Journal of COMPUTER AND FORENSIC SCIENCES

CFS 2022, Vol. 1, Issue 1, pp. 55–75
https://doi.org/10.5937/1-42712
Original research paper

Performance Modeling of File System Pairs in Hypervisor-Based Virtual Environment Applied on KVM Hypervisor Case Study

Borislav Đorđević[1], Valentina Timčenko[2], Nemanja Maček[3*], Mitko Bogdanoski[4]

[1] Mihajlo Pupin Institute, University of Belgrade, Serbia & Academy of Technical and Art Applied Studies, School of Electrical Engineering, Belgrade, Serbia; borislav.djordjevic@pupin.rs

[2] Mihajlo Pupin Institute, University of Belgrade, Belgrade, Serbia; valentina.timcenko@pupin.rs

[3] Academy of Technical and Art Applied Studies, School of Electrical and Computer Engineering, Belgrade, Serbia & University Business Academy in Novi Sad, Serbia & SECIT Security Consulting, Serbia; macek.nemanja@gmail.com

[4] Military Academy General Mihailo Apostolski, Skopje, RN Macedonia, mitko.bogdanoski@ugd.edu.mk

* Corresponding author: macek.nemanja@gmail.com

Received: 2022-10-08 • Accepted: 2022-12-05 • Published: 2022-12-30

Abstract: This paper proposed an approach to mathematical modeling of the file system performance in a hypervisor-based virtual environment, with special focus on the file system pair interactions. The main goal of this research is to conduct an in-depth analysis of the filesystem pair behavior with respect to the performance costs originating from the employed technologies, such as H-Trees, B-Trees and Copy-on-Write/Overwrite update method, and different application workload types. The modeling provides a collection of hypotheses about the expected behavior. The modeling and the hypotheses are validated based on the results obtained for a specific case study. Our study reports on a file system performance comparison in the context of KVM hypervisor-based full hardware virtualization, application-level benchmarking, and 64-bit Linux filesystems Ext4, XFS, and Btrfs. The Filebench benchmark tool is applied for comprehensive testing of the filesystem performance under fair-play conditions. According to the obtained results, we provide a set of recommendations (i.e., a Knowledge Data Base) for optimal filesystem pair selection for the KVM hypervisor. Finally, it is important to note that the proposed modeling is also applicable to other hypervisor-based virtualizations.

Keywords: filesystems, operating systems, performance evaluation, platform virtualization, virtual machine monitors.

Citation: B. Đorđević, V. Timčenko, N. Maček, and M. Bogdanoski, "Performance Modeling of File System Pairs in Hypervisor-Based Virtual Environment Applied on KVM Hypervisor Case Study," *Journal of Computer and Forensic Sciences*, 1(1), https://doi.org/10.5937/1-42712.

1. INTRODUCTION

Virtualization is a method for organizing computer resources within multiple operational environments, by means of hardware and software partitioning, time-sharing, partial or complete hardware emulation, paravirtualization, etc. [1–3]. Modern approaches to virtualization include two fundamental methods: hypervisor-based virtualization and container-based virtualization. Virtualization has gained popularity in many different areas, including cloud computing (CC), Internet of Things, cyber physical systems, and big data. It represents the core technology behind proper cloud computing functioning, which mainly depends on the sophistication of its design and implementation.

Quality of Service (QoS) is another important aspect to consider. QoS represents guaranteed levels of performance and availability of a service provided to users [4–6]. A large number of factors affect QoS, but the three primary aspects are computing, storage, and network performance. In a hypervisor-based virtual environment, there are three fundamental components: a host operating system, a hypervisor, and guest operating systems. A host operating system represents the driver support and management layer for virtualization, and a hypervisor is a software layer that serves as an intermediary between the host operating system and virtual machines, i.e., hypervisors behave as kernels for virtualization.

A hypervisor and a host operating system create a virtual environment for guest operating systems. This environment does not necessarily have the same characteristics as the physical environment. In the context of hypervisor-based virtualization, three types of virtualization are dominant: full hardware virtualization, paravirtualization, and operating system (OS) level virtualization. The full hardware virtualization represents complete hardware emulation so that the installation and execution of guest OSs require no additional adaptations. This type of virtualization is the most appropriate for employment, but it suffers from low performance level, which can be boosted by using Intel VT-x or AMD-V as special CPU features for virtualization. Paravirtualization requires significant modifications to the host and guest OSs but enables significantly better hypervisor-based performance of virtual machines. OS-level virtualization is based on applying the same kernel for several OS instances.

