2012 IEEE Seventh International Conference on Networking, Architecture, and Storage

An Empirical Study on the Interplay Between Filesystems and SSD

Ke Zhou, Ping Huang, Chunhua Li and Hua Wang
School of Computer Science & Technology,Huazhong University of Science & Technology
Key Laboratory of Data Storage Systems, Wuhan National Lab for Optoelectronics
Wuhan, China
Email: pinghp.hust@gmail.com,{k.zhou,li.chunhua,hwang}@hust.edu.cn,

***Abstract*—This study presents a comprehensive empirical investigation on the interplay between actively-deployed file systems and an SSD. We test and analyze the performance of four widely used Linux file systems, which are ext2, ext3, XFS and Reiserfs, on an SSD with a range of different workloads. It is primarily intended to serve two purposes. One is that considering its widespread adoption trend, we are realistically motivated to have first-hand numbers of the actual performance of the emerging storage technology, especially in the contexts of daily deployment scenarios. The other goal is that we attempt to disclose the internal details behind the SSD's thin interface from a high-level perspective, totally different than those previous studies, which are typically micro-test-only. As a result of this study, we obtain several interesting and useful findings: (1) Generally, different file systems perform disparately on the SSD due to their various design principles, sometimes even with up to one order of magnitude of performance discrepancy. (2) File system format/mount options and workload characteristics have significant impacts on performance. (3) SSD would deliver optimal performance if used in a friendly manner. (4) Workloads, file systems and SSD interact in an intrinsically complicated way and in order to have optimal synergistic performance anticipation, users should seriously consider all of the three factors together, when setting up their SSD-based storage systems.**

***Keywords*-solid state drives; file systems; performance evaluation;**

I. Introduction

The performance gap between CPU computing speed and the lagging I/O storage subsystem has long been existing and has recently become unprecedentedly grave with the emergence of multi-core processors and the ever-growing requirements of processing extremely large volume of data. Traditional storage systems built only on both DRAM as the primary storage and rotating Hard Disk Drives(HDDs) as the secondary storage have severely limited the achievable computing power potentials of modern processors, significantly slowing the overall system performance, especially in data-intensive environments [1], [2]. Though a large variety of techniques[3], [4], [5], [6] have been suggested in the research community to alleviate that problem, the whole system performance still has been consistently suffering due to the orders of magnitude access gap between DRAM and HDDs[7]. To this end, for the past two decades, flash memory technology whose performance and cost lie reasonably in the intermediacy between DRAM memory and HDD has been intensively researched and intelligently explored to be integrated into the storage stack to improve the storage subsystem [8], [9], [7], [10], [11], [12], [13]. Actually, the integration of flash technology into the storage hierarchy has largely been proved to be very effective, not only in the viewpoint of performance[8], [9], [10], but also in terms of energy consumption and cost[1], [2].

Built entirely on semiconductor technologies, flash-based SSDs bear many salient features over long-standing HDDs equivalents which consist of magnetic platter surface and mechanically rotating components. The advantages include fast random access, high shock resistance, small form size, low power consumption, etc. However, on the other hand, they also suffer from a variety of inherent technical limitations. For example, flash pages can be reused only after going through erasure operations, which imposes significant impacts, especially on the small random write performance [14], [15] and they can only sustain a limited number of Programming/Erasure(P/E) cycles, with 10K and 100K being the typical values for NAND and NOR chips, respectively, after which storage on flash would become unreliable[12], [16]. Thus, in order to get the maximum out of flash memory while at the same time avoiding its downsides, storage systems are typically designed to use the flash components in a friendly manner[17], [7], [10], [18]. To further better realize this goal, we think we should judiciously rethink or redesign the storage systems in at least two important aspects. First of all, as system designers, we should have a good understanding of the unpredictable and complicated performance dynamics of the underlying flash-based SSDs, especially the very sophisticated modern SSDs[19], [20], [21], from a combined perspective of system design and SSDs internal dynamics. Second of all, as users, we hope to use the SSD-based storage systems in an effective and efficient way, which requires a combined consideration of targeted applications, system designs and SSD dynamics. In other words, users hope to have some useful instructions at hand to facilitate their using or setting up SSD-based storage systems.

Toward that goal, in this paper, we present an empirical study on the interplay between four actively used Linux file systems and an SSD. Our research objective is two-fold. On the one hand, we want to see how the file systems and SSD

978-0-7695-4722-0/12 $26.00 © 2012 IEEE
DOI 10.1109/NAS.2012.21

interact with each other in reality under various workloads and how the mutual interaction relates to performance. On the other hand, we attempt to ascertain the mysterious SSD internals behind the provided thin interface from a high-level perspective, by observing and comparing the exhibited performance behaviors of different workloads. While there already exist a number of previous studies on characterizing SSDs [21], [19], [22], [20], our method is still considered to be novel in that it has more practical implications on the SSD usage than those. We selected ext2, ext3, XFS and ReiserFS as our study examples, due to their wide deployment in the academia and industry production systems. Then we used FileBench which is an application level workload generator capable of emulating a large variety of workloads to generate the driving workloads. Workloads with different characteristics can be easily defined in FileBench thanks to its flexible *Workload Model Language(WML)*[23]. We ran a rich set of experiments using different workloads and various file system format and mount options to drive our test platform. Our experimental findings reveal that different file systems exhibit different performance behaviors even when running the same workloads, file system format and mount options have significant impacts on performance and choosing the appropriate system configurations is quite beneficial. Finally, we provide some empirical suggestions/instructoins, aiming to help users with setting up or optimally configuring their SSD-based storage systems.

The remaining of this paper is structured as follows. We give the background and our detailed motivation in Section II. We survey the related work in Section III. Then we briefly describe our experimental setups and the used workloads in Section IV. In Section V we present and discuss the empirical results at length. In the end, we conclude in Section VI with a summary of the empirical study and several suggestions for users facing to use SSDs as their storage alternatives.

