A Fast and Slippery Slope for File Systems

Ricardo Santana Raju Rangaswami
Florida International University
rsant144@fiu.edu raju@cs.fiu.edu

Vasily Tarasov Dean Hildebrand
IBM Research—Almaden
vtarasov@us.ibm.com
dhildeb@us.ibm.com

Abstract

There is a vast number and variety of file systems currently available, each optimizing for an ever growing number of storage devices and workloads. Users have an unprecedented, and somewhat overwhelming, number of data management options. At the same time, the fastest storage devices are only getting faster, and it is unclear on how well the existing file systems will adapt. Using emulation techniques, we evaluate five popular Linux file systems across a range of storage device latencies typical to low-end hard drives, latest high-performance persistent memory block devices, and in between. Our findings are often surprising. Depending on the workload, we find that some file systems can clearly scale with faster storage devices much better than others. Further, as storage device latency decreases, we find unexpected performance inversions across file systems. Finally, file system scalability in the higher device latency range is not representative of scalability in the lower, sub-millisecond, latency range. We then focus on Nilfs2 as an especially alarming example of an unexpectedly poor scalability and present detailed instructions for identifying bottlenecks in the I/O stack.

Categories and Subject Descriptors D.4.2 [*Storage Management*]: Secondary storage; D.4.3 [*File Systems Management*]: File organization; D.4.8 [*Performance*]: Measurements; D.4.8 [*Performance*]: Modeling and prediction

General Terms Measurement, Performance, Design

Keywords File systems, high-speed devices

1. Introduction

The number and variety of available file systems can give users decision paralysis. To complicate matters, storage devices are getting faster and faster, and it is not clear that a file system that works well on todays hardware will continue to work well on the next generation hardware. Further, the workload type, of which there is no shortage, also has a large impact on file system selection. Due to the broad proliferation of file systems, this problem impacts numerous system domains, ranging from servers, laptops, embedded devices, mobile phones, and virtual machines to large distributed file systems in the enterprise.

The problem is further complicated by the fact that we are about to enter an era where storage latency across device types varies by as much as four orders of magnitude. While disk drives offer multi-millisecond latencies, flash-based solid-state drives provide latencies in sub-millisecond range and the newer persistent memory based storage devices offer even lower latencies of 200-300ns [11]. These diverse device characteristics are in many cases hidden from the file system. For example, VMs and storage virtualization abstract the true nature of the underlying hardware, but still allow users to request a block device with specific latency bounds [15]. Given these developments, file systems should be capable of performing well on storage devices with arbitrary performance characteristics.

Conventionally, the evaluation of file systems has focused on a single type of storage hardware. Without a comprehensive study that can keep up with the changing landscape of available file systems and storage hardware, users often make implicit generalizations of file system performance for similar or dissimilar hardware and workloads. Given the rapid pace of change, there is a need to reevaluate this conventional thinking, and be much more comprehensive with respect to how file systems are evaluated.

In this paper, we motivate the evaluation of file system performance across both workloads and storage device speeds. Using an instrumented block device layer that is capable of emulating both multi- and sub-millisecond latencies, we evaluate five Linux file systems—Ext4, XFS, BTRFS, Nilfs2, and F2FS—using several common workloads. Our initial findings are both unexpected and counter-intuitive and motivate substantial follow-on work to understand in greater detail how popular file systems perform

INFLOW'15, October 4–7, 2015, Monterey, CA.

http://dx.doi.org/10.1145/http://dx.doi.org/10.1145/2819001.2819003

across a variety of device characteristics in both end-user and enterprise settings.

We make several specific observations.

Observation 1. *File systems are not created equal for all storage speeds; not only do some file systems scale better than others as storage latencies decrease, their relative scaling capabilities are highly workload-sensitive.*

Observation 2. *With some workloads unexpected performance inversions occur; file systems that perform faster than other file systems at higher latencies perform slower at lower latencies, and vice versa.*

Observation 3. *File system performance models built on the reasonable expectation that performance scales with the storage device speed are arbitrarily inaccurate, especially in the lower, sub-millisecond, latency range; file system scaling properties are complex and require further investigation.*

Understanding the root causes of unexpected behavior is critical to file system evolution for supporting next generation storage devices efficiently. For example, our experiments revealed that for some workloads Nilfs2 throughput remains almost flat across high- and low-latency devices. We used Nilfs2 as a case study to develop a general guide for identifying bottlenecks in a file system and an I/O stack. We uncovered a high level of metadata contention in Nilfs2 that fetters its ability to scale as device latency decreases. Ultimately, our work motivates revisiting file system designs for the new era of diverse storage device characteristics.