There are two types of hypervisors. Type-1 hypervisors (i.e., so-called native hypervisors) execute directly on physical hardware, and their prominent instances include ESXi, Xen, KVM/Proxmox, and Hyper-V. Type-2 hypervisors execute as applications within a host OS, and their prominent instances include Oracle-Virtual Box and VMware Workstation. Normally, type-1 hypervisors have much better performance than type-2 hypervisors. Related to host OSs, type-1 hypervisors can be Linux-based (e.g., ESXi, Xen, KVM/Proxmox), and MS Windows-based (e.g., Hyper-V).

Hypervisor-based virtualization builds upon interactive pairs of OSs, i.e., interaction between a host OS and one or more potentially different guest OSs. Both the host and guest OSs support a number of filesystem (FS) types. The host OS stores VM image files into the underlying filesystems, whereas the guest OSs employ one or more of these filesystems. As hypervisor-based virtualization imposes an OS pair, it also establishes interactive FS pairs. There are many available combinations of OSs and underlying FSs, but the overall FS performance may significantly vary among different FS pairs depending on the charac-

teristics of the workload (WL). Out of a relatively large number of factors that determine FS performance, we focus on the influence of the interactive FS pair. This paper evaluates the performance of different filesystem pairs in a virtualization environment, and the obtained results are aimed at being applied in the context of CC.

2. RELATED WORK, OBJECTIVE AND MOTIVATION

The paper reports on a FS performance analysis of FS pairs for type-1 hypervisors. We emphasize that FS performance is a fundamental factor for achieving an adequate level of QoS in a CC environment. In related work, FS performance in Virtual Environment (VE) has been analyzed in different ways. The most common approach includes the FS performance comparison of different hypervisors, such as KVM, VMWare, Xen, and Hyper-V. This approach relies on the use of filesystem benchmark applications such as HD Tune Pro, Bonnie++, Iozone, LMbench, LINPACK, etc. For details on certain performance comparisons, reader may cf. [7–12].

Some evaluation approaches include the analysis of I/O speed with respects to overall IO performances in a virtual environment for wide range of cloud applications [13–14].

In some papers, the experimental results relate to the impact of the estimated costs for the realization of cloud technologies [15]. Further work is dedicated to the question of virtual infrastructure management, showing cloud resources can be limited in order to respond to dynamic changes in a VE [16].

Finally, certain research efforts have been dedicated to performing comparative analysis of the modern hypervisors, which is in line with the approach applied in our paper [12], [17–23].

The main contribution of this study relates to comprehensive mathematical modeling of the FS performance in a VE employing type-1 hypervisors, with special focus on the interactive FS pairs. The FS pair modeling includes many factors that can be explored as if being independent or mutually correlated. The model proposed in this paper is applicable to most of the type-1 VE. The basic idea underlying our approach is to provide a specific mathematical model, apply it to a particular case study, and then interpret and validate the experimental results. We also contribute by proposing a Knowledge Data Base (KDB) comprising the collection of optimal FS pairs, available to VE administrators.

Compared to related work, we believe that our study introduces more comprehensive modeling of the FS performance in VE. At the practical level, while most of the related approaches consider just a single case study [17–23], we consider three case studies. Compared to related work, our main focus is FS pair modeling and KDB with the optimal FS pairs. Like most of the related studies, we show that there is no optimal FS pair that suits all possible use cases, but that the optimal FS pairs depend on WL and many other factors and change over time with the emergence of new FS versions and other VE factors.

Our paper presents the FS performance evaluation in fair-play conditions, i.e., it reports on the performances of the FS pairs formed from the selected FS types (Ext4, XFS, and Btrfs) applied with KVM as a representative of the type-1 hypervisors. The fair-play conditions assume the use of identical hardware for all the evaluated elements, the same charac-

teristics of the generated virtual machines (VM) and the identical version of the guest OS. We select KVM with full hardware virtualization, whereas FS types for host and guest FS are Ext4, XFS, and Btrfs. In addition, we use Filebench, which is a modern multi-threaded based benchmark, easily configurable, and adequate for the simulation of real-world applications. Four different test workloads are selected to simulate realistic workload conditions and applications: web server, e-mail server, fileserver, and random file access. The research protocol can be briefly described as follows. We introduce the mathematical model for FS pairs, select FS pairs for the evaluation, define the hypotheses related to the expected behavior of FS pairs, and finally proceed with the benchmark measurements. The results are interpreted in the context of the introduced mathematical model and hypotheses. We perform an in-depth analysis of guest and host OS FS behavior with different workload types and believe that the reported results are insightful for system administrators dealing with virtualization and some QoS issues in small-scale CC environments.