II. Background and Extended Motivation

A. Flash Memory and SSD

Flash memory is a non-volatile storage medium and falls into two categories named NAND and NOR flash chips, both of which can be further characterized into Multi-Level Cell(MLC) and Singe-Level Cell(SLC) types according to the amount of stored bit information per programmable cell[12]. NOR flash chips support random accesses in bytes granularity and are often used for special purposes, e.g., storing code, while NAND flash chips are designed for data storage and thus are more denser and only page-addressable. Since in this paper we only deal with NAND MLC flash chips, we subsequently use flash memory to refer to them specifically. Flash memory is internally built upon a hierarchy of the basic concepts of page which is typically 4KB in size, block which consists of hundreds of pages, and plane which groups thousands of blocks as an individual operational unit to realize elementary internal parallism[12], [20]. Compared with magnetic storage, flash memory exhibits many widely different operational idiosyncrasies. First, flash memory supports three kinds of operations, *read*,*write* and *erase*. *Read* and *write* operations are performed in units of pages, while *erase* operation is only allowed to be conducted in whole blocks. Second, the supported operations are asymmetric. *Reads* complete within tens of microseconds, *writes* finish within hundreds of microseconds, while *erasures* are much more expensive and take several milliseconds to finish. Furthermore, pages can only be reused after they are subject to the Garbage Collection process and erased as clean again. This peculiarity causes the notorious "erase-before-write" problem[24]. Third, flash memory is only able to withstand a limited number of write and erase cycles after which reliability would become a problem.

SSDs are internally composed of flash memory chips as the physical storage medium, a controller which is responsible for the conversion between the upper layer request commands and flash's native commands, certain amount of DRAM buffer which stores the metadata information including FTL mapping information and temporarily buffers the incoming and outgoing data, and a Host Interface Logic which enables the SSDs to emulate acting like a traditional block-based HDD. An important intermediate software in SSD is the Flash Translation Layer(FTL), which dynamically manages how the logical addresses seen by the upper layer are mapped to physical locations on the flash chips, hiding the peculiarities behind the provided interface. To even out the consumed erasure budget among all the chips, FTL implements a wear-leveling algorithm and a garbage collection strategy[12]. Good wear-leveling algorithm prevents the whole SSD from prematurely breaking down due to being partially worn-out caused by workload locality. Garbage collection allows the flash chips to be reused and provides an opportunity of swapping hot and cold regions[25]. Modern SSDs have evolved significantly and become rather sophisticated with many advanced features, for instance, highly parallelized architectures, good prefetching and caching mechanisms and advanced commands supporting[20], [26], etc.

B. File Systems

Ext2, Ext3,XFS and Reiserfs, among the hundreds of Linux local file systems, are the most popular, stable and widely used candidates in Linux servers. All of the four file systems were designed with different principles and provide various format and mount options. Ext2[27] divides the disk partition into fix-sized *blocks* and introduces the concept of *block groups* each of which consist of a fixed number(equates to the number of bits in a block) of blocks to reduce file fragmentation. The first block is excluded from any *block group* and used to store the *superblock*.

Each block group contains a set of metadata including *group descriptors*, *data block bitmap*, *inode bitmap*, *inode table* and the remaining file data blocks. It is independently responsible for the management of the whole space belonging to the group. The heuristic used when allocating blocks is that Ext2 tries to obtain file inodes from the same parent directory and store data in the same group with the goal of reducing fragmentation and enhancing locality in mind. Address translations in Ext2 are carried out simply by performing linear calculations, starting with the array-like information kept in the corresponding inodes. Aiming to improve the consistency management problem associated with crashes, Ext3[27] is developed based on Ext2 with an extension of JBD journaling layer. Every write request to Ext3 is in advance logged in the journal, then committed to their final destinations. By doing that, the system state is guaranteed to be consistent even when crashes occur by checking the journal. Ext3 provides three journaling modes, *Journal*, *Ordered*, *Write-back*. Except for the added journaling mechanism, Ext3 is identical to and thus compatible with Ext2. Unlike the statically allocated on-disk layouts of Ext3 and Ext2, XFS[28] adopts B+ trees to manage dynamic allocation of inodes, tree space and represent the relationship between meta-data and data of file/directories. Data and meta-data are all stored in variable sized and contiguous *extents*. Additionally, XFS's partition is also divided into fix-sized regions named *allocation groups(AGs)*, similar to *block groups* in Ext2 and Ext3. However, AGs are introduced mainly for scalability and parallelism purposes. XFS uses a delayed allocation policy for the better of large-size files. ReiserFS[29] is a general-purpose, journaling file system. In Reiserfs, partition is divided into fixe-sized blocks as well. All the concepts in Reiserfs are represented by *items* which are managed by a *balanced S+ tree*. Each *item* is uniquely identified by a key (somewhat similar to the usual inode number). The most distinguishing feature of Reiserfs is the so-called *tail packing* feature, which is a scheme intended to reduce internal fragmentation of small-sized files by packing the small amount of data into the corresponding inode block.

C. Extended Motivation

Flash memory has been traditionally used in limited scenarios, for instance, in mobile embedded devices and consumer electronic appliances, which require relatively small capacity. Recently, large-scale storage systems based exclusively on SSD disks have been built[1]. However, these kinds of SSD-based systems are totally SSD-oriented built in every aspect from the very beginning. With the SSD prices continuously decreasing and capacities steadily expanding, it is quite likely that SSD would become increasingly popular and play a critical role even in our daily use scenarios in the forseeable future, which renders us in a position to investigate how it performs in daily situations and how to make best use of it through better understanding of our targeted workloads, system configurations and physical properties of the underlying SSD, in the meanwhile, with almost no requirement of modifications to the large amount of legacy systems and applications.

Extensive benchmarking is an important and effective way to explore system possibilities and see the benefits and drawbacks of file systems under various circumstances[30]. Through extensive testing, we can know well of the behaviors of different file systems on SSD and use the obtained observations and insights to wisely guide the integration of SSD into our storage systems. Also, different applications expect different requirements from the supporting system and infrastructure and have distinct impacts on them by executing or stressing different paths[23]. In this regard, we conduct benchmarking tests on Ext2, Ext3, XFS and Reiserfs, with various kinds of workloads generated by FileBench[23]. It should be noted that though there are a number of flash-specific file systems already available, we don't choose them as study examples because they are either designed for specific scenarios, e.g., journaling file systems JFFS2[31] and YAFFS[32] are more tailored for embedded environments, or still not mature enough to be widely deployed[18].

