INFOTEH-JAHORINA Vol. 15, March 2016.

Performance Analysis of 64-bit ext4, xfs and btrfs filesystems on the Solid-State disk technology

Jelena Kljajić, Nada Bogdanović,
Marko Nankovski, Marjan Tončev
School of electrical engineering, University of Belgrade
Mihajlo Pupin Institute, University of Belgrade
Belgrade, Serbia
jelena.kljajic@pupin.rs

Borislav Djordjević
Mihajlo Pupin Institute, University of Belgrade
Belgrade, Serbia

bora@impcomputers.com

*Abstract***— In this paper a comprehensive performance analysis of three commonly used journaling filesystems, xfs, ext4 and btrfs, on a solid-state drive (SSD) is undertaken. It has been proved that solid-state drives are superior to traditional magnetic discs [1]. Successful methods for existing HDDs might not be fully suitable for SSDs. It is thus important to determine which filesystem meets the requirements of the new technology mostly. Performances of the selected filesystems were compared using random and sequential benchmark tests. The results show the capability of the young btrfs to become the default filesystem on many Linux distributions.**

Keywords- ***ext4; xfs; btrfs; filesystem; solid-state drive; Linux***

I. Introduction

With the increase in the presence of computers in everyday life and growing amount of information that is kept, it is of paramount importance to properly organize the stored data. The data is handled by local storage devices. Filesystems are used to systemize that data.

Two types of disk drives are commonly in use: magnetic hard disk drives (HDD) and solid-state disk drives (SSD). Storing, retrieving, caching and cleaning up data define the overall performance of each disk drive. The standard hard drive (HDD) has been the predominant storage device for a long time. Its greatest advantages are the storage size and low cost. However, solid state drives (SSD) tend to replace them as faster and more reliable.

Filesystems are used to control how a hard drive stores, accesses and manages files. They organize data placed in a storage area by separating and organizing it, usually into multiple physical units on the device, and identifying it. Some of the most common filesystems for Linux, and the ones presented in this paper are xfs, ext4 and btrfs.

Recently, it has been the goal of commercial as well as scientific research to develop an optimal Linux kernel. It is important to provide additional features that allow Linux to scale to larger amounts of storage. [2] With that on mind, our idea was to consider filesystems that are currently in use and have the potential for improvement, and compare their performance on a SSD.

In the following section, solid state disk technology is briefly presented. Section III, gives an overview of the three commonly used Linux filesystems. The fourth section includes experimental results and comparative performance analysis of all three filesystems considered. The report is rounded off with a conclusion.

II. Solid State Drive

Solid State Drives (SSD) have been introduced as a transformative solution for computer storage systems. Their most important feature is the outstanding performance for random data access. [3] In addition, they are compact in size and shock resistant, which makes more reliable in comparison to traditional magnetic disks.

SSD are based on NAND flash memory, designed for data storage with greater capacity, which only allows access in blocks. [1] It is organized in layers in the order of block, page, row, and cell. The write unit is a page and the erase unit is a block. [4]

Most solid state disks have the same host interface as in hard disks drives. However, under this common interface, their mode of operation is much more different. [1]

Since they are assembled from semiconductor chips and have no rotating parts, solid state disks provide lower seek times and fewer mechanical delays. Therefore, read latency of SSD is negligible. [4]

Delay occurs in random write. Hard disk drives can be written and re-written many times. Erasing on magnetic storage is performed by simply overwriting with new data. On the other side, writing data in solid state technology requires two stages: erase and write.

Internal organization of the flash memory determines that old data must be erased before blocks can be reused. On update of the present data, the content of the entire block is copied into a new location. Then, the old block is deleted. Updated contents of the old block are then written to the new block. The process of removing temporary blocks that are not used any more is called garbage collection.

SSD encounter an additional problem concerning flash memory. Flash memory has a limited number of erase cycles, after which memory cells cease to hold data. There are three main strategies to overcome this problem. The first is to write on different memory blocks on every new input, tending to use blocks equally for their lifetime extension. This is called wear leveling. [4] The other ones include occasionally moving files that are not used often to other memory blocks and having additional memory blocks to replace expired memory cells.[3]

In conclusion, Solid State Drives are becoming a new standard for data storage, replacing hard disks in many modern devices. This has motivated an ongoing effort to optimize filesystems according to their architecture. [5]

III. Linux filesystems

A. *xfs*

Xfs is a 64-bit filesystem developed by Silicon Graphics, Inc. in 1993. It was first used on IRIX 5.3 and ported to Linux in 2001. In 2002. it was first added to the Linux kernel version 2.4. Starting from June 2014, Red Hat Enterprise Linux (RHEL) 7.0 uses xfs as the default system. [6] Today it is the default filesystem for Red Hat Enterprise Linux 7. [7]

