the same and increases as the measurements become more dispersed. To reasonably compare variations across datasets with different mean values, it is common to normalize the variation by dividing any absolute metric of variation by the mean value. There are several different metrics for variation. In this paper we initially considered two that are most commonly used in descriptive statistical analysis:

- *Relative Standard Deviation (RSD)*: the RSD, (or *Coefficient of Variation (CV)*) is

$$RSD = \frac{\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}}{\bar{x}} \quad (1)$$

- *Relative Range*: this is defined as the difference between the smallest and largest values:

$$RelativeRange = \frac{max(X) - min(X)}{\bar{x}} \quad (2)$$

Because a range uses maximum and minimum values in its calculation, it is more sensitive to outliers. We did not want to exclude or otherwise diminish the significance of performance outliers. We found that even a few long-running I/O operations can substantially worsen actual user experience due to outliers (which are reproducible). Such outliers have real-world impact, especially as more services are offloaded to the cloud, and customers demand QoS guarantees through SLAs. That is one reason why researchers recently have begun to focus on tail latencies [9, 17, 18]. In considering the two metrics above, we felt that the RSD hides some of the magnitudes of these variations—because using square root tends to "compress" the outliers' values. We therefore decided to use the *Relative Range* as our main metric of variation in this work and the rest of this paper.

3 Methodology

Although we encountered storage stack performance variations in past projects, we were especially struck by this issue in our recent experiments on automated recognition of optimal storage configurations. We found that multiple runs of the same workload in a carefully controlled environment could sometimes produce quite unstable results. We later observed that performance variations and their magnitude depend heavily on the specific configuration of the storage stack. Over 18% of 24,888 different storage stack configurations that we evaluated (repeatedly over several workloads) exhibited results with a relative range higher than 9% and relative standard deviation higher than 5%.

Workloads also impact the degree of performance variation significantly. For the same storage stack configuration, experiments with different workloads could produce different magnitudes of variation. For example, we found one Btrfs configuration produces variation with over 40% relative range value on one workload but only 6% for another. All these findings led us to study the characteristics and analyze performance variations in benchmarking various storage stack configurations under multiple workloads. Due to the high complexity of storage stacks, we have to apply certain methodologies in designing and conducting our experiments.

Reducing the parameter space. In this work we focus on evaluating *local* storage stacks (e.g., Ext4, Linux block layer, SSD). This is a useful basis for studying more complex distributed storage systems (e.g., Ceph [46], Lustre [27], GPFS [34], OpenStack Swift [30]). Even a small variation in local storage system performance can result in significant performance fluctuations in large-scale distributed system that builds on it [9, 25, 28].

Despite its simple architecture, the local storage stack has a large number of parameters at every layer, resulting in a vast number of possible configurations. For instance, common parameters for a typical local file system include block size, inode size, journal options, and many more. It is prohibitively time consuming and impractical to evaluate every possible configuration exhaustively. As shown in Table 1, Ext4 has 59 unique parameters that can have anywhere from 2 to numerous allowed values each. If one experiment runs for 15 minutes and we conduct 10 runs for each configuration, it will take us 7.8×10^{33} years of clock time to finish evaluating all Ext4 configurations.

Parameter Space	#Unique Params.	#Unique Configs.	Time (years)
Ext4	59	2.7×10^{37}	7.8×10^{33}
XFS	37	1.4×10^{19}	4.1×10^{15}
Btrfs	54	8.8×10^{26}	2.5×10^{23}
Expert Space	10	1,782	1.52
Sample Space	10	107	33.4 days

Table 1: Comparison for Parameter Spaces. Time is computed by assuming 15 minutes per experimental run, 10 runs per configuration and 3 workloads in total.

Therefore, our first task was to reduce the parameter space for our experiments by carefully selecting the most relevant storage stack parameters. This selection was done in close collaboration with several storage experts that have either contributed to storage stack designs or have spent years tuning storage systems in the field. We experimented with three popular file systems that span a range of designs and features. ■ **(1) Ext4** [12] is a popular file system that inherits a lot of internal structures from Ext3 [6] and FFS [26]) but enhances performance and scalability using extents and delayed allocation. ■ **(2) XFS** [35, 38] was initially designed for SGI's IRIX OS [38] and was later ported to Linux. It has attracted users' attention since the 90s thanks to its high performance on new storage devices and its high scalability regarding large files, large numbers of files, and large directories. XFS uses B+ trees for tracking free extents, indexing directory entries, and keeping track of dynamically allocated inodes. ■ **(3) Btrfs** [5, 31] is a complex file system that has seen extensive development since 2007 [31]. It uses copy-on-write (CoW), allowing efficient snapshots and clones. It has its own LVM and uses B-trees as its main on-disk data structure. These unique features are garnering attention and we expect Btrfs to gain even greater popularity in the future.

For the three file systems above we experimented with the following nine parameters. ■ **(1) Block size**. This is a group of contiguous sectors and is the basic unit of space allocation in a file system. Improper block size selection can reduce file system performance by orders of magnitude [18]. ■ **(2) Inode size**. This is one of the most basic on-disk structures of a file system [3]. It stores the metadata of a given file, such as its size, per-

missions, and the location of its data blocks. The inode is involved in nearly every I/O operation and thus plays a crucial role for performance, especially for metadata-intensive workloads. ■ **(3) Journal mode**. Journaling is the write-ahead logging implemented by file systems for recovery purposes in case of power losses and crashes. In Ext4, three types of journaling modes are supported: *writeback*, *ordered*, and *journal* [13]. The *writeback* mode journals only metadata whereas the *journal* mode provides full data and metadata journaling. In *ordered* mode, Ext4 journals metadata only, but all data is forced directly out to the disk prior to its metadata being committed to the journal. There is a trade-off between file system consistency and performance, as journaling generally adds I/O overhead. In comparison, XFS implements metadata journaling, which is similar to Ext4's *writeback* mode, and there is no need for journaling in Btrfs because of its CoW nature. ■ **(4) Allocation Group (AG) count**. This parameter is specific to XFS which partitions its space into regions called Allocation Groups [38]. Each AG has its own data structures for managing free space and inodes within its boundaries. ■ **(5) Nodatacow** is a Btrfs mount-time option that turns the CoW feature on or off for data blocks. When data CoW is enabled, Btrfs creates a new version of an extent or a page at a newly allocated space [31]. This allows Btrfs to avoid any partial updates in case of a power failure. When data CoW is disabled, partially written blocks are possible on system failures. In Btrfs, *nodatacow* implies *nodatasum* and compression disabled. ■ **(6) Nodatasum** is a Btrfs mount-time option and when specified, it disables checksums for newly created files. Checksums are the primary mechanism used by modern storage systems to preserve data integrity [3], computed using hash functions such as SHA-1 or MD5. ■ **(7) atime Options**. These refer to mount options that control the inode access time. We experimented with *noatime* and *relatime* values. The *noatime* option tells the file system not to update the inode access time when a file data read is made. When *relatime* is set, atime will only be updated when the file's modification time is newer than the access time or atime is older than a defined interval (one day by default). ■ **(8) I/O scheduler**. The I/O Scheduler manages the submission of block I/O operations to storage devices. The choice of I/O scheduler can have a significant impact on the I/O stack performance [4]. We used the *noop*, *deadline*, and *Completely Fair Queuing (CFQ)* I/O schedulers. Briefly explained, the *noop* scheduler inserts all incoming I/O requests into a simple FIFO queue in order of arrival; the *deadline* scheduler associates a deadline with all I/O operations to prevent starvation of requests; and the *CFQ* scheduler try to provide a fair allocation of disk I/O bandwidth for all processes that requests I/O operations. ■ **(9) Storage device**. The underlying storage device plays an important role in nearly every I/O operation. We ran our experiments on three types of devices: two HDDs (SATA vs. SAS) and one (SATA) SSD.