The mysterious scene behind the SSD block interface has been attracting a great deal of attention from the research community, since good understandings of the internal organizations and algorithms are of great value to helping the upper layer designs. Unfortunately, the true internal details are often regarded as highly confidential commercial intellectual property and are often absent or obscured in the available specifications. Those specifications only provide very conservative field data and even sometimes do not faithfully reflect the real story[21]. Previous studies on characterizing SSDs are often conducted with micro-benchmarking approaches[21], [19], [20]. They typically put the SSDs subject to well-known and controllable workloads and observe the correspondingly exhibited performance behaviors to infer the internal details. While those studies are crucial to characterizing SSDs, we claim that it's only part of the story, since some sophisticated mechanisms may only come into effects when tested under more complicated real world workloads with non-deterministic variations. Besides of that, one important conclusion derived from previous studies is that though SSD can deliver striking raw performance, if not used properly by the upper layers, it can possibly even underperform traditional HDDs[9]. Thus, it is necessary and beneficial to examine SSD in a more comprehensive contexts, for example, using more realistic workloads.

In summary, our purpose of this study is to attempt to answer the following questions: (1) Does the promising raw performance of SSD always exist? How does the SSD perform under different file systems and different workloads? (2) Are there any interesting findings about the internal details of SSD, which were not possible to be found by micro

approaches in previous work? (3) How does the workloads, file systems and SSD interact with each other and how they synergistically affect the overall performance? (4) Is it possible to conclude some practical and useful insights and suggestions for users' references?

III. Related work

An enormous amount of works have been conducted on SSD during the past decade. In this section, we only survey the most related works, specifically including studies on SSD, integrating SSD into storage stack and evaluating file systems and characterizing SSD.

Studies on SSD: Studies on SSD are mainly centered around how to improve the internal structures and FTL algorithms, overcome the inherent technical limitations and characterizing SSD. Agrawal et al.[12] discussed a wide range of internal design trade-offs when implementing SSDs and developed an SSD simulator. Huang et al.[33] constructed a black-box performance model for SSD, which can accurately predict the performance given the workloads characteristics. Hu et al.[26] specifically investigated by simulation how several key internal factors named *flash page size*, *allocation strategy*, *advanced commands* and *parallelism* affect SSD performance and endurance under different workloads and concluded several insights that is beneficial to SSD designs and deployment. Boboila et al.[34] specifically examined the endurance of flash chips using a range of approaches and tried to determine the underlying factors affecting flash endurance. FTL and buffer management are the two key components and most performance-sensitive in an SSD. A variety number of corresponding schemes have been proposed. DFTL[35] is a pure page-mapping scheme which exploits workloads locality to dynamically load only part of FTL mapping entries to save on-flash RAM, while at the same time doesn't degrade performance. HAT[36] is a novel FTL scheme that deploys Phase Change Memory(PCM) to assume the task of managing FTL mapping entries. Due to the separated path of FTL accesses from data I/O and PCM merits, HAT has been demonstrated to be able to achieve page-mapping level performance at a cost of block-mapping scheme. More recently, Song et al.[37] proposed to leverage the workloads spatial locality to reduce the consumed RAM for storing mapping information. Their basic idea was to compact the mapping entries of consecutive/sequential writes into just one entry. BPLRU[15] is a buffer management scheme to improve random write performance. It converts expensive *full merge* to easy *switch merge* through *page padding* to reduce erases. Similarly, PUD-LRU[24] also attempts to reduce the number of erasures and improve response time, however, by judiciously destaging blocks based on their recency and frequency which can generate better *clean efficiency*[12].

Integrating SSD into storage stack: SSDs have been actively deployed in the storage stack to improve performance in the form of replacement of HDDs or hybrid coupling with HDDS which takes the complementary advantages of both. ChunkStash[7] utilizes flash-based SSD's fast random access to hold the hash indexes in replacement of HDDs in deduplication systems to address the disk-index bottleneck problem, significantly improving the deduplication throughput and system scalability. BufferHash[9] is new hash-based key-value store completely built upon DRAM and flash for data-intensive networked systems, and it has much greater insert and update performance. Both of ChunkStash and BufferHash have used flash-friendly mechanisms in the design to better use SSDs. SSDAlloc[8] uses SSD as external memory extension to expand the memory space at reasonable cost. ICASH[10] intelligently couples HDD and SSD by storing unchanged reference blocks on SSD and logging frequent small update deltas on HDD, taking advantage of their respective merits. By the same principle, Griffin[17] uses a log-structured HDD as a caching layer on top of SSDs to absorb overwrites and periodically migrates the cached data to SSDs. Griffin has been shown to reduce the writes to SSD and thus extend its lifetime.

Hystor[11] selectively stores both performance-affecting data blocks and semantically-critical blocks on SSD to speed the system up.

Evaluating file systems and SSD: Surprisingly, to the best of our knowledge, there are very few studies in the literature on evaluating file systems performance specifically on SSDs. Sehgal et al.[23] conducted a comprehensive evaluation of the performance and energy consumption of the exact four files systems that we use in this study for common server workloads, but in the contexts of HDDs. Being the closest work to our evaluation, Polte et al[38] used IoZone to generate a number of microbenchmark patterns to test the examined SSDs and HDDs in the context of ext3 file system. They concluded that SSD is comparable or even more cost-effective than HDDs in terms of performance per dollar for sequential and random access patterns, respectively. However, their examined test space is very limited, while our study features a much richer set of workload characteristics and can better represent the real world facts. Seppanen el at.[39] discussed the impacts that high-performance PCI-express SSDs have on Linux system design and correspondingly proposed several guidelines for benchmarking SSDs under Linux. There are numerous works on attempting to peek into SSD internals. Grupp et al.[21] characterized flash memory through exhaustive experiments on a test rig from performance, reliability and power consumption perspectives and made some interesting and even unexpected findings. Chen et al.[19] tried to gain insights into the SSD internal details by observing the ext3 file system performance on three low-, median- and high-end SSDs using *Intel® Open Storage Toolkit* benchmarking tool. Later on, they used this same tool to specifically study the roles of rich internal parallelism within modern sophisticated SSDs. Yoo et al.[22]