Xfs is a high performance journaling filesystem, which means that it maintains a log, or journal, of activity that has taken place in the main data areas of the disk. Updates to the metadata in directories and bit maps are written to a serial log, so that any lost data can be recreated. This provides a protection in case of a system crash or power failure. [8]

All of the filesystems presented in this paper use allocation for setting aside the space on a hard drive for storing files. Those files can be either ones already modified or newly created. Xfs supports the following allocation methods: extent-based allocation, stripe-aware allocation policies, space pre-allocation and delayed allocation. There are several advantages to the delayed allocation method:

1. Larger sets of blocks are processed before being written, which reduces the processor utilization.

2. A large number of blocks that are most likely contiguous are allocated at once – this reduces fragmentation.

3. Delayed allocation reduces processor time and disk space for files that are short-term temporary files used and deleted in cache before being written.

Delayed allocation is probably the best method for files where the file size is unknown at the time of writing, usually because they are still being created or modified at the time. [9]

For preventing fragmentation and increasing performance, besides delayed allocation method, xfs uses sparse files. If the real file contains large sections of zeroes, metadata will be written instead of all the zeroes, so the space can be saved. When accessed again, the file is expanded to its normal state in memory.

Even with the mentioned methods for reducing fragmentation, it can still occur in case of a low free space. To lessen this issue, xfs uses online defragmentation. In this process, files can be moved into contiguous blocks to reduce fragmentation. xfs can be defragmented and enlarged while mounted and active.

For allocating space on the filesystem, xfs uses extents. Managing the free space on the filesystem is accomplished by using B+ trees for tracking these spaces. [6]

B. *ext4*

Ext4 stands for "fourth extended filesystem" and was developed as a scalable extension of the ext3 filesystem. It was introduced in 2008. The ext2 and ext3 filesystems are based on an indirect block mapping scheme, known to be very efficient for small files, but not so for larger ones. Instead of its predecessors, ext4 introduces extents, which improve performance and reduce metadata overhead for large files. Extents are basically descriptors that represent a range of contiguous physical blocks. In ext4, files can allocate extents instead of individual blocks. [10] [11]

Similar to the ext3, the ext4 is also a journaling filesystem for Linux, with additional journal checksums. This way, a disk I/O wait during journaling can be avoided, which results in quicker crash recovery, and in that way, an improvement in performance.

As for the faster filesystem checking, ext4 also labels unallocated block groups and inode table sections, which allows them to be skipped during a filesystem check.

The ext4 features the following allocation schemes: persistent pre-allocation, delayed allocation, multi-block allocation and stripe-aware allocation. Delayed allocation is also known as allocate-on-flush, and is used by both ext3 and ext4 filesystem. The main difference is that ext3 automatically writes newly created files to disk almost immediately. Meanwhile, the ext4 often waits several seconds to write out changes to disk. This allows the ext4 filesystem to reduce fragmentation and improve performance comparing to ext3.

C. *btrfs*

Btrfs, “B-Tree File System” or “Better F S”, was created by Oracle in 2007. and merged into the mainline Linux Kernel 2.6.29 in 2009.

The principal data structure for btrfs is a B+ tree. It consists of a root, internal nodes and leaves. The main advantage is that their logarithmic growth in depth enables and improves accessing and updating large blocks of data no matter how large the tree grows. B+ trees are also used in xfs filesystem. The main difference between those two is that in btrfs, in-place modification is avoided by copying the processed data to a new location. The benefit is faster crash recovery.

Btrfs implements implicit sharing, otherwise known as Copy-on-Write (CoW). It is used for handling resources when multiple tasks are using the same data. Usually, when an application requests data from a file, the data is sent to memory or cache, meaning that each application then has its own memory space. In that case, each application has its own memory space. In order to save the space, if multiple applications request the same data, that data is then allocated in one place and pointed to by all the applications. In case one of the applications needs to change the data, then that application will be given its own memory space with the new updated information, while the other applications continue using the older pointers with original data.

It also features two types of compression: LZO and ZLIB, which are used for preventing the filesystem from becoming full. The LZO method produces smaller files, while ZLIB compresses faster. If a disk space becomes full, btrfs is capable of online volume growth and shrinking or online defragmentation. This functionality improves performance.

Btrfs includes support for redundancy (duplicating parts to prevent a failure as a whole) and fault tolerance (the ability of a computer system to continue working after a failure). For that, btrfs uses RAID, specifically: RAID 0, RAID 1 and RAID 10; RAID 5 and RAID 6 are considered experimental features. [12]

If necessary, filesystem snapshots enable a system to roll back to a prior state, or they can be used to back up files. [13]

Another btrfs feature is checksum functionality, which improves error detection and ensures data integrity.

D. Filesystem hypothesis

For every workload applies

$$T_{workload} = T_{Dir} + T_{Meta} + T_{FL} + T_{FB} + T_{J} + T_{HK} \quad (1)$$

where $T_{workload}$ is the total time needed to complete all operations on the workload.