2. Goals and Models

File systems remain the most common abstraction through which applications access underlying storage. Block-based file systems translate file-level operations into block-level accesses to the storage device. While some recent proposals eliminate the block abstraction for fast persistent memory based devices [10–12, 20], other proposals have advocated the opposite [7, 9] for backward compatibility. We believe that the block abstraction will remain a significant building block in the foreseeable future.

The main goal of our study therefore is to gain initial insights in how file systems perform across a range of block device latencies. Specifically, we are interested in understanding file system scaling characteristics with faster storage devices. We believe this work can provide a direct benefit to the file system research community by identifying bottlenecks across diverse storage devices and workloads, as well as indicating potential performance improvements when users upgrade their storage. However, our methodology is *not* designed to gain a precise understanding of file system behaviors. Instead, the goal is to identify general trends in file system performance to further research and engineering efforts. To this end, we introduce a user-centric performance model to guide our efforts.

User Performance Expectation Model (UPEM) represents file system performance as expected by users. Let's assume that the average latency of a storage device and I/O software stack are l_{dev} and l_{sw}, respectively. The value of l_{sw} is constant for a specific system setup, file system, and workload, while l_{dev} is a device-specific characteristic. The total average latency of a single I/O operation is proportional to $(l_{dev} + l_{sw})$. Then file system throughput is inversely proportional to the latency:

$$Throughput(l_{dev}) = \frac{C}{l_{dev} + l_{sw}}$$

where C is a coefficient of proportionality specific to the system setup, file system, and workload.

If the file system throughput is known for two l_{dev} latencies, then two constants C and l_{sw} can be computed from the system of equations. Our experiments evaluated performance for device latencies between 0–10ms to represent both currently available and future storage devices. To calculate C and l_{sw} and thereby calibrate this model, we picked the slowest latency of 10ms and the middle-range latency of 5ms. We believe that providing the model with information of half of the range should provide reasonable model accuracy. We use the rest of the latencies (0–5ms) to compare model predictions against experimental results.

It is important to distinguish UPEM, whose goal is to capture user performance expectations (i.e., "My file system should go faster with a faster storage device"), from a real "file system performance model" that describes the specifics of a file system's design and implementation.

3. Testing Methodology

Evaluating the performance of multiple file systems across multiple workloads and devices is a challenging but feasible task. Our approach is both systematic and pragmatic.

Hardware and Operating System. All experiments were run on identical IBM System x3650 M4 servers equipped with two 8-core Intel Xeon CPUs and 96GB of memory, running Red Hat Enterprise Linux 7.0 (RHEL7). We encountered issues with BTRFS on RHEL7's 3.10 Linux kernel, so we switched to the vanilla 3.14 kernel for all experiments.

File Systems. We chose five local file systems available in the Linux kernel: 1) Ext4, 2) XFS, 3) BTRFS, Nilfs2, and 5) F2FS. These file systems have significantly different internal architectures and we consider them representative of recent developments in file system design for both conventional and newer block storage devices. Ext4 [3] is a traditional FFS-like file system that uses inode tables, bitmaps, and indirect blocks, and implements journaling, extents, and delayed allocation. XFS [19] uses B+ trees instead of linear structures for data management. Designed as a highly scalable local file system, it is the default file system in RHEL7. BTRFS [16] combines a volume manager and file system and uses COW B-trees for storing data and metadata.

Nilfs2 [5] is a log-structured file system (LFS [17]) implementation for Linux. F2FS [13] was designed to provide optimal performance for block-based NAND devices. As with Nilfs2, it uses a log-structured approach to avoid expensive random writes. F2FS also implements a hot/cold data separation scheme and NAND-friendly index block updates. We used default mount and format options for all file systems.

Emulating Devices. Real physical devices can exhibit a whole spectrum of performance irregularities depending on the technology used, specific device implementation, and workload. For tractability, we eschew low-level device-specific performance features and emulate devices that present a constant latency per I/O operation independent of type and size. In the cloud storage era, the practice of exporting virtual block devices to users with guaranteed latency *service level agreement* (SLA) will become commonplace [15]. The device emulation approach used in our study is particularly well-suited for evaluating file systems when using such opaque, but performant, virtual devices. However, as demonstrated in Section 4, we also found a good match between performance of our emulator and real physical devices, such as SSDs.

We emulated devices with configurable latencies using dm-delay [2]. The dm-delay virtual block device delays every I/O request to the underlying storage for a configurable number of milliseconds. We deployed a RAM disk underneath dm-delay to emulate low latency devices. The RAM disk latency was at most $5\mu s$ when measured from user space and represents the true baseline latency in our experiments.