File System	Parameter	Value Range
Ext4	Block Size	1024, 2048, 4096
	Inode Size	128, 512, 2048, 8192
	Journal Mode	data=journal, ordered, writeback
XFS	Block Size	1024, 2048, 4096
	Inode Size	256, 512, 1024, 2048
	AG Count	8, 32, 128, 512
Btrfs	Node Size	4096, 16384, 65536
	Special Options	nodatacow, nodatasum, default
All	atime Options	relatime, noatime
	I/O Scheduler	noop, deadline, cfq
	Storage Devices	HDD (SAS, SATA), SSD (SATA)

Table 2: List of parameters and value ranges.

Table 2 summarizes all parameters and the values used in our experiments.

Latin Hypercube Sampling. Reducing the parameter space to the most relevant parameters based on expert knowledge resulted in 1,782 unique configurations ("Expert Space" in Table 1). However, it would still take more than 1.5 years to complete the evaluation of every configuration in that space. To reduce the space further, we intelligently sampled it using *Latin Hypercube Sampling (LHS)*, a method often used to construct computer experiments in multi-dimensional parameter spaces [19,24]. LHS can help explore a search space and discover unexpected behavior among combinations of parameter values; this suited our needs here. In statistics, a *Latin Square* is defined as a two-dimensional square grid where each row and column have only one sample; *Latin Hypercube* generalizes this to multiple dimensions and ensures that each sample is the only one in the axis-aligned hyper-plane containing it [24]. Using LHS, we were able to sample 107 representative configurations from the Expert Space and complete the evaluation within 34 days of clock time (excluding lengthy analysis time). We believe this approach is a good starting point for a detailed characterization and understanding of performance variation in storage stacks.

4 Experimental Setup and Workloads

This section details our experimental setup, which used a variety of storage devices and workloads.

Hardware. Our experiments were conducted on four identical Dell PowerEdge R710 servers equipped with Intel Xeon quad-core 2.4GHz CPUs. To maintain realistically high ratio of the dataset size to the RAM size and ensure that our experiments produce enough

Workload	Average File Size	# Files		Time(s)		I/O Size			Num. of Threads	R/W Ratio
		Default	Actual	Default	Actual	Read	Write	Append		
Webserver	16KB	1,000	640,000	60	800	WF	-	16KB	100	10:1
Fileserver	128KB	10,000	80,000	60	800	WF	WF	16KB	50	1:2
Mailserver	16KB	1,000	640,000	60	2,000	WF	-	16KB	16	1:1

Table 3: Filebench workload characteristics used in our experiments. WF stands for Whole-File read or write.

I/O, we limited the RAM size on all machines to 4GB. Each server has three types of storage devices installed: (1) 250GB Fujitsu SATA HDD with 5,400 RPM, (2) 147GB Seagate SAS HDD with 15,000 RPM, and (3) 200GB Intel SATA SSD (MLC). This allowed us to evaluate the impact of devices on storage stack performance variation. On each device, we created a full-disk partition. The machines ran an Ubuntu 14.04.1 LTS system, with the kernel upgraded to version 4.4.12. We also updated *e2fsprogs*, *xfsprogs*, and *btrfs-progs* to latest version as of May, 2016.

Workloads. We used *Filebench* [14, 41] v1.5 to generate various workloads in our experiments. In each experiment, if not stated otherwise, we formatted and mounted the storage devices with a file system and then ran Filebench. We used the following three pre-configured Filebench macro-workloads: ■ **(1) Mailserver** emulates the I/O workload of a multi-threaded email server. It generates sequences of I/O operations that mimic the behavior of reading emails (open, read the whole file, and close), composing emails (open/create, append, close, and fsync) and deleting emails. It uses a flat directory structure with all the files in one single directory, and thus exercises the ability of file systems to support large directories and fast lookups. ■ **(2) Fileserver** emulates the I/O workload of a server that hosts users' home directories. Here, each thread represents a user, which performs create, delete, append, read, write, and stat operations on a unique set of files. It exercises both the metadata and data paths of the targeted file system. ■ **(3) Webserver** emulates the I/O workload of a typical static Web server with a high percentage of reads. Files (Web pages) are read sequentially by multiple threads (users); each thread appends to a common log file (Web log). This workload exercises fast lookups, sequential reads of small files and concurrent data and metadata management.

Table 3 shows the detailed settings of these workloads. All are set to Filebench default values, except for the number of files and the running time. As the average file size is an inherent property of a workload and should not be changed [41], the dataset size is determined by the number of files. We increased the number of files such that the dataset size is 10GB—2.5× the machine RAM size. By fixing the dataset size, we normalized the experiments' set-size and run-time, and ensured that the experiments run long enough to produce enough I/O. With these settings, our experiments exercise both in-memory cache and persistent storage devices [42].

We did not perform a separate cache warm-up phase in our experiments because in this study we were interested in performance variation that occurred *both* with cold and warm caches [42]. The default running time for Filebench is set to 60 seconds, which is too short to warm the cache up. We therefore conducted a "calibration" phase to pick a running time that was long enough for the cumulative throughput to stabilize. We ran each workload for up to 2 hours for testing purposes, and finally picked the running time as shown in Table 3. We also let Filebench output the throughput (and other performance metrics) every 10 seconds, to capture and analyze performance variation from a short-term view.

5 Evaluation

In this work we are characterizing and analyzing storage performance variation from a variety of angles. These experiments represent a large amount of data, and therefore, we first present the information with brief explanations, and in subsequent subsections we dive into detailed explanations. §5.1 gives an overview of performance variations found in various storage stack configurations and workloads. §5.2 describes a case study by using Ext4-HDD configurations with the Fileserver workload. §5.3 presents temporal variation results. Here, temporal variations consist of two parts: changes of throughput over time and latency variation.

5.1 Variation at a Glance

We first overview storage stack performance variation and how configurations and workloads impact its magnitude. We designed our experiments by applying the methodology described in §3. We benchmarked configurations from the Sample Space (see Table 1) under three representative workloads from Filebench. The workload characteristics are shown in Table 3. We repeated each experiment 10 times in a carefully-controlled environment in order to get unperturbed measurements.