T_{Dir} represents the time needed to complete all directory related operations, T_{Meta} the time needed to complete all metadata operations, T_{FL} the time needed to complete all free lists operations, T_{FB} the time needed to complete direct file blocks operations, T_{J} the time needed to complete journaling operations and T_{HK} the time needed to complete housekeeping operation within the filesystem.

All three filesystems of interest are based on an extent. The greatest differences between them are in the directory organization. ext4 filesystem directories are in the form of H-Trees, while xfs and btrfs directories are in the form of B+ trees.

B+ tree is the dominant structure for all areas of filesystems management in xfs and btrfs. Computational complexity for the B+ tree for all 4 types of actions (insertion, retrieval, updating, deleting) is:

$$T_{B+Tree}(mngmnt) \approx O(\log(n)) \quad (2)$$

where,

$$T_{B+Tree}(mngmnt) = Function(indexes, keys, nodes, balancing) \quad (3)$$

Another difference between them is in the write cycles. Ext4 and xfs use the overwrite method,

$$T\ (writting) = Function(overwrite\ \_overhead) \quad (4)$$

while btrfs filesystem uses *Copy-on-Write* (CoW) update method, which causes the migration of data and increase in the amount of data to be written.

$$T\ (writting) = Function(CoW\ \_overhead) \quad (5)$$

For btrfs filesystem there is the greatest housekeeping time, as it consumes checksum both for metadata and data operations, which additionally lengthens the write cycles.

IV. Experimental Results and Analysis

A. Hardware Configuration

The specification of used hardware is presented in table I. Tests were performed on Ubuntu Linux operating system using SSD [14], whose specification is shown in tables II.

TABLE I. HARDWARE SPECIFICATION

Hardware	Specification
RAM	8 GB
CPU Model	Intel(R) Core(TM) i5-4690 CPU @ 3.50GHz
Number of CPU Cores	4
Solid-State Drive	Transcend, TS128GSSD370S, 128GB, 2.5"
Operating System	Ubuntu 14.04.1, kernel – Linux 3.13.0-32 - generic x86_64

TABLE II. SOLID-STATE DRIVE SPECIFICATION

SSD	Specification
Model	Transcend, TS128GSSD370S, 128GB, 2.5"
Capacity	128 GB
Interface	Serial ATA III
Transfer Rate to Host	6 Gb/s
Storage Media	Synchronous MLC NAND Flash memory
Controller	Transcend TS6500
Buffer	None
Max. Read	550 MB/s
Max. Write	170 MB/s

B. Results

The results are obtained using Bonnie++ benchmark program. It is C++ software used for evaluating performance of different storage units and filesystems. Results of performance analysis can be divided into two groups: random (Fig. 1) and sequential (Fig. 2). Fig. 1a depicts the random write test results obtained using *putchar()* function and Fig. 1b shows the random read test results obtained using *getchar()* function.

The results are expected. btrfs is conceivably the best in the random read test, as a result of acceleration coming from B+ trees in many filesystems metadata and data structures by formulas (2) and (3), whereby for this workload B+ tree technology for btrfs works better than B+ trees technology for xfs. In the random write test, xfs and ext4 have similar performance, while btrfs is noticeably weaker because of CoW update method, according to the formula (5), as well as time needed for housekeeping.

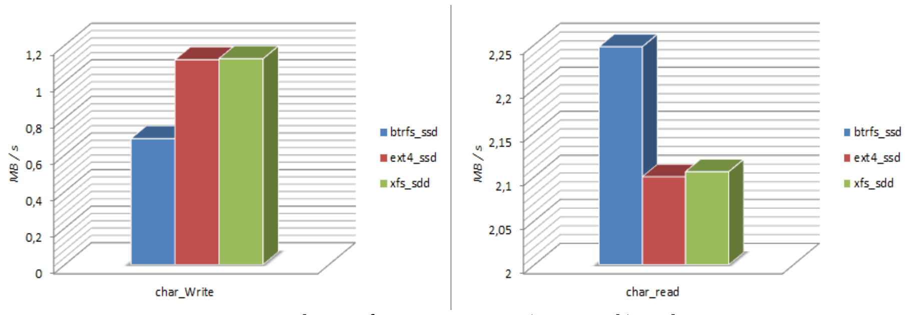

Fig. 1 – Random performance testing: a) writing; b) reading.