The original dm-delay implementation supports a minimum delay granularity of 1ms. To emulate devices with sub-millisecond delays we modified dm-delay using high-resolution Linux timers that provide $1ns$ granularity. After a request arrival from the upper layer, dm-delay sets up a high-resolution timer to trigger request processing at an appropriate later time. After the timer is triggered the request is passed to the lower layer almost instantly with a slight extra delay caused by the task scheduler. We posted our patches to the device-mapper's developers mailing list [1]. In this study we measured file system performance for delays incremented in 1ms steps between 1ms and 10ms and in $100\mu s$ steps between 0 and 1ms ($1000\mu s$).

Dm-delay allows to specify different delays for read and write operations but the delay remains the same across all I/O sizes. In this study, we set both read and write delays to identical values. There is also no additional queuing delay in dm-delay, i.e., all requests are executed practically in parallel. We believe that this setup is sufficient to observe general file system performance trends. That said, there are limitations to this model that we discuss in Section 4.

Workloads. We used three workload personalities from Filebench [4] to evaluate the file systems across a range of emulated devices. Web-server is a read-dominated workload. File-server is metadata-heavy workload that performs lots of file creates and deletes. Finally, mail-server personality generates a balanced mix of read and write operations.

Workload performance is significantly influenced by caching effects in the operating system. Picking the right cache-to-dataset size ratio for evaluation is hard since it varies significantly in real environments. While we did not want to make the cache so small that it is entirely ineffective, we also did not want to evaluate caching performance exclusively. We fixed a 1:2 (cache:dataset) ratio that ensures both cache and I/O activity across all workloads. Each experiment ran for 20 minutes and was repeated several times.

4. Findings

Our findings are based on a large number of experiments across multiple file systems, workloads, and device speeds. Figures 1–5 depict the throughput of Ext4, XFS, Nilfs2, F2FS, and BTRFS under the Web-, File-, and Mail-server workloads. In each plot, the X-axis denotes the latency for the storage device and the Y-axis denotes the throughput reported by Filebench. Inset graphs zoom into the sub-millisecond range. Each figure plots the experimental results (solid-line) and the UPEM curve (dashed-line). We report means and 95% confidence intervals. Figure 6 combines the performance data for all file systems in a single plot.

To validate that dm-delay works properly we first collected average request latencies from `/proc/diskstats` and they matched the values set. We also wanted to get a sense of how far off is our emulated device from a real one. We ran the above workloads on a Micron MTFDDAK128MAR-1J SSD and collected average request latencies at the block layer along with the Filebench-reported throughput. The big yellow circles on Figures 1–5 depict these results. Some graphs are missing real device points because request latencies were higher than 10ms. These large latencies were caused by large delays in the block-level and device queues and are typical for write-intensive workloads. For the Web-server workload, which is read-intensive, the latency was always within 10ms.

From the data, we also conclude that in the majority of the cases our model's performance matched the performance of the real device. The only case when SSD performance was much higher than the one of an emulated device was F2FS under Mail workload. Further investigation revealed that for this workload, although the average SSD latency was about 4ms, the average read latency was only 1.5ms. The performance of the foreground File-server workload was mostly impacted by reads and not by writes which can be completed asynchronously. Since the emulated device had a latency set to 4ms for both reads and writes, F2FS could not achieve the same throughput due to the higher read latencies it experienced. When we set the read delay for the emulated device to 1.5ms, the throughput of F2FS increased to almost 4,000 ops/sec—close to the one reported by the real SSD—validating our hypothesis.

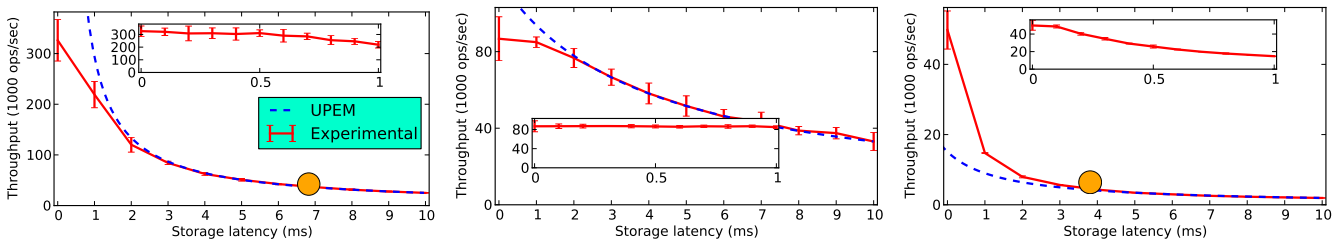


Figure 1. Ext4 throughput depending on the underlying storage speed for three different workloads. Yellow circle represents real SSD performance.