Figure 1 shows the results as scatter plots broken into the three workloads: *Mailserver* (Figure 1(a)), *Fileserver* (Figure 1(b)), and *Webserver* (1(c)). Each symbol represents one storage stack configuration. We use squares for Ext4, circles for XFS, and triangles for Btrfs. Hollow symbols are SSD configurations, while filled symbols are for HDD. We collected the cumulative throughput for each run. As described in §2, we

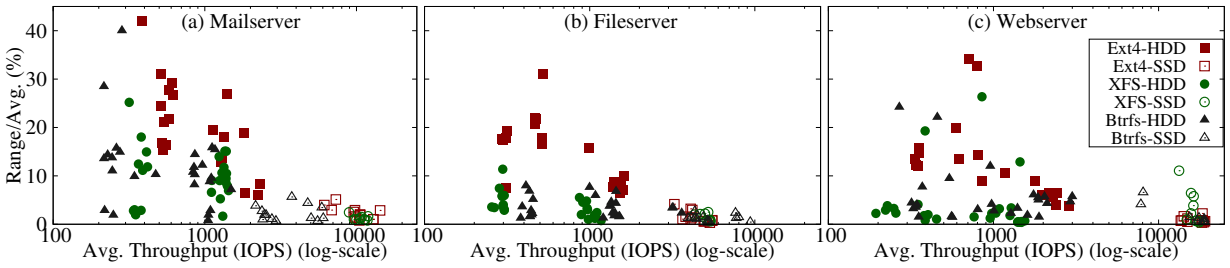


Figure 1: Overview of performance and its variation with different storage stack configurations under three workloads: (a) Maileserver, (b) Fileserver, and (c) Webserver. The X axis represents the mean of throughput over 10 runs; the Y axis shows the relative range of cumulative throughput. Ext4 configurations are represented with squares, XFS with circles, and Btrfs with triangles. HDD configurations are shown with filled symbols, and SSDs with hollow ones.

define the cumulative throughput as the average number of I/O operations completed per second throughout each experiment run. This can also be represented as *Throughput-800* for *Fileserver* and *Webserver*, and *Throughput-2000* for *Mailserver*, as per our notation. In each subfigure, the Y axis represents the relative range of cumulative throughputs across the 10 runs. As explained in §2, here we use the relative range as the measure of variation. A higher relative range value indicates higher degree of variation. The X axis shows the mean cumulative throughput across the runs; higher values indicate better performance. Since performance for SSD configurations is usually much better than HDD configurations, we present the X axis in log_{10} scale.

Figure 1 shows that HDD configurations are generally slower in terms of throughput but show a higher variation, compared with SSDs. For HDDs, throughput varies from 200 to around 2,000 IOPS, and the relative range varies from less than 2% to as high as 42%. Conversely, SSD configurations usually have much higher throughput than HDDs, ranging from 2,000 to 20,000 IOPS depending on the workload. However, most of them exhibit variation less than 5%. The highest range for any SSD configurations we evaluated was 11%.

Ext4 generally exhibited the highest performance variation among the three evaluated file systems. For the *Mailserver* workload, most Ext4-HDD configurations had a relative range higher than 12%, with the highest one being 42%. The *Fileserver* workload was slightly better, with the highest relative range being 31%. Half of the Ext4-HDD configurations show variation higher than 15% and the rest between 5–10%. For *Webserver*, the Ext4-HDD configuration varies between 6–34%. All Ext4-SSD configurations are quite stable in terms of performance variation, with less than 5% relative range.

Btrfs configurations show a moderate level of variation in our evaluation results. For *Mailserver*, two Btrfs-HDD configurations exhibited 40% and 28% ranges of throughput, and all others remained under 15%. Btrfs was quite stable under the *Fileserver* workload, with the highest variation being 8%. The highest relative range value we found for Btrfs-HDD configurations under *Webserver* is 24%, but most of them were below 10%. Similar to Ext4, Btrfs-SSD configurations were also quite stable, with a maximum variation of 7%.

XFS had the least amount of variation among the three file systems, and is fairly stable in most cases, as others have reported before, albeit with respect to tail latencies [18]. For *Mailserver*, the highest variation we found for XFS-HDD configurations was 25%. In comparison, Ext4 was 42% and Btrfs was 40%. Most XFS-HDD configurations show variation smaller than 5% under *Fileserver* and *Webserver* workloads, except for one with 11% for *Fileserver* and three between 12–23% for *Webserver*. Interestingly, however, across all experiments for all three workloads conducted on SSD configurations, the highest variation was observed on one XFS configuration using the *Webserver* workload, which had a relative range value of 11%.

Next, we decided to investigate the effect of workloads on performance variation in storage stacks. Figure 2 compares the results of the same storage stack configurations under three workloads. These results were extracted from the same experiments shown in Figure 1. Although we show here only all Ext4-HDD configurations, we have similar conclusions for other file systems and for SSDs. The bars represent the relative range of 10 repeated runs, and correspond to the left Y1 axis. The average throughput of 10 runs for each configuration is shown as symbols, and corresponds to the right Y2 axis. The X axis consists of configuration details, and is formatted as the six-part tuple ⟨*block size - inode size - journal option - atime option - I/O scheduler - device*⟩. We can see that some configurations remain unstable in all workloads. For example, the configuration *2K-128-writeback-relatime-deadline-SATA* exhibited high performance variation (around 30%) under all three workloads. However, for some configurations, the actual workload played an important role in the magnitude of variation. For example, in the configuration *2K-2K-writeback-noatime-noop-SATA*, the *Mailserver* workload varies the most; but in the configuration *4K-512-*

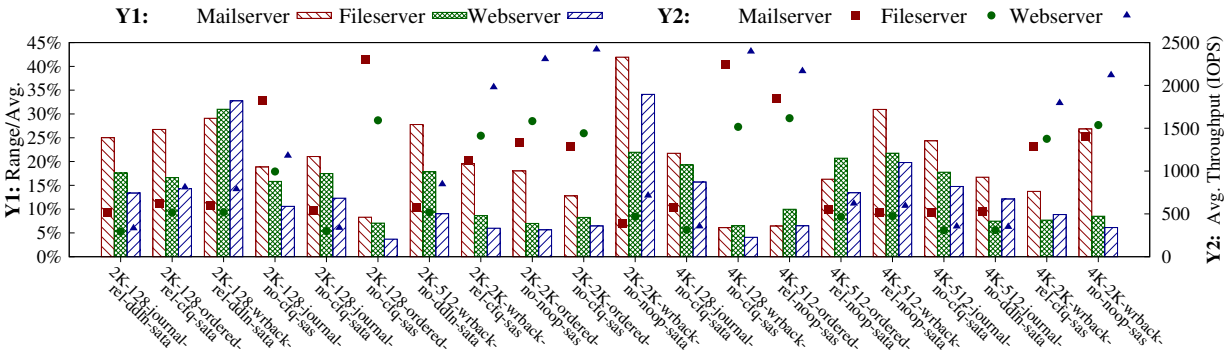


Figure 2: Storage stack performance variation with 20 sampled Ext4-HDD configurations under three workloads. The range is computed among 10 experiment runs, and is represented as bars corresponding to the Y1 (left) axis. The mean of throughput among the 10 runs is shown with symbols (squares, circles, and triangles), and corresponds to the Y2 (right) axis. The X axis represents configurations formatted by ⟨block size - inode size - journal - atime - I/O scheduler - device⟩.

ordered-relatime-noop-SATA, the highest range of performance was seen on *Fileserver*. Finally, configurations with SAS HDD drives tended to have a much lower range variation but higher average throughput than SATA drives.

5.2 Case Study: Ext4

Identifying root causes for performance variation in the storage stack is a challenging task, even in experimental settings. Many components in a modern computer system are not isolated, with complex interactions among components. CPU, main memory, and secondary storage could all contribute to storage variation. Our goal was not to solve the variation problem completely, but to report and explain this problem as thoroughly as we could. We leave to future work to address these root causes from the source code level [44]. At this stage, we concentrated our efforts solely on benchmarking local storage stacks, and tried to reduce the variation to an acceptable level. In this section we describe a case study using four Ext4 configurations as examples. We focused on Ext4-HDD (SATA) here, as this combination of file systems and device types produced the highest variations in our experiments (see Figures 1 and 2).