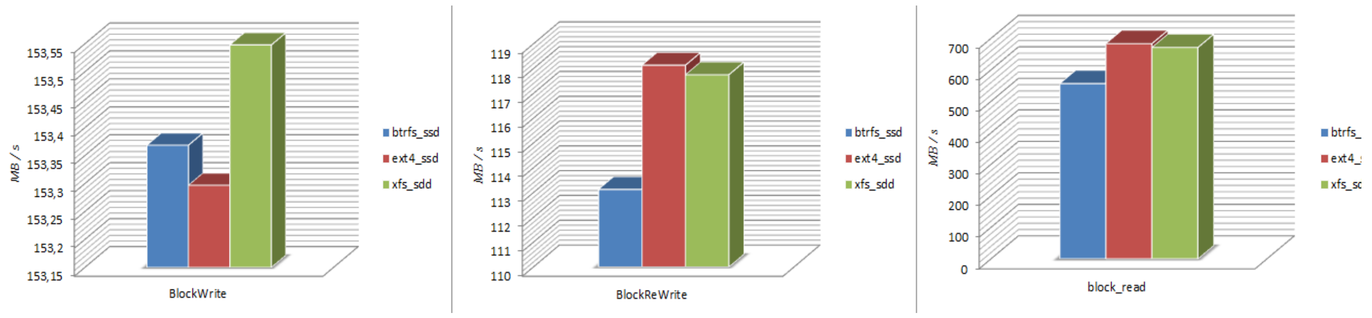

Fig. 2 – Sequential performance testing: a) writing; b) read-modify-write; c) reading.

Fig. 2 represent sequential performance of the drives, obtained using *getblock()* function. Interesting result is that btrfs is considerably weaker in all cases of sequential testing except in the case of sequential write when compared to ext4.

Large sequential write transfers produce much CoW data traffic, which considerably slows down btrfs filesystem according to the formula (5).Considering all sequential test cases under the Ubuntu Linux, the best performances are generally obtained using xfs filesystem, due to the B+ Tree structure according to formulas (1) and (2), as well as overwrite and update method according to the formula (4).

V. Conclusion

The purpose of this paper was to compare the performances of three 64 bit filesystems using modern SSD technology, under Ubuntu Linux OS. Expected results were that the most modern filesystem, btrfs, will have significantly better performances for all read tests, but it was detectable only for random read, and not for sequential read as well. It was expected that the btrfs has a lower write performance due to the CoW method, which was confirmed in all tests. On the other hand, two overwrite update filesystems, ext4 and xfs, exhibited very similar performance. We can note that the btrfs filesystem is still not fully accepted for Ubuntu Linux distributions, as it is not in the recommended choices during the installation procedures, but has to be installed additionally. On the other hand, there are numerous papers that prove quality performance of the btrfs filesystem on the other Linux distributions, like Centos, etc. in which the btrfs is present in the basic offer on the install.

References

[1] F. Cheng, D. A. Koufaty, X. Zhang, *"Understanding Intrinsic Characteristics and System Implications of Flash Memory based Solid State Drives"*, ACM SIGMETRICS Performance Evaluation Review, Vol. 37 Issue 1, June 2009.

[2] *"HTG Explains: What is a File System, and Why Are There So Many of Them?"* (http://www.howtogeek.com/196051/htg-explains-what-is-a-file-system-and-why-are-there-so-many-of-them/)

[3] P. Bednar, V. Katos, *"SSD: New Challenges for Digital Forensics"*, ItAIS 2011, Proceedings of the 8th Conference of the Italian Chapter of the Association for Information Systems In Information Systems

[4] *"How is SSD Changing Software Architecture?"*, http://www.cubrid.org/blog/dev-platform/how-ssd-changing-software-architecture/ [15.01.2015.]

[5] O. Rodeh, J. Bacik, C. Mason, *"BTRFS: The Linux B-tree Filesystem"*, Vol. 9 Issue 3, Article No. 9, August 2013.

[6] *"XFS File System"*, http://www.linux.org/threads/xfs-file-system.4364/ [15.01.2015.]

[7] *"The XFS File System"*, https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Storage_Administration_Guide/ch-xfs.html [15.01.2015.]

[8] *"Journaled filesystem"*, http://www.webopedia.com/TERM/J/journaled_file_system [15.01.2015.]

[9] *"Allocation Methods"*, http://www.linux.org/threads/allocation-methods.4319/ [15.01.2015.]

[10] *"The ext4 File System"*, https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Storage_Administration_Guide/ch-ext4.html [15.01.2015.]

[11] A. Mathur, M. Cao, S. Bhattacharya, *"The new ext4 filesystem: current status and future plans"*, Ottawa Linux Symposium (2007). *http://ols.108.redhat.com/2007/Reprints/mathur-Reprint.pdf* [15.01.2015.]

[12] *"RAID"*, http://www.linux.org/threads/intro-to-raid.4132/ [15.01.2015.]

[13] *"Btrfs"*, https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Storage_Administration_Guide/ch-btrfs.html [15.01.2015.]

[14] *http://www.transcend-info.com/products/images/modelpic/579/No3118_TSXGSSD370S_V10_Datasheet.pdf* [15.01.2015.]