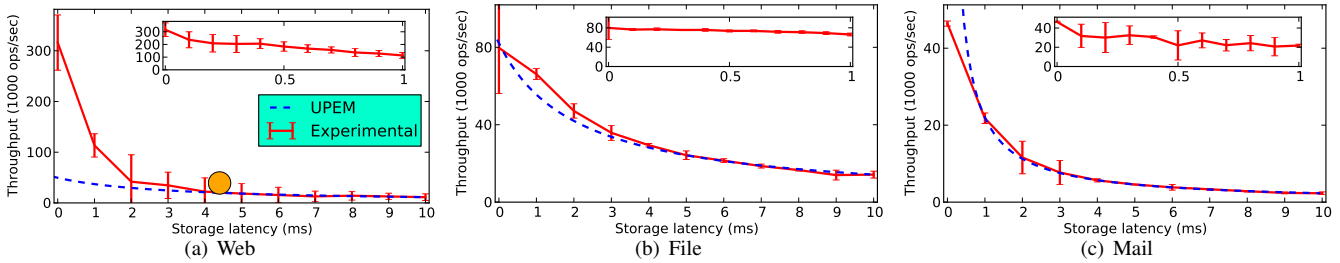


Figure 2. XFS throughput depending on the underlying storage speed for three different workloads. Yellow circle represents real SSD performance.

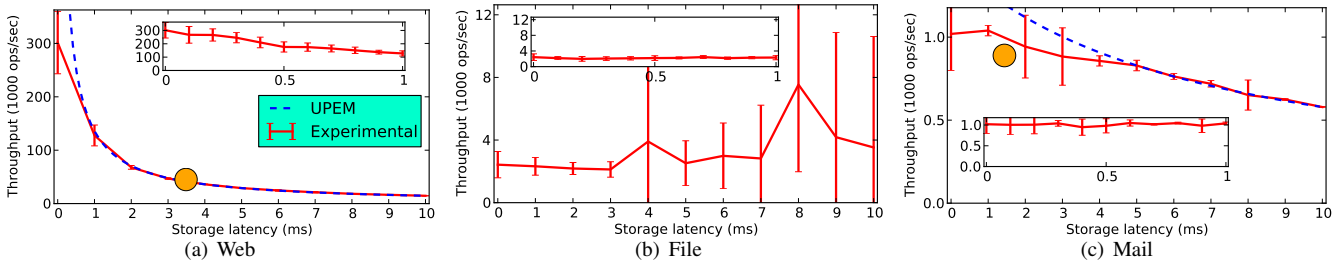


Figure 3. Nilfs2 throughput depending on the underlying storage speed for three different workloads. Yellow circle represents real SSD performance.

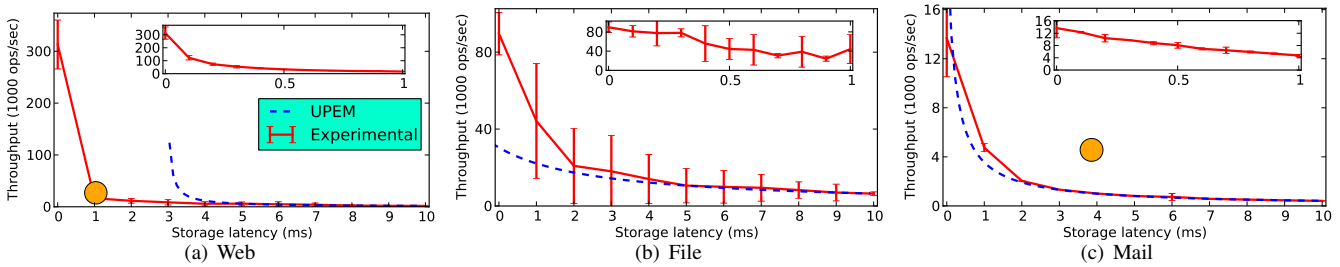


Figure 4. F2FS throughput depending on the underlying storage speed for three different workloads. Yellow circle represents real SSD performance.

Below we describe three observations about the behavior of file system performance as the speed of storage increases:

Observation 1. Almost all file systems improve their performance as underlying storage latency decreases. However, the rate of improvement and the range of latencies with the most rapid improvements varies drastically across the file systems and workloads.

Performance generally grows much slower below 1ms—flat lines are prevalent in the inset graphs. This fact stresses the need for optimizing or redesigning modern storage software stack to successfully break sub-millisecond performance barriers.