Figure 3 shows results as two boxplots for the *Fileserver* workload, where each box plots the distribution of throughputs across the 10 runs, with the relative range shown below. The top border represents the 1^{st} quartile, the bottom border the 3^{rd} quartile, and the line in the middle is the median value. Whiskers show the maximum and minimum throughputs. We also plotted one dot for the throughput of each run, overlapping with the boxes but shifted to the right for easier viewing. The X axis represents the relative improvements that we applied based on our successive investigations and uncovering of root causes of performance variation, while the Y axis shows the cumulative throughput for each experiment run. Note that the improvement label is prefixed with a "+" sign, meaning that an additional feature was

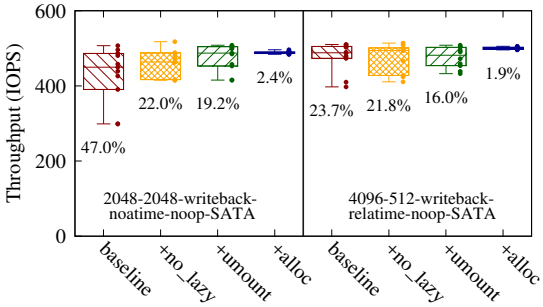


Figure 3: Performance variation for 2 Ext4-HDD configurations with several diagnoses. Each experiment is shown as one box, representing a throughput distribution for 10 identical runs. The top border line of each box marks the 1^{st} quartile; the bottom border marks the 3^{rd} quartile; the line in the middle is the median throughput; and the whiskers mark maximum and minimum values. The dots to the right of each box show the exact throughputs of all 10 runs. The percentage numbers below each box are the relative range values. The bottom label shows configuration details for each figure.

added to the previous configuration, cumulatively. For example, *+umount* actually indicates *baseline* + *no_lazy* + *umount*. We also added labels on the bottom of each subfigure showing the configuration details, formatted as ⟨*block size - inode size - journal option - atime option - I/O scheduler - device*⟩.

After addressing all causes we found, we were able to reduce the relative range of throughput in these configurations from as high as 47% to around 2%. In the rest of this section, we detail each root cause and how we addressed it.

Baseline. The first box for each subfigure in Figure 3 represents our original experiment setting, labeled *baseline*. In this setting, before each experimental run, we format and mount the file system with the targeted configuration. Filebench then creates the dataset on the mounted file system. After the dataset is created, Filebench issues the *sync* command to flush all dirty data and metadata to the underlying device (here, SATA

HDD); Filebench then issues an *echo 3 > /proc/sys/vm/drop_caches* command, to evict non-dirty data and metadata from the page cache. Then, Filebench runs the Fileserver workload for a pre-defined amount of time (see Table 3). For this baseline setting, both Ext4-HDD configurations show high variation in terms of throughput, with range values of 47% (left) and 24% (right).

Lazy initialization. The first contributor to performance variation that we identified in Ext4-HDD configurations is related to the lazy initialization mechanism in Ext4. By default, Ext4 does not immediately initialize the complete inode table. Instead, it gradually initializes it in the background when the created file system is first mounted, using a kernel thread called *ext4lazyinit*. After the initialization is done, the thread is destroyed. This feature speeds up the formatting process significantly, but also causes interference with the running workload. By disabling it during format time, we reduced the range of throughput from 47% to 22% for Configuration *2048-2048-writeback-noatime-noop-SATA*. This improvement is labelled *+no_lazy* in Figure 3.

Sync then umount. In Linux, when *sync* is called, it only guarantees to *schedule* the dirty blocks for writing: there is often a delay until all blocks are actually written to stable media [29, 39]. Therefore, instead of calling *sync*, we *umount* the file system each time after finishing creating the dataset and then *mount* it back, which is labelled as *+umount* in Figure 3. After applying this, both Ext4-HDD configurations exhibited even lower variation than the previous setting (disabling lazy initialization only).

Block allocation and layout. After applying the above improvements, both configurations still exhibited higher than 16% variations, which could be unacceptable in settings that require more predictable performance. This inspired us to try an even more strictly-controlled set of experiments. In the *baseline* experiments, by default we re-created the file system before each run and then Filebench created the dataset. We assumed that this approach would result in identical datasets among different experiment runs. However, block allocation is not a deterministic procedure in Ext4 [18]. Even given the same distribution of file sizes and directory width, and also the same number of files as defined by Filebench, multiple trials of dataset creation on a freshly formatted, clean file system did not guarantee to allocate blocks from the same or even near physical locations on the hard disk. To verify this, instead of re-creating the file system before each run, we first created the file system and the desired dataset on it. We then dumped out the entire partition image using *dd*. Then, before each run of Filebench, we used *dd* to restore the partition using the image, and mounted the file system back. This approach guaranteed an identical block layout for each run.

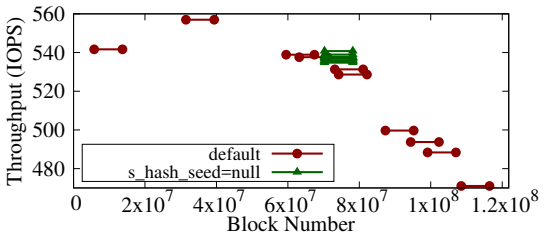


Figure 4: Physical blocks of allocated files in Ext4 under the Fileserver workload. The X axis represents the physical block number of each file in the dataset. Since the Fileserver workload consists of small files, and one extent per file, we use the starting block number for each file here. The Y axis is the final cumulative throughput for each experiment run. Note that the Y axis does not start from 0. Lines marked with solid circles are experiment runs with the default setting; lines with triangles represent experiment runs where we set the field s_hash_seed in Ext4s's superblock to null.

Figure 3 shows these results using *+alloc*. We can see that for both Ext4-HDD configurations, we were able to achieve around 2% of variation, which verified our hypothesis that block allocation and layout play an important role in the performance variation for Ext4-HDD configurations.

Storing the images of file systems using the *dd* command, however, could be too costly in practice, taking hours of clock time. We found a faster method to generate reproducible Ext4 layouts by setting the *s_hash_seed* field in Ext4's superblock to *null* before mounting. Figure 4 shows the distribution of physical blocks for allocated files in two sets of *Fileserver* experiments on Ext4. This workload consists of only small files, resulting in exactly one extent for each file in Ext4, so we used the starting block number (X axis) to represent the corresponding file. The Y axis shows the final cumulative throughput for each experiment run. Here the lines starting and ending with solid circles are 10 runs from the experiment with the full-disk partition. The lines with triangles represent the same experiments, but here we set the *s_hash_seed* field in Ext4's superblock to *null*. We can see that files in each experiment run are allocated into one cluster within a small range of physical block numbers. In most cases, experimental runs with their dataset allocated near the outer tracks of disks, which correspond to smaller block numbers, tend to produce higher throughput. As shown in Figure 4, with the default setting, datasets of 10 runs clustered in 10 different regions of the disk, causing high throughput variation across the runs. By setting the Ext4 superblock parameter *s_hash_seed* to *null*, we can eliminate the non-determinism in block allocation. This parameter determines the group number of top-level directories. By default, *s_hash_seed* is randomly generated during for-

mat time, resulting in distributing top-level directories all across the LBA space. Setting it to *null* forces Ext4 to use the hard-coded default values, and thus the top-level directory in our dataset is allocated on the same position among different experiment runs. As we can see from Figure 4, for the second set of experiments, the ranges of allocated block numbers in all 10 experiment runs were exactly the same. When we set the *s_hash_seed* parameter to *null*, the relative range of throughput dropped from and 16.6% to 1.1%. Therefore, setting this parameter could be useful when users want stable benchmarking results from Ext4.