3. SELECTED TYPE-1 HYPERVISORS AND FILESYSTEM PAIRS

The type-1 hypervisor-based virtualization representatives are the following: ESXi with the original VMware FHV (Full Hardware Virtualization); Xen with two virtualization types FHV (QEMU based) and PV (paravirtualization) which is suitable for open-source PV guests; KVM/Proxmox with the FHV (QEMU based) virtualization; and Hyper-V with two kind of virtualizations, FHV (Microsoft original) and PV (paravirtualization) for Microsoft Windows OS.

In a hypervisor-based virtualization architecture, there are three fundamental components: the hypervisor as a kernel optimized for a particular VE, a host OS, and a guest OS. The host OS supplies drivers and supports management. It contains a host FS as a storage for host OS (hOS) and VM images. A VM contains a guest OS (gOS) and guest FSs which serve as storage for guest OS and guest applications. The hypervisor-based virtualization imposes interactive FS pairs (gFS-on-hFS). Each OS can support several modern FS types that undergo long-term development. Most of them are 64-bit, extent based, with accelerating techniques for allocation and searching (H-Tree/B-Tree). The overwriting or Copy-on-Write (CoW) techniques are adopted as write/update methods. The FS performance depends on the file caching, journaling, and different tunable parameters.

For a VE, both the host and guest OSs can be Linux-based or Microsoft Windows-based. Similarly, hypervisors with an accompanying host OS can be a Linux-based (e.g., ESXi, Xen and KVM/Proxmox) or Windows-based (e.g., Hyper-V). Linux OSs support a number of FS types, whereas the Microsoft Windows OS family implements only two FS types: NTFS and FAT (the latter of which is unsuitable for this purpose).

Thus, considering the number of available FS pairs (gFS on hFS) we can observe the following:

- Linux-based hypervisors and Linux VMs can include a very large number of FS pairs (gFS on hFS);
- Microsoft Windows-based hypervisors and Linux VMs can still include a significant number of FS pairs (gFS on NTFS);

• Microsoft Windows hypervisors and Windows VMs include only one FS pair (NTFS on NTFS).

Since the number of FS types is large, in the reported study we focused on the three popular 64-bit FSs: Ext4, XFS, and Btrfs. The main reason behind this decision is that all of them are modern, extent-based, and commonly used in Linux environments. Ext4 is robust and powerful in its performance, with relatively simple technologies (such as H-Tree, extent-Tree, pre-allocation, delayed allocation), and it is unlikely that any significant progress of its features will be made in the near future. XFS and Btrfs are two modern and promising FSs with data structures based on B+ Trees. The B+ Tree technology is constantly being developed and improved. Btrfs relies on the CoW method, which is novel when compared with the traditionally applied overwrite methods. This study is particularly focused on the case when virtualization is applied to both the host and guest Linux OSs. The performance of the chosen FSs in a VE pair is different from their performance in a real-life hardware environment. A very interesting comparison between the aforementioned FSs is presented in this paper as a case study encompassing a combination of 9 (3x3) FS pairs.

In the rest of this section, we briefly describe the three chosen Linux FSs.

Ext4 is a native Linux FS developed to resolve the capability and scalability issues of its predecessor (ext3 FS) caused by double and triple indirect block mapping characteristics. Ext4 manages storage in extents (a range of continuous physical blocks that improve large file performance and reduce fragmentation). It employs a tree-based index to represent files and directories in the form of H-Trees [24–25]. A write-ahead journal is applied to ensure the operation atomicity, and the checksumming is performed on the journal, but not on the user data. Although it has many advantages over its predecessor (such as extents, persistent pre-allocation, delayed allocation, and improved timestamps), the backward compatibility enforces some limitations (e.g., no support for snapshots).