tried to reveal the SSD internal details from the energy consumption perspective, based on the fact that different internal organizations would entail different external current behaviors when operations are being carried out. However, those previous attempts on characterizing SSD are only part of the extremely complex story, because their micro-benchmarkings typically access raw devices directly thus not taking into account the effects of cache layers and generate prior-known, monotonous, predictable patterns, which could possibly make their conclusions incomplete or even superfluous for further extrapolation. Furthermore, microbenchmarks tools designed for HDDs contexts is not assumed to be very appropriate for SSDs because of their much different properties from HDDs[39]. Therefore, we believe that using real world workloads to characterize SSDs is necessary and more accurate, persuasive for practicality purposes.

IV. Experimental Methodology

A. Experimental Setup

Our testbed consists of a server featuring a dual-core processor, with 4GB memory, and a Kingston MLC 60GB SSD. The SSD was connected to the server through SATA interface. To better expose the characteristics of SSD, we set noop scheduling policy for the SSD, by writing "echo noop $>$ /sys/block/dev/sdb/queue/scheduler"(in our server, the SSD is represented as sdb in the system). We respectively created the four file systems on the SSD and ran the workloads on each of them. When we specifically consider certain mount options, we use the default format options. Similarly, when we consider different file system formats, we mount the file system using default mount options. To avoid the cache pollution and previously created files' impacts, we emptied the test file system namespace and umounted/remounted the file system before each run. The operating system is FC8 with linux kernel 2.6.23.

B. Workloads

As said before, one of our primary goals is to investigate how different kinds of workloads would benefit from the high-performance SSD. We used FileBench[23], a very flexible open-source benchmarking tool that can emulate a large variety of workloads to drive our tests. The generated workloads are derived according to the results of comprehensive analysis and statistics of realworld workloads, so they can well represent realistic workloads. We selected four server workloads as our target applications, including Fileserver, Webserver, Mailserver and Database(OLTP). Filesever emulates a simple file-server I/O activities and is similar to SPECsfs. Each of the users is emulated by a thread. Each thread performs a sequence of creates, deletes, appends, reads, writes and attributes operations within a different directory tree. It exercises both the meta-data and data-paths of the file systems. Webserver emulates a simple web-server I/O activities. Each individual thread conducts a sequence of open-read-close on multiple files in a directory tree. At the same time, all of the threads share a common log file and contends to append to the log file. It features sequential read-intensive operations. Mailserver is somewhat similar to Postmark but multi-threaded. It emulates I/O activity of a simple mail server that stores each e-mail in a separate file. The workload consists of a multi-threaded set of create-append-sync, read-append-sync, read and delete operations in a single flat directory. Database server performs file system operations using Oracle 9i I/O model. It tests the performance of small random reads and writes, and is sensitive to the latency of moderate size (128k+) synchronous writes to the log file. By default it launches 200 reader processes, 10 processes for asynchronous writing, and a log writer. All of them report IOPS at the end of the running. More details of the workloads are referred to reference[23].

V. Empirical Evaluation and Analysis

This section presents the empirical study results. For the convenience of exposition in the figures, we use the following abbreviations. We use ext2, ext3, xfs and rfs to denote ext2, ext3, xfs and reiserfs, respectively, and *blk* and *isz* to refer to *block size* and *inode size*,respectively. For instance, *ext2-blk2k* and *xfs-isz1k* refers to ext2 file system with 2KB block size and XFS file system whose inode size is set to 1KB, respectively. And *agcnt* is short for *allocation group count* and is specific to XFS file system. Others mainly refer to different mount options, e.g., *noatime* and *wrbck* indicate the file systems were mounted with *noatime* and *data=writeback* options, respectively. For comparison reasons, we also report the performance of default ext3 file system on the ordinary HDD which was already available on our test server in the last column of each of the figures.

A. Fileserver Performance

Figure 1 shows the file server performance on different file systems. As clearly demonstrated, SSD far outperforms HDD for all of the examined workloads, sometimes even with up to one order of magnitude performance difference. Even for the same SSD, the examined file systems exhibit widely different performance. Among all of them, *rfs-blk1k* has the worst performance, while *rfs-wrbck* performs the best and is more than three times better than *rfs-blk1k*. Except for XFS file system, changing block size from default 4KB to smaller sizes generally causes degraded performance. For example, *ext2-4KB* outperforms *ext2-1KB* and *ext2-2KB* by 22.7% and 29.2%, respectively, and *ext3-4KB* outperforms *ext3-1KB* and *ext3-2KB* by 107.6% and 185.6%, respectively. The reason why 4KB file systems performs better than smaller block size file systems is that when the file systems block size is the same as the SSD page size(the page size of our SSD is 4KB), the SSD would deliver best performance. This is because writes matching the underlying page size would generally avoid partial page updating/overwriting and

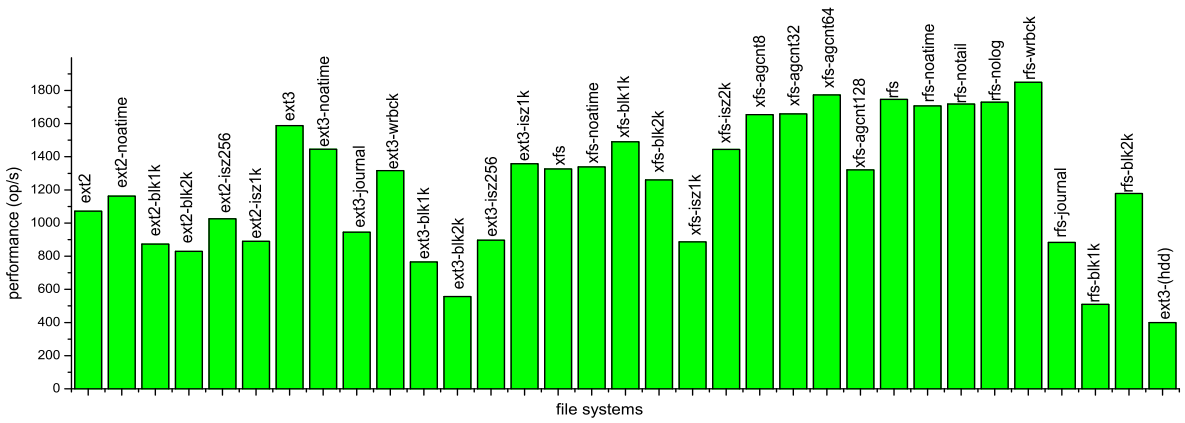

Figure 1. Fileserver performance on different file systems.