Performance is also somewhat unstable, leading to wide confidence intervals. This is more common in the sub-millisecond range, e.g., Ext4-(Web), XFS-(Web), but occurs for high latencies as well, e.g., Nilfs2-(File). We ran over

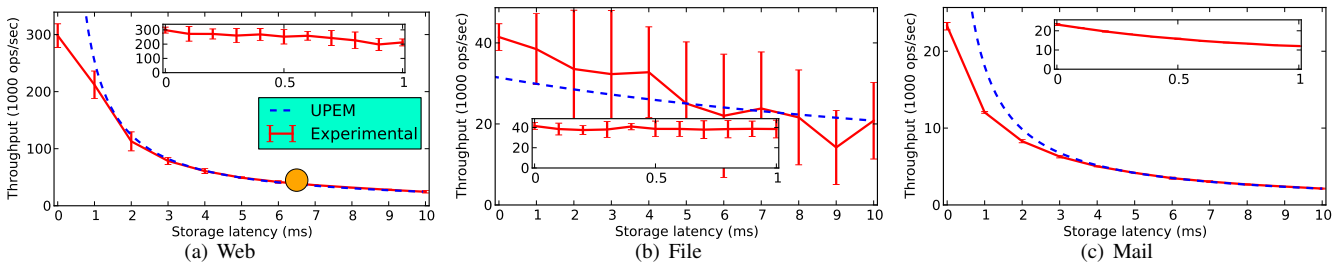


Figure 5. BTRFS throughput depending on the underlying storage speed for three different workloads. Yellow circle represents real SSD performance.

	Web	File	Mail
Ext4	∅	F2FS	XFS, BTRFS
XFS	∅	BTRFS, F2FS	Ext4
BTRFS	F2FS	XFS, F2FS	Ext4
Nilfs2	F2FS	∅	F2FS
F2FS	BTRFS, Nilfs2	Ext4, XFS, BTRFS	Nilfs2

Table 1. The listed file systems experienced performance inversions for the given workload.

one hundred experiments for cases with wide confidence intervals, and the resulting uniform distribution of results leads us to believe that there is a lot of unknown activity that justifies further investigation.

The performance leader is typically Ext4, as shown in Figure 6, for almost all latencies and all workloads except Mail-server, in which XFS dominated (except with 0 latency). Nilfs2 performs poorly for write-intensive File- and Mail-server workloads, but interestingly scaled rather well for the read-only Web-server workload.

Observation 2. While the relative performance of file systems is maintained across workloads in the majority of cases, we did observe several performance inversions between file systems as device latencies change. We list the set of inversions we observed in Table 1. The performance inversions can be best observed in Figure 6. Let us consider F2FS performance for the File-server workload as an example. BTRFS performs better than F2FS for high-latency devices but then performs worse in sub-millisecond latency range. Further, F2FS which performs worse than BTRFS, Ext4, and XFS at higher device latencies outperforms all of these file systems when device latency approaches zero. Further investigation into file system design specifics is required to understand what makes a file system excel at one latency but then slow down for a different latency.

Observation 3. We classify the remaining observed behaviors with respect to our User Performance Expectation Model into three classes.

Class I systems follow UPEM closely until the storage latency reaches approximately 1ms, but not from 1ms to 0ms. Ext4-(Web, File), XFS-(Mail), Nilfs2-(Web), and BTRFS-(Web) fall into this category. This indicates that upgrading storage devices in Class I systems will generally show clear benefits, but if the new storage device has a latency below