XFS was originally developed as a native Silicon Graphics IRIX FS and ported to Linux in 2001. Nowadays, it is supported by most Linux distributions and some of them recommend it as the default FS for home or boot partitions. XFS is a high-performance 64-bit FS that allocates space in extents with data stored in B+ Trees [26]. The efficient allocation of free extents is achieved by dual indexing (one tree is indexed by the size, and the other by the starting block of the free extent), whereas the delayed allocation prevents FS fragmentation. Although snapshots are not supported and the underlying volume manager is expected to support that operation, the meta-data journaling and write barriers ensure data consistency. Extreme scalability of I/O threads and FS bandwidth originate from the parallel execution of I/O operations. The issue of addressing slow meta-data operations, which result in poor performance when write operations are performed on a large number of small files, has been partially resolved with a delayed logging feature.

B-Tree FS (Btrfs) is a native CoW Linux FS designed to offer more efficient storage management and better data integrity features. It aims at solving scalability problems for larger and faster storage, such as lack of pooling, snapshots, checksums, integral multi-device spanning, and built-in RAID support. The FS layout is based on a forest of CoW friendly B-Trees [27–31]. The main idea behind the CoW friendly B-Trees is to use standard B+ Tree construction, employ top-down update procedure, remove leaf-chaining, and use lazy reference-counting for space management. Based on the CoW technique, the FS may

be self-healing in some configurations. Disk blocks are managed in extents, with checksumming being performed for the purpose of integrity, and reference counting for the purpose of space reclamation. A vast variety of other useful features are implemented in Btrfs, including online defragmentation, online volume growth and shrinking, online block device addition and removal, online balancing, online data scrubbing, subvolumes, hierarchical per-subvolume quotas, and out-of-band data de-duplication. The B-Tree is a data structure that stores generic items organized by a key. Nodes contain only keys and pointers to the child node or leaf below, whereas leaves contain the actual variable sized data of the Tree. As they are tailored to systems reading and writing large blocks of data, B-Trees are suitable data structures for databases or FSs. Thus, FSs use B-Trees to search directories and extent descriptors, and for file allocation and file retrieval. Btrfs is particularly organized as a forest of B-Trees.

4. MATERIALS AND METHODS

In general, the proposed mathematical modeling of FS pair performance includes a large number of factors, but in this research, special attention is dedicated to interactive FS pairs. The modeling encompasses the following characteristics: the Workload (WL), VMs (with the accompanying gOS and gOS FS), the hypervisors, and the gOS and hOS FS. For the purpose of FS performance evaluation, the benchmark or real-life applications can be used, where all kinds of test procedures generate specific FS WLs. For each workload, we consider the parameter TW representing the total processing time. Each workload contains a mix of four cycle types: random reading (RR), random writing (RW), sequential reading (SR), and sequential writing (SW), whereas writing can be synchronous or asynchronous, so writing performance depends significantly on the FS caching.

In a given FS, each workload generates different kinds of operations related to directories, metadata, free lists, file blocks, journaling, and housekeeping (HK).

In hypervisor-based virtualization, the analysis of the workload processing time is quite complex. The FS performance in VEs depends on a number of factors originating from the type of virtualization applied (FHV, PV), hOS and gOS. Additionally, considering the context of the hypervisor VE environment, the overall data path becomes quite complex and relies on six components: application (benchmark), gOS kernel, guest OS FS, hypervisor as hOS kernel, VM image file, and host OS FS.

Figure 1 depicts an overview of the overall data path of a workload. The path is created by four objects (benchmark, gOS FS, VMI, and hOS FS), and two kernels (gOS kernel and hypervisor as hOS kernel).

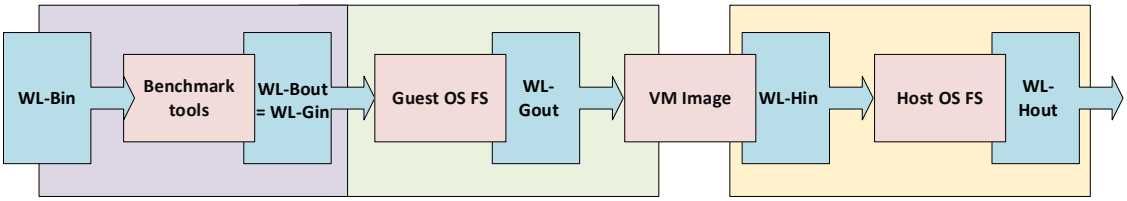

Figure 1. *Data path in virtualized environment.*

The benchmark has its own definition at the beginning of the procedure, and it provides a framework for the generation of the workload at the input (WL-Bin), taking into account the following parameters: the total number of workload files, the mean tree depth, the average file size, the file read/write block size, and the mean size of the appending block.