correspondingly improve garbage collection efficiency[14]. Another two interesting and counter-conventional-wisdom findings are that: (1) Default ext3-4KB file system is much better than default ext2-4KB file system. Ext3-4KB obtains a performance of 1588.266 IOPS, while ext2-4KB only shows a performance of 1071.571 IOPS. The reason why the ext3 journaling layer helps the performance instead of hurts the performance is that write requests are reported to be finished as soon as they are written to the JBD layer, which is used in a log-structure way. Due to its inherent characteristics[12], [14], SSD would exhibit better performance when used in a log-style way; (2) For fileserver workload, *noatime* mount option exhibits different effects on the four file systems. For *ext2* and *xfs* file systems, the *noatime* helps improving the performance slightly, while for *ext3* and *rfs* it caused marginally degraded performance. We suspect that the saved updating time field does not have much effects on the logging file systems. In addition, inode size also has significant impacts on the performance. For instance, *ext3-isz1k* is 1.5X better than *ext3-256B* and *xfs-isz2k*'s performance is 1.63 times that of *xfs-isz1k*. Allocation group is a unique feature to XFS file system. Intuitively, the more allocation group the file system has, the better performance it would deliver. However, according to our empirical results, allocation group exhibits a more complicated story and shows no consistent tendency. The performance of 16, 32, 64 and 128 allocation group are 1326.598, 1658.674, 1773.029 and 1320.963, respectively. We attribute these performance dynamics to the rich parallelism within modern SSD[20]. For smaller degree of allocation group(16 allocation group), each allocation group is relatively larger and it is very likely that the workload only exercises only part of one certain allocation group which occupies only one plane, for example, and the performance suffers. For moderate degree of allocation group(32 and 64 allocation group), the working files are distributed among the allocation groups and the internal parallelism helps to improve performance. But when the allocation group is too large, then parallelism contribution disappears due to the aggressive contention for shared resources, like shared data buses, connecting circuitry.

B. Webserver Performance

Figure 2 shows the webserver performance on different file systems. As we can see from the figure, unlike the file server, all of the file systems exhibit similar and comparable performance. The main reason behind the scene is that Webserver is a read-intensive workload(the read/write ratio is 10:1) and SSD would deliver superior read performance. Another reason is that all of the write requests of webserver are appends. In other words, webserver would present a very friendly sort of access activities to SSD. The most distinct exception of webserver is that HDD unexpectedly outperforms some of the other file systems, demonstrating that if presented with friendly access patterns, HDD is comparable to SSD in terms of performance, which is consistent with previous findings[38]. The reason is assumed to its relatively smaller working set and most of the I/Os are served by the cache and buffer layer.

C. Mailserver Performance

Figure 3 shows the mailserver performance on different file systems. As it is shown, SSD again outperforms HDD for all of the examined file systems. On the whole, all of *ext2*, *ext3* and *rfs* perform better than XFS and most of the time their performance are two times that of XFS. This demonstrates that XFS has problems in dealing with large number of small files. Similar to fileserver, changing the default 4KB block size to 1KB would also cause degraded performance. However, unlike fileserver, 2KB block size file systems perform comparably to the 4KB counterparts. For

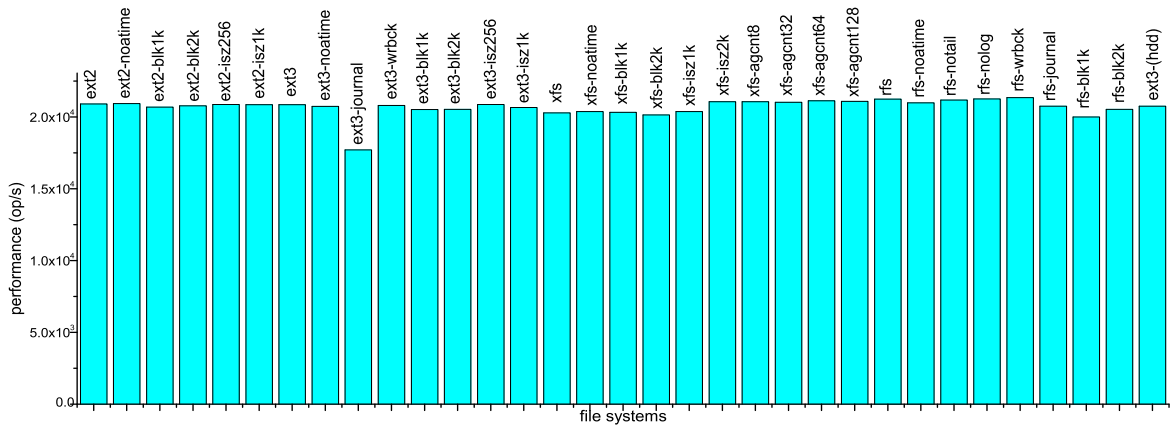

Figure 2. Webserver performance on different file systems.

example, *ext2-4KB* is 1.88X that of *ext2-1KB* and *ext3-4KB* is 1.39X that of *ext3-1KB*. In contrast to the file server, *noatime* mount option slightly improves the performance of all of the file systems. Additionally, the performance is not that sensitive as fileserver to inode sizes. For example, *ext2-isz256B* and *ext3-isz256B* perform comparably to *ext2-isz1K* and *ext3-isz1k*. For journaling file systems, journal mode outperforms other journaling mode. Specifically, *ext3-journal* delivers 4557.366 IOPS, while *ext3-wrbck* and *ext3-ordered* exhibit 3469.217 and 3446.273 IOPS, respectively; *rfs-journal* delivers 4195.084 IOPS, while *rfs-wrbck* and *rfs-ordered* exhibit 3303.183 and 3423.212 IOPS, respectively. For reiserfs, *notail* mount option improve performance by 6.2%, demonstrating that packing small files into inode size doesn't necessarily always help the performance. This time for *xfs*, the number of allocation group seems to have minimal impacts on performance, which is different than that of fileserver.