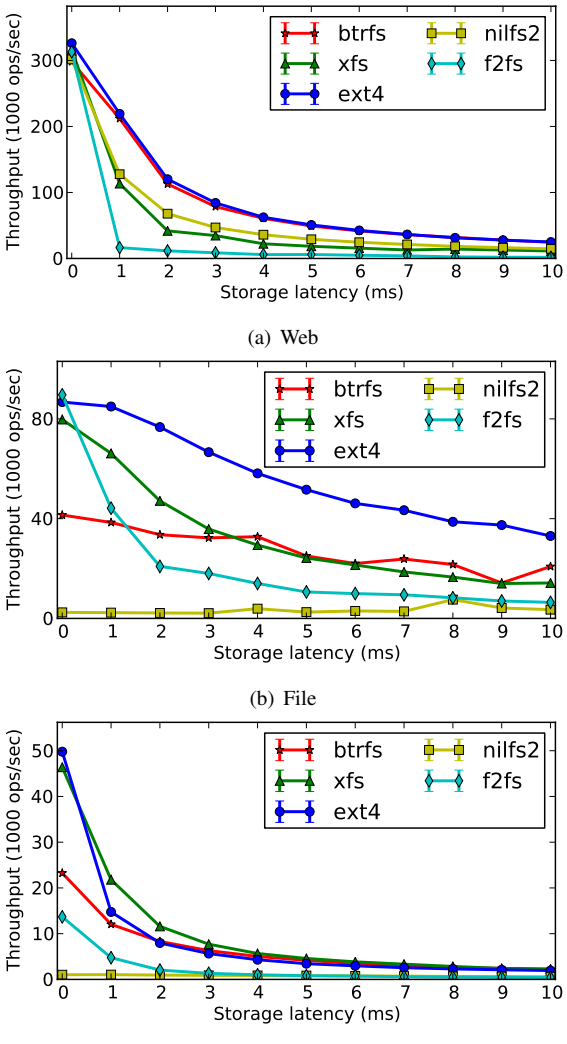
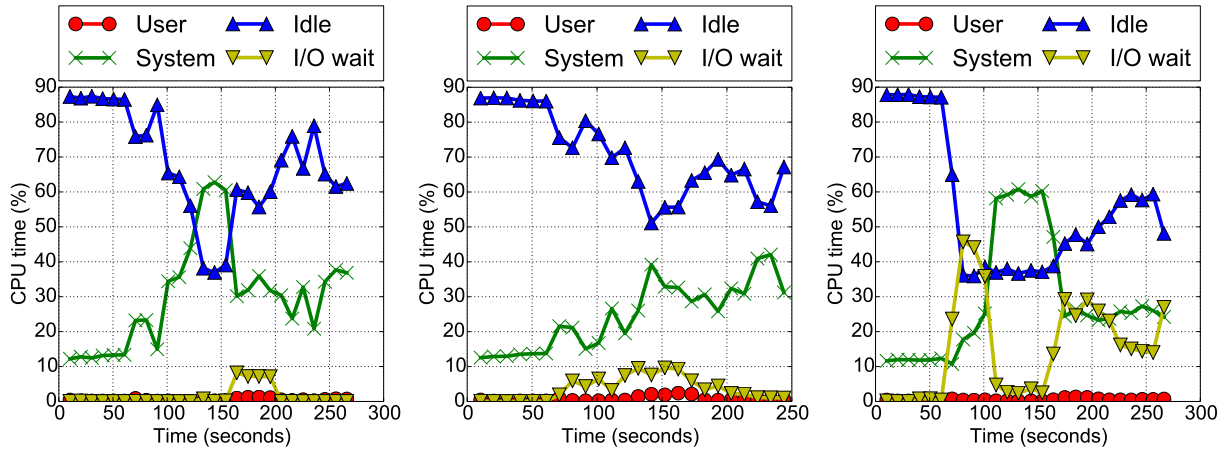


Figure 6. Throughput of Ext4, XFS, Nilfs2, F2FS, and BTRFS depending on storage speed for Web-, File-, and Mail-server workloads.

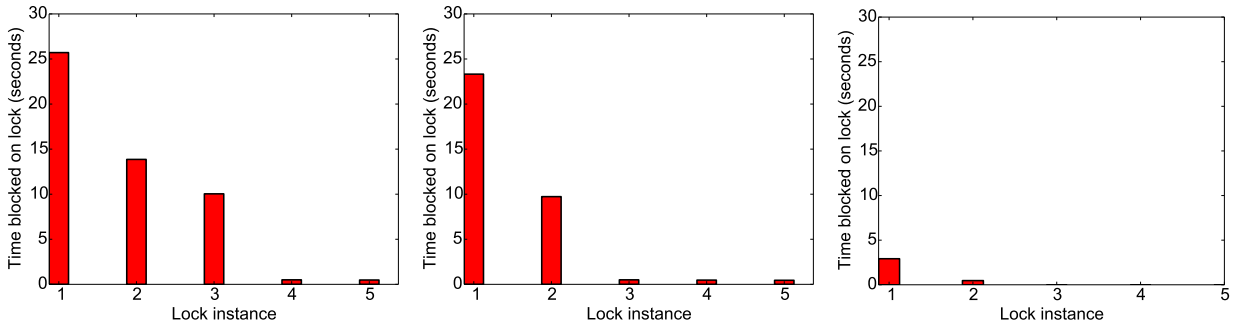
1ms, then the user will experience diminishing returns on the investment.

Class II systems outperform UPEM. Specifically, for Ext4-(Mail), XFS-(Web, File), and F2FS-(File, Mail), the



(a) Device delay 0ms (b) Device delay 1ms (c) Device delay 10ms

Figure 7. CPU utilization vs. time with Nilfs2 under File-server workload



(a) Device delay 0ms (b) Device delay 1ms (c) Device delay 10ms

Figure 8. Nilfs2's `i_mutex_dir_key` lock contention under File-server workload

UPEM curve is below the experimental data curve indicating that the file systems perform better than expected. This means that users will find surprisingly faster systems after upgrading their storage (to any latency).

Class III systems initially scale as expected with higher latencies, but then perform worse than UPEM at some point as the latency gets lower. All combinations in this class perform worse than UPEM at the 1ms mark, but some much sooner. For example, Nilfs2-(Mail) scaling slows at 4–5ms delay and BTRFS-(Mail) at 2–3ms delay. Interestingly, F2FS-(Web) scales poorly and well below expectations, but then does significantly better in the 0ms to 1ms range.