In addition to the case study we conducted on Ext4-HDD configurations, we also observed similar results for Ext4 on other workloads, as well as for Btrfs. For two of the Btrfs-HDD configurations, we were able to reduce the variation to around 1.2%, by using *dd* to store the partition image. We did not try to apply any improvements on XFS, since most of its configurations were already quite stable (in terms of cumulative throughput) even with the *baseline* setting, as shown in Figure 1.

5.3 Temporal Variation

In Sections 5.1 and 5.2, we mainly presented and analyzed performance variation among repeated runs of the same experiment, and only in terms of throughput. Variation can actually manifest itself in many other ways. We now focus our attention on *temporal* variations in storage stack performance—the variation related to time. §5.3.1 discusses temporal throughput variations and §5.3.2 focuses on latency variations.

5.3.1 Throughput over Time

After finding variations in cumulative throughputs, we set out to investigate whether the performance variation changes over time within single experiment run.

To characterize this, we calculated the throughput within a small time window. As defined in §2, we denote throughput with window size of N seconds as *Throughput-N*. Figure 5 shows the *Throughput-120* value (Y axis) over time (X axis) for Btrfs-HDD, XFS-HDD, and Ext4-HDD configurations using the *Fileserver* workload.

Here we use a window size of 120 seconds, meaning that each throughput value is defined as the average number of I/O operations completed per second with the latest 120 seconds. We also investigated other window sizes, which we discuss later. The three configurations shown here exhibited high variations in the experiments discussed in §5.1. Also, to show the temporal aspect of throughput better, we extended the running time of this experiment set to 2 hours, and we repeated each experiment 10 times. Two lines are plotted connecting the maximum and minimum throughput values among 10 runs. We fill in colors between two lines, this producing a color band: green for Btrfs, red for Ext4, and blue for XFS. The line in the middle of each band is plotted by connecting the average *Throughput-120* value among 10 runs. We observed in Figure 1(b) that for the *Fileserver* workload, Ext4-HDD configurations generally exhibited higher variations than XFS-HDD or Btrfs-HDD configurations in terms of final cumulative throughput. However, when it comes to *Throughput-120* values, Figure 5 leads to some different conclusions. The Ext4-HDD configuration still exhibited high variation in terms of short-term throughout across the 2 hours of experiment time, while the Btrfs-HDD configuration is much more stable. Surprisingly, the XFS-HDD configuration has higher than 30% relative range of *Throughput-120* values for most of the experiment time, while its range for cumulative throughput is around 2%. This suggests that XFS-HDD configurations might exhibit high variations with shorter time windows, but produces more stable results in longer windows. It also indicates that the choice of window sizes matters when discussing performance variations.

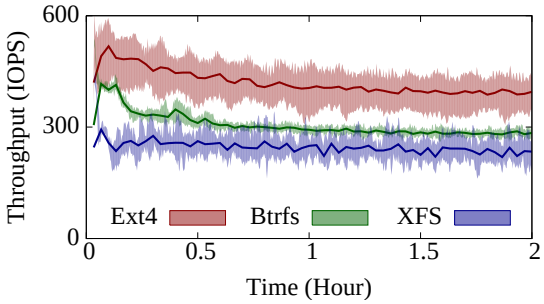


Figure 5: Throughput-120 over time for Btrfs, XFS, and Ext4 HDD configurations under the Fileserver workload. Each configuration was evaluated for 10 runs. Two lines were plotted connecting maximum and minimum throughput values among 10 runs. We fill in colors between two lines, green for Btrfs, red for Ext4, and blue for XFS. We also plotted the average Throughput-120 among 10 runs as a line running through the band. The maximum relative range values of Throughput-120 for Ext4, Btrfs, and XFS are 43%, 23%, and 65%, while the minimum values are 14%, 2%, and 7%, respectively.

We can see from the three average lines in Figure 5 that performance variation exists even within one single run—the short-term throughput varies as the experiment proceeds. For most experiments, no matter what the file system type is, performance starts slow and climbs up quickly in the beginning phase of experiments. This is because initially the application is reading cold data and metadata from physical devices into the caches; once cached, performance improves. Also, for some period of time, dirty data is kept in the cache and not yet flushed to stable media, delaying any impending slow writes. After an initial peak, performance begins to drop rapidly and then declines steadily. This is because the read per-

formance already reached its peak and cached dirty data begins to be flushed out to slower media. Around several minutes in, performance begins to stabilize, as we see the throughput lines flatten.

The unexpected difference in variations for short-term and cumulative throughput of XFS-HDD configurations lead us to investigate the effects of the time window size on performance variations. We calculated the relative range of throughput with different window sizes for all configurations within each file system type. We present the CDFs of these range values in Figure 6. For example, we conducted experiments on 39 Btrfs configurations. With a window size of 60 seconds and total running time of 800 seconds, the corresponding CDF for Btrfs is based on $39 \times \frac{800}{60} = 507$ relative range values. We can see that Ext4's unstable configurations are largely unaffected by the window size. Even with *Throughput-400*, around 20% of Ext4 configurations produce higher than 20% variation in terms of throughput. Conversely, the range values for Btrfs and XFS are more sensitive to the choice of window size. For XFS, around 40% of the relative range values for *Throughput-60* are higher than 20%, whereas for *Throughput-400*, nearly all XFS values fall below 20%. This aligns with our early conclusions in §5.1 that XFS configurations are relatively stable in terms of cumulative throughput, which is indeed calculated based on a window size of 800 seconds; whereas XFS showed the worst relative range for *Throughput-60*, it stabilized quickly with widening window sizes, eventually beating Ext4 and Btrfs.

All the above observations are based on the throughput within a certain window size. Another approach is to characterize the *instant throughput* within an even shorter period of time. Figure 7 shows the instantaneous throughput over time for various configurations under the *Fileserver* workload. We collected and calculated the throughput every 10 seconds. Therefore we define instantaneous throughput as the average number of I/O operations completed in the past 10 seconds. This is actually *Throughput-10* in our notation. We normalize this by dividing each value by the maximum instantaneous throughput value for each run, to compare the variation across multiple experimental runs. The X axis still shows the running time.

We picked one illustrative experiment run for each configuration (Ext4-HDD, XFS-HDD, Btrfs-HDD, and Ext4-SSD). We can see from Figure 7 that for all configurations, instantaneous performance fluctuated a lot throughout the experiment. For all three HDD configurations, the variation is even higher than 80% in the first 100 seconds. The magnitude for variation reduces later in the experiments, but stays around 50%.

The throughput spikes occur nearly every 30 seconds, which could be an indicator that the performance variation in storage stacks is affected by some cyclic activity (e.g., kernel flusher thread frequency). For SSD configurations, the same up-and-down pattern exists, although its magnitude is much smaller than for HDD configurations, at only around 10%. This also confirms our findings from §5.1 that SSDs generally exhibit more stable behavior than HDDs.