D. Databaseserver Performance

Databaseserver has a much smaller file set than other workloads, and is an even more aggressively read-intensive feature(its read/write ratio is 20:1). It also features high-concurrency operations. Its performance on different file systems are shown in figure 4. As demonstrated in that figure, SSD beats HDD for all of the examined file systems and the most outstanding behavior from other workloads is that decreasing the default 4KB block size to 2KB improves the performance significantly. For example, for *ext2*, *ext3* and *reiserfs*, decreasing the block size from 4KB to 2KB improves the performance by 17.3%, 23.6% and 24.4%, respectively, which is contrary to other workloads. The reason is that by default databaseserver issues 2KB IO requests and other workloads generate much bigger IO requests, e.g., 16KB. In other words, file system block size is more sensitive to performance than the underlying SSD page size and when the file system block size matches the IO size, the performance is the best. For *reiserfs*, as opposite to other workloads, *nolog* mount option improves performance due to two reasons. First, databaseserver is a much more read-intensive workload, which means that the logging region itself is seldom used and would get minimal benefits due to the absence of write requests. Second, databaseserver works in a very small set of files and due to the SSD inherent log-style-aware page-allocation policy, the interferences of background activities are well suppressed. Though webserver also has a very high read/write ratio, but webserver's write patterns are only appends which causes similar performance behaviors. Similar to mailserver, the allocation group also has minimal effects on performance. The possible reason is that in XFS files are allocated within allocation groups and databaseserver has only a smaller number of files. Thus, a small number of allocation group is sufficient to satisfy the parallelism requirement and increasing the allocation group further does not improve performance correspondingly.

E. Different I/O Sizes

To investigate how I/O request size impacts performance, we used varying I/O sizes of database server workloads to test those file systems formatted with different block sizes. We used 1KB, 2KB, 4KB, 8KB I/O sizes and 4KB, 1KB, 2KB file system block sizes. Figure 5 shows the performance of different configurations. The x-axis represents the file systems with different block sizes and y-axis indicates the performance. From that figure, we know that when the file system block size matches the I/O request size, the performance is consistently better than the other combinations. For example, for *ext2-4KB* file system, 4KB request workloads enjoys the best performance, beating 1KB, 2KB and 8KB by 11.3%, 17.8% and 16.9%, respectively. The reason why the matched block size and request size outperforms other options is that in this scenario the read-modify-write

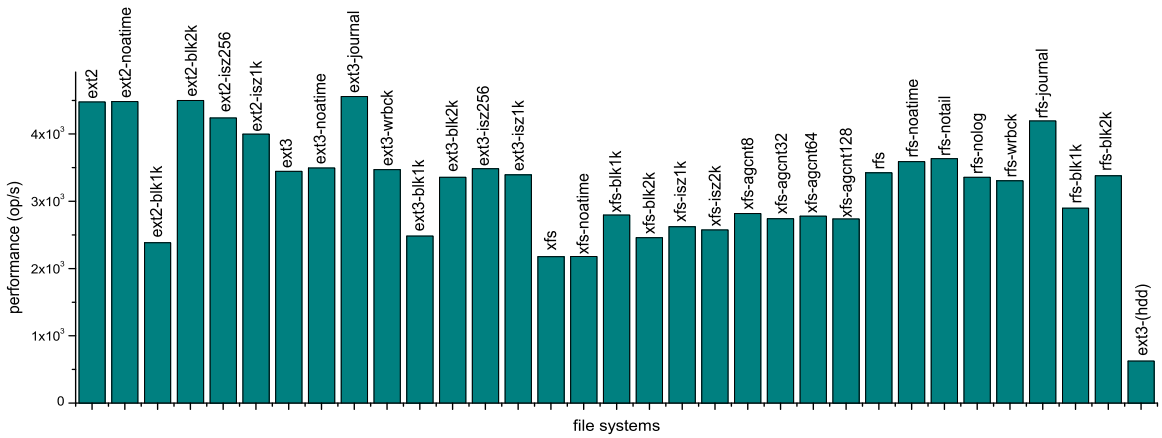

Figure 3. Mailserver performance on different file systems.

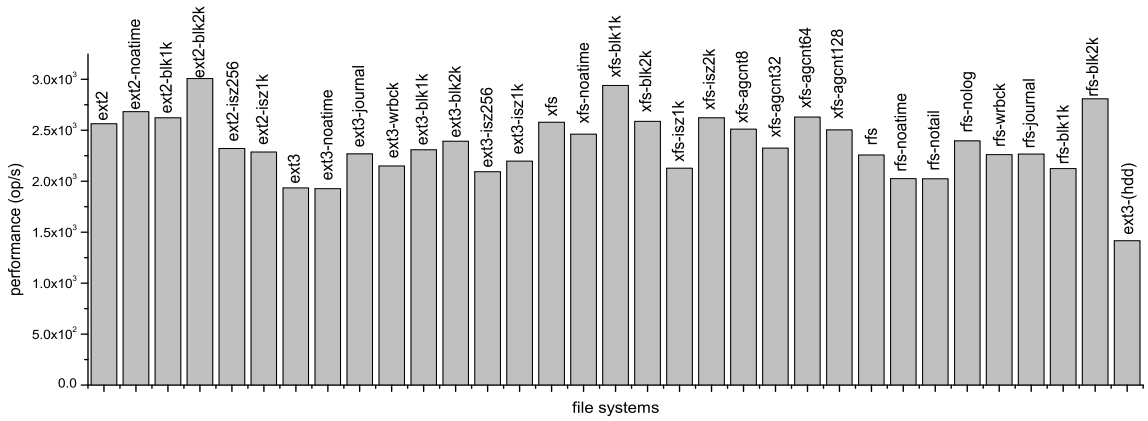

Figure 4. Databaseserver performance on different file systems.