For Nilfs2-(File), the performance actually *decreases* as devices get faster. The UPEM curve is missing because the model cannot fit such a poorly scaling curve. We present a detailed analysis of this behavior in the following section.

5. Nilfs2 Analysis

One of the unexpected findings is related to Nilfs2 and the File-server workload where the performance decreases as the device gets faster (see Figure 4(b)). The analysis methodology we discuss below is broadly applicable and can be used to analyze other performance behaviors as well.

The first step is to understand the workload itself. The File-server workload uses 82K small files. Average file size is 128KB; files are distributed across directories that contain 20 files on average. 50 threads operate on these files issuing create, write, append, read, and delete operations. Files are read and written completely and sequentially, and the average append size is 16KB. The file set is preallocated and caches are dropped before every experiment so that the system state is the same across all experiments.

Next, to understand workload performance behavior, we examine how CPU time is spent by the workload across different device delays. Figure 7 shows a break-down of CPU utilization across user, system, I/O wait, and idle time, as the execution of the workload progresses for three different device delay values, 0ms, 1ms, and 10ms. With large device delays (Figure 7(c)), we expect that a significant amount of time would be spent waiting for I/O by the workload threads. We also anticipate that this I/O wait time would decrease as the latency of the device decreases; this is corroborated by

Figures 7(a) and 7(b). Furthermore, since both Nilfs2 and ramdisk operations consume time, we anticipate that the percent of time spent within the system software to increase as I/O wait times decrease. However, as we see in Figures 7(b) and 7(a), while the I/O wait time decreases significantly as the latency of the device decreases, the system time does not increase accordingly. In fact, if we compare the system CPU utilization for 0ms and 10ms latencies, we notice that the slower device spends more system time than the faster device. High idle time with a busy workload implies that the worker threads are sleeping. This led us to the hypothesis that *as device speed increases, the Filebench threads increasingly contend for a shared resource, thus increasing system idle time.* To evaluate this hypothesis, we obtained block level statistics for the emulated devices for each delay. For a delay of 10ms, the average throughput for the device is 38MB/s for reads and 98MB/s for writes. For a delay of 1ms, the throughput of the device is 44MB/s for reads and 110MB/s for writes. An increase was expected because I/O performance increased but it was not as significant as we had expected, indicating the existence of other resource bottlenecks. Finally, for an emulated device with no added delay the throughput is 42MB/s for reads and 101MB/s for writes. Given that the utilization of the device remains roughly the same as with 1ms delay, our hypothesis was strengthened and we pursued it further.

Locks are a common shared resource inside the kernel. The Linux *perf* tool [6] reports on various events inside the kernel including locks acquired, locks released, and locks contended. We recorded the lock events reported by perf for 60 seconds for the workload in question. Under smaller latencies we identified a single lock that caused a large amount of contention named `i_mutex_dir_key`. It is used when requesting information about the inode mapping file, checkpoint files, and regular file inodes. Backtracking the function references to Nilfs2 code, this mutex is used only when an inode is being unlocked, which occurs if newly initialized from storage or just created.

Perf lock reports the amount of contention on each instance of the `i_mutex_dir_key`. Figure 8 depicts the five mutex instances with most wait times are shown for different delays over the 60 second recording. When the device latency is 10ms, the contention on this type of lock is very low, only 3 seconds for one instance and under half a second for the rest. As the latency of the device decreases, the contention across different instances increases significantly reaching 33 seconds in total for the 1ms delay and over 50 seconds when the device has no added delay.

Nilfs2 uses the concept that everything is a file, including file metadata, which also have an inode that references them (referred to ifile). Design knowledge helps in associating file system performance with workload characteristics; the File-server workload is write intensive and involves a lot of metadata operations. Finally, Nilfs2 is fully log-structured. Every time that a block is modified, it is copied from the previous segment where it was stored into an uncommitted segment. This operation also involves copying a new version of the inode into the given segment and an update in the `DAT` file that is used to optimize the index updates of a data block. All of these operations increase the amount of inode-level lock contention traffic to the detriment of Nilfs performance.

6. Related Work

A number of studies in the past conducted performance analysis across file systems [8, 18]. However, the authors overwhelmingly used only one and rarely just a few storage devices. Unlike other studies, we systematically evaluate the entire spectrum of current and future device speeds. Several studies proposed accurate models of storage devices [22] and file systems [23]. In contrast, we use the simpler but more generic latency-based model that characterizes trends in file system behavior as device latencies change, providing us coarse-grained, easy-to-use insights. Some recent studies proposed improving file system performance for fast devices by completely eliminating the block abstraction [11, 21]. Others optimized existing I/O stack performance for fast block devices [7, 14], but the scaling limits of existing file systems were never evaluated. Our complementary contribution establishes the latencies below which current block-based file systems do not scale.