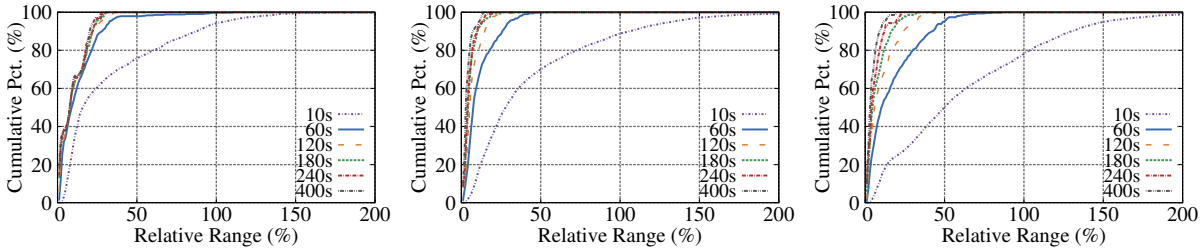
5.3.2 Latency Variation

Another aspect of performance variation is latency, defined as the time taken for each I/O request to complete. Much work has been done in analyzing and taming long-tail latency in networked systems [20, 22] (where 99.9^{th} percentile latency is orders of magnitude worse than the median), and also in local storage systems [18]. Throughout our experiments, we found out that long-tail latency is not the only form of latency variation; there are other factors that can impact the latency distribution for I/O operations.

A *Cumulative Distribution Function (CDF)* is a common approach to present latency distribution. Figure 8(a) shows the latency CDFs for 6 I/O operations of one Ext4-HDD configuration under the *Fileserver* workload. The X axis represents the latency in log_{10} scale, while the Y axis is the cumulative percentage. We can see that for any one experimental run, operations can have quite different latency distribution. The latencies for *read*, *write*, and *create* form two clusters. For example, about 20% of the *read* operation has less than 0.1ms latency while the other 80% falls between 100ms and 4s. Conversely, the majority of *stat*, *open*, and *delete* operations have latencies less than 0.1ms.

The I/O operation type is not the only factor that impacts the latency distribution. Figure 8(b) presents 10 CDFs for *create* from 10 repeated runs of the same experiment. We can see for the 60^{th} percentile, the latency can vary from less than 0.1ms to over 100ms.

Different I/O operations and their latencies impact the overall workload throughput to a different extent. With the empirical data that we collected—per-operation latency distributions and throughput—we were able to discover correlations between the speed of individual operations and the throughput. We first defined a metric to quantify the difference between two latency distributions. We chose to use the Kolmogorov-Smirnov test (K-S test), which is commonly used in statistics to determine if two datasets differ significantly [43]. For two distributions (or discrete dataset), the K-S test uses the maximum vertical deviation between them as the distance. We further define the range for a set of latency distributions as the maximum distance between any two latency CDFs. This approach allows us to use only one number to represent the latency variation, as with throughput. For each operation type, we calculated its



(a) Ext4 (b) Btrfs (c) XFS

Figure 6: CDFs for relative range of throughput under Fileserver workload with different window sizes. For window size N, we calculated the relative range values of throughput for all configurations within each file system type, and then plotted the corresponding CDF.

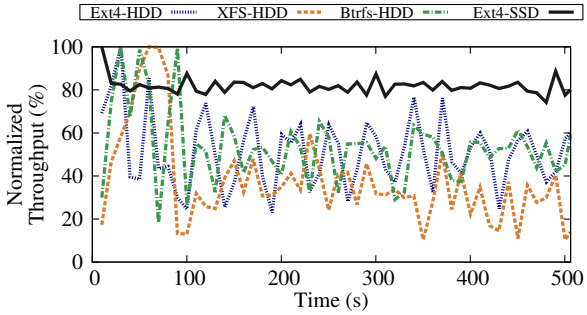


Figure 7: Normalized instantaneous throughput (Throughput-10) over time for experiments with various workloads, file systems, and devices. The Y axis shows the normalized values divided by the maximum instantaneous throughput through the experiment. Only the first 500s are presented for brevity.

range of latency variation for each configuration under all three workloads. We then computed the Pearson Correlation Coefficient (PCC) between the relative range of throughput and the range of latency variation.

Figure 9 shows our correlation results. The PCC value for any two datasets is always between [-1,+1], where +1 means total positive correlation, 0 indicates no correlation, and –1 means total negative correlation. Generally, any two datasets with PCC values higher than 0.7 are considered to have a strong positive correlation [32], which we show in Figure 9 with a horizontal dashed red line. The Y axis represents the PCC value while the X axis is the label for each operation. We separate workloads with vertical solid lines. As most SSD configurations are quite stable in terms of performance, we only considered HDD configurations here. For Ext4 configurations, *open* and *read* have the highest PCC values on both *Mailserver* and *Webserver* workloads; however, on *Fileserver*, *open* and *stat* have the strongest correlation. These operations could possibly be the main contributors to performance variation on Ext4-HDD configurations under each workload; such operations would represent the first ones one might tackle in the future to help stabilize Ext4's performance on HDD. In comparison, *write* has a PCC value of only around 0.2, which indicates that it may not contribute much to the performance variation. Most operations show PCC values larger than 0.4, which suggest weak correlation. This is possibly because I/O operations are not completely independent with each other in storage systems.

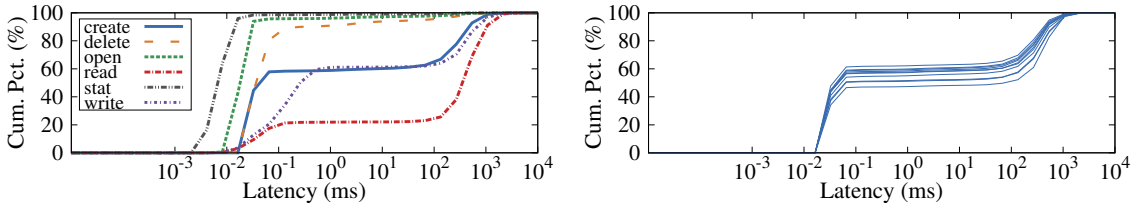
For the same workload, different file systems exhibit different correlations. For example, under the *Webserver* workload, Ext4 show strong correlation on both *read* and *open*; but for XFS, *read* shows a stronger correlation than *open* and *write*. For Btrfs, no operation had a strong correlation with the range of throughput, with only *read* showing a moderate level of correlation.

Although such correlations do not always imply direct causality, we still feel that this correlation analysis sheds light on how each operation type might contribute to the overall performance variation in storage stacks.

6 Related Work

To the best of our knowledge, there are no systematic studies of performance variation of storage stacks. Most previous work focuses on long-tail I/O latencies. Tarasov et al. [40] observed that file system performance could be sensitive to even small changes in running workloads. Arpaci-Dusseau [2] proposed an I/O programming environment to cope with performance variations in clustered platforms. Worn-out SSDs exhibit high latency variations [10]. Hao et al. [16] studied device-level performance stability, for HDDs and SSDs.