operations are minimized, as mentioned in Section V-A. Furthermore, when block size, request size and SSD page size are consistent, the performance is even better. Take *xfs* for example, the 4KB workload on *xfs-4KB* delivers 3122.82 IOPS, while the 1KB on *xfs-1KB* delivers 2939.766 IOPS and the 2KB on *xfs-2KB* delivers 2587.357 IOPS, demonstrating that when those three impacting factors are the same, the performance would be the best.

VI. Summary and Conclusion

In this study, we conducted extensive empirical experiments on the performance of a variety of workloads on four different file systems in the context of an SSD. We have found the following observations: (1) Different file systems have disparate performance behaviors, even for the same workload. Also, different format and mount options have significant impacts on performance. (2) Different workloads exercise different paths of file systems, making certain file systems perform better for certain workloads and work worse for other workloads. Unlike previously reported raw SSD behaviors, file systems internal caches and optimization techniques would optimize the access patterns presented to SSD, thus improving performance. (3) On the one hand, it's well-known that SSD would provide certain superior performance, e.g., read performance. On the other hand, SSD also exhibits complicated and unpredictable performance dynamics. For example, if properly deployed, the rich set of parallelism within SSD would accelerate performance greatly, however, if too aggressively deployed, the performance would slow down rather than speed up. (4) Workload I/O size, file system block size and SSD internal page size affect the overall performance in an intrinsical manner. File system block size is more sensitive to performance than SSD page size. That said, when the file system block size is equal to I/O size, the performance is optimal. But when both of them again equal to SSD page size, the performance could

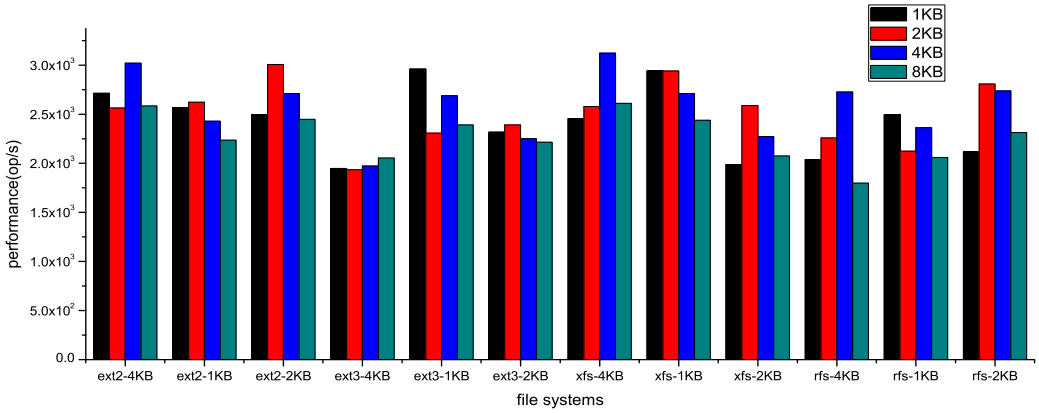

Figure 5. The performance of different I/O sizes of database server in different file systems.

be further improved.

When setting up the storage system, to better use SSD in the storage stack, users should consider a combination of workloads characteristics, intended usage scenarios, expected performance behaviors and file system peculiarities. For read-intensive and append-only workload, there are few optimization opportunities, most of them perform equally and sometimes the SSD advantages disappear and is inferior to HDD counterpart. For workloads that need to deal with large number of small-sized files, XFS is not the right choice. For XFS, it is wise to configure the number of allocation group to be moderate to obtain the potential benefits provided by the SSD internally parallel structure. Also, it is quite beneficial to keep the average I/O size, file system block size and SSD page size consistent. In summary, the SSD alone can not guarantee overall optimal high-level performance expectation, though it features superior raw performance. Only taking all of the impacting factors can we really make the best of SSD in reality.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their constructive feedbacks and suggestions, which improve the paper quality significantly. This paper was supported in part by National Basic Research Program (973 Program) of China under Grant No.2011CB302305 and National Key Technology R&D Program under Grant No.2012BAH35F03.

REFERENCES

[1] A. M. Caulfield, L. M. Grupp, and S. Swanson, "Gordon: using flash memory to build fast, power-efficient clusters for data-intensive applications," in *Proceedings of 14th International Conference on Architectural Support for Programming Languages and Operating Systems(ASPLOS'09)*, 2009.

[2] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan1, "Fawn: A fast array of wimpy nodes," in *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP'09)*, 2009.

[3] R. Pai, B. Pulavarty, and M. Cao, "Linux 2.6 performance improvement through readahead optimization," in *Proceedings of the Linux Symposium*, 2004.

[4] S. Jiang, F. Chen, and X. Zhang, "Clock-pro: An effective improvement of the clock replacement," in *Proceedings of 2005 USENIX Annual Technical Conference*, 2005.

[5] S. Jiang, X. Ding, F. Chen, E. Tan, and X. Zhang, "Dulo: An effective buffer cache management scheme to exploit both temporal and spatial locality," in *Proceedings of the 4th USENIX Conference on File and Storage Technologies(FAST'05)*, 2005.

[6] X. Ding, S. Jiang, F. Chen, K. Davis, and X. Zhang, "Diskseen: Exploiting disk layout and access history to enhance i/o prefetch," in *Proceedings of 2007 USENIX Annual Technical Conference*, 2007.

[7] B. Debnath, S. Senguptaz, and J. Li, "Chunkstash: Speeding up inline storage deduplication using flash memory," in *Proceedings of 2010 USENIX Annual Technical Conference*, 2010.