7. Conclusion

Computer systems are entering an era of diverse storage devices with latencies ranging from a few microseconds to several milliseconds. Users expect their file systems to perform in a predictable manner on devices of various speeds. In particular, when upgrading underlying storage to faster alternatives users expect file systems to become proportionally faster. Our study demonstrated that in practice it is hard to predict file system performance for fast devices based on performance numbers for slower devices. Furthermore, performance improvements are highly sensitive to both file system and workload selection. Across all three of our workloads, we found that performance limits as well as performance inversions exist across all five file systems. These findings are revealing and provide pointers for future work on building file systems that can better utilize next generation high-speed block devices.

In this study, we also demonstrated how the empirical measurements are valuable in finding underlying file system bottlenecks such as the significantly increased lock contention we found in Nilfs2 with high speed devices. Future work along these lines involves analyzing more file systems in depth. One caveat with the the device emulation based approach is that it may need fine tuning to better match the performance characteristics of a real device under consideration. One specific instance was the anomaly we found in

the case of F2FS for one workload; we pinned this down to a read vs. write performance discrepancy between the emulated device and actual-device behavior. Such fine tuning should ideally consider read-write latency differences, I/O sizes, as well as I/O parallelism, among other device-specific properties. While these considerations are important for in-depth comparative analysis, we found that simple device emulation could provide a wealth of information and guide detailed analysis well.

Acknowledgments

This work is supported in part by NSF awards CNS-1018262 and CNS-1448747 and an Intel ISRA award.

References

[1] Device-mapper mailing list. `http://www.redhat.com/mailman/listinfo/dm-devel`.

[2] Dm-delay device-mapper target. `<LinuxSourceCode>/Documentation/device-mapper/delay.txt`.

[3] Ext4. `http://ext4.wiki.kernel.org/`.

[4] Filebench. `http://filebench.sourceforge.net/`.

[5] Nilfs. `http://www.nilfs.org/`.

[6] Perf-tools. `<LinuxSourceCode>/Documentation/perf/perf.text`.

[7] M. Bjørling, J. Axboe, D. Nellans, and P. Bonnet. Linux block IO: Introducing multi-queue SSD access on multi-core systems. In *Proc. of SYSTOR*, 2014.

[8] R. Bryant, R. Forester, and J. Hawkes. Filesystem performance and scalability in linux 2.4.17. In *Proc. of ATC*, 2002.

[9] A. M. Caulfield, T. I. Mollov, L. A. Eisner, A. De, J. Coburn, and S. Swanson. Providing safe, user space access to fast, solid state disks. In *Proc. ASPLOS*, 2012.

[10] J. Coburn, A. Caulfield, A. Akel, L. Grupp, R. Gupta, R. Jhala, and S. Swanson. Nv-heaps: Making persistent objects fast and safe with next-generation, non-volatile memories. In *Proc. ASPLOS*, 2011.

[11] S. R. Dulloor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, and J. Jackson. System software for persistent memory. In *Proc. of Eurosys*, 2014.

[12] J. Guerra, L. Marmol, D. Campello, C. Crespo, R. Rangaswami, and J. Wei. Software persistent memory. In *Proc. of USENIX ATC*, 2012.

[13] C. Lee, D. Sim, J. Hwang, and S. Cho. F2FS: A new file system for flash storage. In *Proc. of FAST*, 2015.

[14] Y. Lu, J. Shu, and W. Wang. ReconFS: a reconstructable file system on flash storage. In *Proc. of FAST*, 2014.

[15] C. R. Lumb, A. Merchant, and G. A. Alvarez. Façade: Virtual storage devices with performance guarantees. In *Proc. of FAST*, 2003.

[16] O. Rodeh, J. Bacik, and C. Mason. BTRFS: The Linux B-tree filesystem. *ACM TOS*, 9(3), 2013.

[17] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured file system. *ACM TOCS*, 10(1), 1992.

[18] P. Sehgal, V. Tarasov, and E. Zadok. Evaluating performance and energy in file system server workloads. In *Proc. of FAST*, 2010.

[19] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck. Scalability in the XFS file system. In *Proc. of ATC*, 1996.

[20] H. Volos, A. J. Tack, and M. Swift. Mnemosyne: Lightweight persistent memory. In *Proc. of ASPLOS*, 2011.

[21] H. Volos, S. Nalli, S. Panneerselvam, V. Varadarajan, P. Saxena, and M. M. Swift. Aerie: Flexible file-system interfaces to storage-class memory. In *Proc. of Eurosys*, 2014.

[22] M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. R. Ganger. Storage device performance prediction with cart models. In *Proc. of MASCOTS*, 2004.

[23] T. Zhao, V. March, S. Dong, and S. See. Evaluation of a performance model of lustre file system. In *Proc. of ChinaGrid*, 2010.