For long-tail latencies of file systems, He et al. [18] developed Chopper, a tool to explore a large input space of file system parameters and find behaviors that lead to performance problems; they analyzed long-tail latencies relating to block allocation in Ext4. In comparison, our paper's goal is broader: a detailed characterization and analysis of several aspects of storage stack performance variation, including devices, block layer, and the file systems. We studied the variation in terms of both throughput and latency, and both spatially and temporally. Tail latencies are common in network or cloud services [9, 22]: several tried to characterize and mitigate their effects [17, 20, 37, 48], as well as exploit them to save data center energy [45]. Li et al. [22] characterized tail latencies for networked services from the hard-



(a) CDFs of operations within one single experiment run (b) CDFs of *create* operation among repeated experiment runs

Figure 8: Latency CDF of one Ext4-HDD configuration under Fileserver workload.

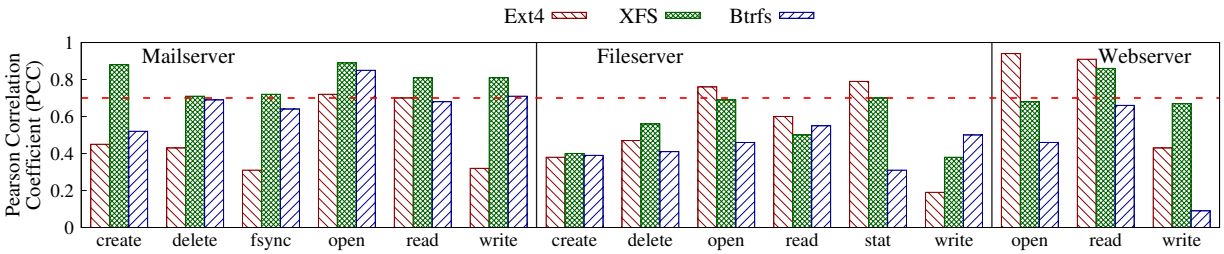


Figure 9: Pearson Correlation Coefficient (PCC) between throughput range and operation types, for three workloads and three file systems. The horizontal dashed red line at Y=0.7 marks the point above which a strong correlation is often considered to exist.

ware, OS, and application-level sources. Dean and Barroso [9] pointed out that small performance variations could affect a significant fraction of requests in large-scale distributed systems, and can arise from various sources; they suggested that eliminating all of them in large-scale systems is impractical. We believe there are possibly many sources of performance variation in storage systems, and we hope this work paves the way for discovering and addressing their impacts.

7 Conclusion

In this work we provided the first systematic study on performance variation in benchmarking a modern storage stack. We showed that variation is common in storage stacks, although its magnitude depends heavily on specific configurations and workloads. Our analysis revealed that block allocation is a major cause of performance variation in Ext4-HDD configurations. From the temporal view, the magnitude of throughput variation also depends on the window size and changes over time. Latency distribution for the same operation type could also vary even over repeated runs of the same experiment. We quantified the correlation between performance and latency variations using a novel approach. Although most of our observations are made in experimental settings, we believe they are a useful step towards a thorough understanding of stability issues in storage stacks of production systems. In conclusion, we list three best practices for people either benchmarking storage systems or dealing with performance variations in real systems. The goal here is not to "teach," but rather provide some guidelines to the best of our knowledge. ■ **(1)** Performance variation is a complex issue, and could be caused and affected by various factors: the file system, configurations of the storage system, the running workload, or even the time window for quantifying the performance. ■ **(2)** Non-linearity is inherent in complex storage systems. It could lead to large differences in results, even in well-controlled experiments; conclusions drawn from these could be misleading or even wrong. ■ **(3)** Disable all lazy initialization and any background activities, if any, while formatting, mounting, and experimenting on file systems.

Future Work. We believe that more work still needs to be done to more fully understand the causes of different types of variation and especially to address them. All experiments in this paper were conducted on freshly-formatted file systems, and thus we only focused on performance variations in such systems. We did not analyze aged file systems, a subject of our future work. We plan to expand our parameter search space (e.g., compression options in Btrfs [31]). Alas, Filebench currently creates files by filling them with 0s, so first we have to make Filebench output data with controlled compression ratios. We plan to use other benchmarking tools such as SPEC SFS 2014 [36] which comes with several pre-configured and realistic workloads. We plan to expand the study to new types of devices such as PCM [15, 47] and SMRs [1], which have their own complex behavior such as worse tail latencies due to internal garbage collection [9, 10]. In the meanwhile, we will tackle other storage layers (LVM, RAID) and networked/distributed file systems. Finally, We plan to make all of our datasets and sources public. This includes not only the data from this work, but also a much larger dataset we continue to collect (now over two years).

Acknowledgments

We thank the anonymous FAST reviewers and our shepherd Remzi Arpaci-Dusseau for their valuable comments; and to Ted Ts'o for his assitance in understanding Ext4's behavior. This work was made possible in part thanks to Dell-EMC, NetApp, and IBM support; NSF awards CNS-1251137, CNS-1302246, CNS-1305360, and CNS-1622832; and ONR award 12055763.

References

[1] Abutalib Aghayev, Mansour Shafaei, and Peter Desnoyers. Skylight—a window on shingled disk operation. *Trans. Storage*, 11(4):16:1–16:28, October 2015.

[2] R. H. Arpaci-Dusseau, E. Anderson, N. T., D. E. Culler, J. M. Hellerstein, D. Patterson, and K. Yelick. Cluster I/O with river: making the fast case common. In *Workshop on Input/Output in Parallel and Distributed Systems*, pages 10–22, Atlanta, GA, May 1999.

[3] Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books, 0.91 edition, May 2015.

[4] D. Boutcher and A. Chandra. Does virtualization make disk scheduling passé? In *Proceedings of the 1st USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '09)*, October 2009.

[5] BTRFS. *http://btrfs.wiki.kernel.org/*.

[6] M. Cao, T. Y. Ts'o, B. Pulavarty, S. Bhattacharya, A. Dilger, and A. Tomas. State of the Art: Where we are with the Ext3 filesystem. In *Proceedings of the Linux Symposium*, Ottawa, ON, Canada, July 2005.

[7] Kevin K. Chang, Abhijith Kashyap, Hasan Hassan, Saugata Ghose, Kevin Hsieh, Donghyuk Lee, Tianshi Li, Gennady Pekhimenko, Samira Khan, and Onur Mutlu. Understanding latency variation in modern DRAM chips: Experimental characterization, analysis, and optimization. In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*, SIGMETRICS'16, pages 323–336, New York, NY, USA, 2016. ACM.

[8] Min Chen, Shiwen Mao, and Yunhao Liu. Big data: A survey. *Mobile Networks and Applications*, 19(2):171–209, 2014.

[9] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Commun. ACM*, 56(2):74–80, February 2013.

[10] Peter Desnoyers. Empirical evaluation of nand flash memory performance. In *HotStorage '09: Proceedings of the 1st Workshop on Hot Topics in Storage*. ACM, 2009.

[11] Nosayba El-Sayed, Ioan A. Stefanovici, George Amvrosiadis, Andy A. Hwang, and Bianca Schroeder. Temperature management in data centers: Why some (might) like it hot. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS'12, pages 163–174, New York, NY, USA, 2012. ACM.

[12] Ext4. *http://ext4.wiki.kernel.org/*.

[13] Ext4 Documentation. *https://www.kernel.org/doc/Documentation/filesystems/ext4.txt*.

[14] Filebench, 2016. *https://github.com/filebench/filebench/wiki*.