The input files are benchmarked with a range of different file operations such as opening, creation, deletion, reading, writing, and file appending. The sequence of the applied operations represents the benchmark output workload (WL-Bout). WL-Bout is applied to the gOS FS through the gOS kernel. It generates the guest output workload, WL-Gout, that consists of file block, inode, extent, free list, and directory read/write operations in the guest FS. The caching of gOS FS has a strong impact on WL-Gout. The WL-Gout component can be assumed to be a function of the benchmark request characteristics and the gOS FS processing procedures. This processing includes gOS FS features, gOS caching and virtual disk drivers.

The output workload from gOS FS is further redirected to the hypervisor, which maps it to a large VMI file. In other words, WL-Gout is mapped to a VMI file through the hypervisor, thus mapping all read/write operations onto VMI file operations. This mapping is marked as WL-Hin. WL-Hin is executed in the hostOS FS, resulting in a final sequence WL-Hout which comprises the following components: file block, inode, extent, freelist, and directory read/write operations in the host FS. Due to a large image file, the extent of read/write operations and the write method (overwrite or CoW) can have a great influence on the performance of WL-Hout. The final processing includes hOS FS features, hOS caching, and physical disk drivers.

The whole data path depends on various factors, including the characteristics of FS types on the guest and host sides, the file caching on the guest and host sides (i.e., a specific cooperation of these two caches), a large VMI file, the hypervisor interconnection of virtual and physical disk drivers, the hypervisor-CPU scheduling, and other.

For parameter T_w in a given VE, we consider six components (cf. equation 1):

$$T_w = f(App, g-kernel, g-FS, H-proc, Hyp-proc, h-FS) \quad (1)$$

1. Application (App) represents the interaction between WL-Bin as the benchmark inputs (definitions) and WL-Bout as the benchmark request for the guest OS FS. The selected application generates WL-Bout with random and sequential components.

2. g-kernel represents the processing time of the gOS kernel which takes the WL-Bout requests and forwards them to gOS FS.

3. Guest OS FS processing, g-FS, is a component targeting the gFS processing, which includes the gOS FS features, gOS FS caching, and virtual disk drivers. This component is very similar to the 6th component, h-FS. For both of these components (i.e., the 3rd and 6th components), the time for the OS-FS processing is represented by the function of the FS processing and FS cache processing (cf. equation 2):

$$g-FS = f(FSproc, FScache), \quad h-FS = f(FSproc, FScache) \quad (2)$$

4. Virtual hardware processing, *VH-proc*, represents the processing time of the virtual disk hardware. *VH-proc* strongly depends on the type of virtualization.

5. Hypervisor processing time, *Hyp-proc*, is the time necessary for the hypervisor to receive the requests from the VM (virtual disk driver) and forward them to the host VMI file. The FS requests from gFS (gOS-FS) are forwarded to hFS (hOS-FS) via the hypervisor and mapped through the VMI file, whereas many hypervisor parameters affect the FS performance.

6. Host OS FS processing, *h-FS*, is a component targeting the host FS processing, which includes the hOS FS features, hOS FS caching, and physical disk drivers. It works with a large VMI file and is a function of the FS processing and cache processing (cf. equation 2).

Some components of equation (1) are closely interrelated, especially the 3rd component and the 6th component. Virtual machines include a guest OS, which support several gOS FS types. Also, each hypervisor is related to its own hOS, which provides the virtual disk drivers and physical disk drivers. The hOS can support one or several hOS FS types.

In hypervisor-based VE, we must consider a FS pair, i.e., g-FS/h-FS, and consider the rather complex interaction between two FS caches. The interaction of FS pair is given in equation (3):

$$T_{FSpair} = f(gFSt/hFSt, gFSc/hFSc, HypFSparam) \tag{3}$$

The first component in equation (3), i.e., *gFSt/hFSt*, relates to FS types in the underlying FS pair. We recall that FS types may have different characteristics, as discussed in Section III.

The second component of equation (3), i.e., *gFSc/hFSc*, represents a pair of FS caches. These caches can be cooperative with Write Back (WB) or Write-through (WT) semantics or exclusive (none mode) when the hypervisor excludes the hOS FS cache for VMs.

The third component of equation (3), i.e., *HypFS param*, is related to the hypervisor tunable parameters. Each hypervisor has a number of tunable parameters, and some of them are significant for FS performance, such as CPU scheduling.