[8] A. Badam and V. S. Pai, "Ssdalloc: Hybrid ssd/ram memory management made easy," in *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation(NSDI'11)*, 2011.

[9] A. Anand, C. Muthukrishnan, S. Kappes, A. Akella, and S. Nath, "Cheap and large cams for high performance data-intensive networked systems," in *Proceedings of the 7th Symposium on Networked Systems Design and Implementation (NSDI'10)*, 2010.

[10] J. Ren and Q. Yang, "I-cash: Intelligently coupled array of ssds and hdds," in *Proceedings of the 17th IEEE International Symposium on High Performance Computer Architecture(HPCA'2011)*, 2011.

[11] F. Chen, D. Koufaty, and X. Zhang, "Making the best use of solid state drives in high performance storage systems," in *Proceedings of the 25th International Conference on Supercomputing(ICS'2011)*, 2011.

[12] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for ssd performance," in *Proceedings of USENIX Annual Technical Conference*, 2008.

[13] P. Huang, K. Zhou, and C. Wu, "Shiftflash: Make flash-based storage more resilient and robust," *Performance Evaluation*, vol. 68, no. 11, pp. 1193–1206, 2011.

[14] S.-W. Lee and B. Moon, "Design of flash-based dbms: an in-page logging approach," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data(SIGMOD'2007)*, 2007.

[15] H. Kim and S. Ahn, "Bplru: A buffer management scheme for improving random writes in flash storage," in *Proceedings of the 6th USENIX Conference on File and Storage Technologies(FAST'2008)*, 2008.

[16] V. Prabhakaran, M. Balakrishnan, J. D. Davis, and T. Wobber, "Depletable storage systems," in *Proceedings of the HotStorage'2010*, 2010.

[17] G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber, "Extending ssd lifetimes with disk based write caches," in *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST'2010)*, 2010.

[18] J. B. Layton, "Nilfs: A file system to make ssds scream. http://www.linux-mag.com/id/7345/."

[19] F. Chen, D. A. Koufaty, and X. Zhang, "Understanding intrinsic characteristics and system implications of flash memory based solid state drives," in *Proceedings of SIGMETRICS/Performance'2009*, 2009.

[20] F. Chen, R. Lee, and X. Zhang, "Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing," in *Proceedings of the 17th IEEE International Symposium on High Performance Computer Architecture(HPCA'2011)*, 2011.

[21] L. M. Grupp, A. M. Caulfield, J. Coburn, and S. Swanson, "Characterizing flash memory: Anomalies, observations and applications," in *Proceedings of the 42nd International Symposium on Microarchitecture(MICRO'2009)*, 2009.

[22] B. Yoo, Y. Won, S. Cho, S. Kang, J. Choi, and S. Yoon, "Ssd characterization: From energy consumptions perspective," in *Proceedings of HotStorage'2011*, 2011.

[23] P. Sehgal, V. Tarasov, and E. Zadok, "Evaluating performance and energy in file system server workloads," in *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST'10)*, 2010.

[24] J. Hu, H. Jiang, L. Tian, and L. Xu, "Pud-lru: An erase-efficient write buffer management algorithm for flash memory ssd," in *Proceedings of the 18th Annual Meeting of the IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems(MASCOTS'10)*, 2010.

[25] E. Gal and S. Toledo, "Algorithms and data structures for flash memories," *ACM Computing Survey*, vol. 37, no. 2, pp. 138–163, 2005.

[26] Y. Hu, H. Jiang, L. Tian, H. Luo, and D. Feng, "Performance impact and interplay of ssd parallelism through advanced commands, allocation strategy and data granularity," in *Proceedings of the 25th International Conference on Supercomputing (ICS'11)*, 2011.

[27] D. P. Bovet and M. Cesati, *Understanding the Linux Kernel*, A. Oram, Ed. O'Reilly Media,Inc, 2005.

[28] "http://oss.sgi.com/projects/xfs/."

[29] "http://en.wikipedia.org/wiki/reiserfs."

[30] A. Traeger, E. Zadok, N. Joukov, and C. P. Wright, "A nine year study of file system and storage benchmarking," *ACM Transactions on Storage (TOS)*, vol. 4, no. 2, pp. 25–80, May 2008.

[31] "http://sources.redhat.com/jffs2/."

[32] "http://www.yaffs.net/."

[33] H. H. Huang, S. Li, A. Szalay, and A. Terzis, "Performance modeling and analysis of flash-based storage devices," in *Proceedings of the 27th IEEE Symposium on Massive Storage Systems and Technologies(MSST'2011)*, 2011.

[34] S. Boboila and P. Desnoyers, "Write endurance in flash drives: Measurements and analysis," in *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST'2010)*, 2010.

[35] A. Gupta, Y. Kim, and B. Urgaonkar, "Dftl: a flash translation layer employing demand-based selective caching of page-level address mappings," in *Proceedings of 14th International Conference on Architectural Support for Programming Languages and Operating Systems(ASPLOS'09)*, 2009.

[36] Y. Hu, H. Jiang, D. Feng, L. Tian, S. Zhang, J. Liu, and W. Tong, "Achieving page-mapping ftl performance at block-mapping ftl cost by hiding address translation," in *Proceedings of the 26th IEEE Symposium on Massive Storage Systems and Technologies(MSST'2010)*, 2010.

[37] S. Jiang, L. Zhang, X. Yuan, H. Hu, and Y. Chen, "S-ftl: An efficient address translation for flash memory by exploiting spatial locality," in *Proceedings of the 27th IEEE Symposium on Massive Storage Systems and Technologies(MSST'2011)*, 2011.

[38] M. Polte, J. Simsa, and G. Gibson, "Comparing performance of solid state devices and mechanical disks," in *Proceedings of the 3rd Petascale Data Storage Workshop held in conjunction with Supercomputing(PDSW'2008)*, 2008.

[39] E. Seppanen, M. T. O'Keefe, and D. J. Lilja, "High performance solid state storage under linux," in *Proceedings of the 27th IEEE Symposium on Massive Storage Systems and Technologies(MSST'2011)*, 2011.