[15] H. Kim and S. Seshadri and C. L. Dickey and L. Chiu. Evaluating Phase Change Memory for Enterprise Storage Systems: A Study of Caching and Tiering Approaches. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies*, pages 33–45, Berkeley, CA, 2014. USENIX.

[16] Mingzhe Hao, Gokul Soundararajan, Deepak Kenchammana-Hosekote, Andrew A Chien, and Haryadi S Gunawi. The tail at store: a revelation from millions of hours of disk and ssd deployments. In *14th USENIX Conference on File and Storage Technologies (FAST 16)*, pages 263–276, 2016.

[17] Md E. Haque, Yong hun Eom, Yuxiong He, Sameh Elnikety, Ricardo Bianchini, and Kathryn S. McKinley. Few-to-many: Incremental parallelism for reducing tail latency in interactive services. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS'15, pages 161–175, New York, NY, USA, 2015. ACM.

[18] Jun He, Duy Nguyen, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Reducing file system tail latencies with Chopper. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies*, FAST'15, pages 119–133, Berkeley, CA, USA, 2015. USENIX Association.

[19] Ronald L Iman, Jon C Helton, James E Campbell, et al. An approach to sensitivity analysis of computer models, part 1. introduction, input variable selection and preliminary variable assessment. *Journal of quality technology*, 13(3):174–183, 1981.

[20] Myeongjae Jeon, Saehoon Kim, Seung-won Hwang, Yuxiong He, Sameh Elnikety, Alan L. Cox, and Scott Rixner. Predictive parallelization: Taming tail latencies in web search. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR'14, pages 253–262, New York, NY, USA, 2014. ACM.

[21] Karthik Kambatla, Giorgos Kollias, Vipin Kumar, and Ananth Grama. Trends in big data analytics. *Journal of Parallel and Distributed Computing*, 74(7):2561–2573, 2014. Special Issue on Perspectives on Parallel and Distributed Processing.

[22] Jialin Li, Naveen Kr. Sharma, Dan R. K. Ports, and Steven D. Gribble. Tales of the tail: Hardware, os, and application-level sources of tail latency. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC'14, pages 9:1–9:14, New York, NY, USA, 2014. ACM.

[23] Chieh-Jan Mike Liang, Jie Liu, Liqian Luo, Andreas Terzis, and Feng Zhao. RACNet: A high-fidelity data center sensing network. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys'09, pages 15–28, New York, NY, USA, 2009. ACM.

[24] W. J. Conover M. D. McKay, R. J. Beckman. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979.

[25] Pradipta De Vijay Mann and Umang Mittaly. Handling OS jitter on multicore multithreaded systems. In *Parallel & Distributed Processing Symposium (IPDPS), 2009 IEEE International*, IPDPS'09, pages 1–12. IEEE, 2009.

[26] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry. A fast file system for UNIX. *ACM Transactions on Computer Systems*, 2(3):181–197, August 1984.

[27] Sun Microsystems. Lustre File System: High-Performance Storage Architecture and Scalable Cluster File System White Paper. *www.sun.com/servers/hpc/docs/lustrefilesystem_wp.pdun.com/servers/hpc/docs/lustrefilesystem_wp.pdf*, December 2007.

[28] Alessandro Morari, Roberto Gioiosa, Robert W Wisniewski, Francisco J Cazorla, and Mateo Valero. A quantitative analysis of OS noise. In *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*, IPDPS'11, pages 852–863. IEEE, 2011.

[29] E. B. Nightingale, K. Veeraraghavan, P. M. Chen, and J. Flinn. Rethink the sync. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI 2006)*, pages 1–14, Seattle, WA, November 2006. ACM SIGOPS.

[30] OpenStack Swift. *http://docs.openstack.org/developer/swift/*.

[31] Ohad Rodeh, Josef Bacik, and Chris Mason. BTRFS: The linux b-tree filesystem. *Trans. Storage*, 9(3):9:1–9:32, August 2013.

[32] Richard P Runyon, Kay A Coleman, and David J Pittenger. *Fundamentals of behavioral statistics* . McGraw-Hill, 2000.

[33] Ricardo Santana, Raju Rangaswami, Vasily Tarasov, and Dean Hildebrand. A fast and slippery slope for file systems. In *Proceedings of the 3rd Workshop on Interactions of NVM/FLASH with Operating Systems and Workloads*, INFLOW '15, pages 5:1–5:8, New York, NY, USA, 2015. ACM.

[34] F. Schmuck and R. Haskin. GPFS: A shared-disk file system for large computing clusters. In *Proceedings of the First USENIX Conference on File and Storage Technologies (FAST '02)*, pages 231–244, Monterey, CA, January 2002. USENIX Association.

[35] SGI. XFS Filesystem Structure. *http://oss.sgi.com/projects/xfs/papers/xfs_filesystem_structure.pdf*.

[36] SPEC SFS® 2014. *https://www.spec.org/sfs2014/*.

[37] Lalith Suresh, Marco Canini, Stefan Schmid, and Anja Feldmann. C3: Cutting tail latency in cloud data stores via adaptive replica selection. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI'15, pages 513–527, Berkeley, CA, USA, 2015. USENIX Association.

[38] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck. Scalability in the XFS file system. In *Proceedings of the Annual USENIX Technical Conference*, pages 1–14, San Diego, CA, January 1996.

[39] sync(8) - Linux man page. *https://linux.die.net/man/8/sync*.

[40] V. Tarasov, S. Bhanage, E. Zadok, and M. Seltzer. Benchmarking File System Benchmarking: It *IS* Rocket Science. In *Proceedings of HotOS XIII:The 13th USENIX Workshop on Hot Topics in Operating Systems*, Napa, CA, May 2011.

[41] V. Tarasov, E. Zadok, and S. Shepler. Filebench: A flexible framework for file system benchmarking. *;login: The USENIX Magazine*, 41(1):6–12, March 2016.

[42] Vasily Tarasov, Zhen Cao, Ming Chen, and Erez Zadok. The dos and don'ts of file system benchmarking. *FreeBSD Journal*, January/February, 2016.

[43] Olivier Thas. *Comparing distributions*. Springer, 2010.

[44] A. Traeger, I. Deras, and E. Zadok. DARC: Dynamic analysis of root causes of latency distributions. In *Proceedings of the 2008 International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2008)*, pages 277–288, Annapolis, MD, June 2008. ACM.

[45] Balajee Vamanan, Hamza Bin Sohail, Jahangir Hasan, and T. N. Vijaykumar. TimeTrader: Exploiting latency tail to save datacenter energy for online search. In *Proceedings of the 48th International Symposium on Microarchitecture*, MICRO'48, pages 585–597, New York, NY, USA, 2015. ACM.

[46] S. Weil, S. Brandt, E. Miller, D. Long, and C. Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI 2006)*, pages 307–320, Seattle, WA, November 2006. ACM SIGOPS.

[47] H-S Philip Wong, Simone Raoux, SangBum Kim, Jiale Liang, John P Reifenberg, Bipin Rajendran, Mehdi Asheghi, and Kenneth E Goodson. Phase change memory. *Proceedings of the IEEE*, 98(12):2201–2227, Dec 2010.

[48] Yunjing Xu, Zachary Musgrave, Brian Noble, and Michael Bailey. Bobtail: Avoiding long tails in the cloud. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, NSDI'13, pages 329–342, Berkeley, CA, USA, 2013. USENIX Association.