It is important to note that in this study we primarily focus on newer versions of FSs, gOSs and guest kernels, hypervisors with hOS, and CPU models with HW extensions. In a Linux-based VE, we consider three popular FSs, i.e., Ext4, XFS, and Btrfs, which allows for the generation of 9 FS pairs. We model the performance of these 64-bit Linux FS indexed by different techniques: B+ Trees, H-Trees, extent-Trees, linear lists, linked lists, etc.

For a B-Tree of order d and with n records, the cost of all operation processing operations grows at logarithmic rate, as $log_d(n)$. For all four types of actions (insertion, retrieval, updating, deleting), general equations for a B-Tree are:

$$T_{B+Tree}(mngmnt) \approx O(log_d(n)) \tag{4}$$

$$T_{B+Tree}(mngmnt) \approx f(indexes, keys, nodes, balancing, etc) \tag{5}$$

For all writing operations, Btrfs employs the CoW method (cf. eq. 6), whereas XFS and Ext4 employ the update (overwrite) method (cf. eq. 7).

$$Twriting=f(CoW) \quad (6)$$

$$T_{writing} = f(overwrite) \quad (7)$$

For directory operations, Btrfs and XFS employ B+ Trees, due to which T_{dir} exposes logarithmic cost as in eqs. (4) and (5), whereas Ext4 employs H-Trees (cf. eq. 8).

$$T_{Dir} = T_{Htree}(mngmnt) = f(hash\_function, hash\_colision) \quad (8)$$

For metadata operations, Btrfs and XFS employ B+ Trees, due to which T_{meta} exposes logarithmic cost as in eqs. (4) and (5), whereas Ext4 employs linear inode table (cf. eq. 9).

$$T_{Meta} = T_{linear-inode-table}(mngmnt) \quad (9)$$

For free list operations such as free inode, free block, and free extent lists, Btrfs and XFS employ B+ Trees, due to which TFL exposes logarithmic cost as in eqs. (4) and (5), whereas Ext4 employs linear bitmap (cf. eq. 10).

$$T_{E} = T_{linear-bitmap}(mngmnt) \quad (10)$$

For FileBlock accesses, Btrfs and XFS employ B+ Trees, due to which $T_{fileblock}$ exposes logarithmic cost as defined in eqs. (4) and (5), whereas Ext4 employs H-Trees in extent Tree structures (cf. eq. 11).

$$T_{FileBlock} = T_{Extent\_tree}(mngmnt) \quad (11)$$

For housekeeping, the most activities are performed by Btrfs (CRC for all operations), eq. (12), whereas XFS and Ext4 are much simpler (CRC for journaling), as given in eq. (13).

$$T_{HK} = f(data\_CRC, metadata\_CRC, Journaling\_CRC) \quad (12)$$

$$T_{HK} = f(Journaling\_CRC) \quad (13)$$

The performance costs of I/O operations are summarized in Table 1. We introduce labels *Cx.y* to denote performance costs of operation *x* performed on FS *y*. Every feature based on a B+ Tree has a logarithmic cost defined in eqs. (4) and (5). As Ext4 relies on the use of linear lists and H-Trees, Table 1 shows the cost values that are dependent on the hash operation and those that are subordinated by the linear search operation. Although B+ Tree has its general principles of generation and file manipulation, there is a cost difference between the FSs that rely on a B+ Tree as they employ this data structure in different ways. This includes differences in the organization of the indexes, keys, nodes, search, balancing, and other techniques. A particular difference is related to the use of the B-Tree in Btrfs and XFS environments, as Btrfs applies the CoW method to the underlying B-Tree, whereas XFS applies the traditional overwrite method. Btrfs FS employs the CoW update method,

whereas XFS and Ext4 employ the overwrite update method for writing operations. The CoW method is expected to have a significantly higher cost.

When working with directories, Btrfs and XFS employ B+ Trees and operate with logarithmic cost properties (eqs. 4 and 5), where *n* is the number of objects in the directory. Ext4 employs H-Trees, resulting in item searching being penalized by the hash performance, which depends on the hash function and the hash collision eq. (8).

The meta-data operations on Btrfs and XFS retain B+ Tree logarithmic cost properties dependent on the number of inodes in the filesystem, cf. eqs. (4) and (5). Ext4 manages a linear inode table and has linear cost properties, eq. (9).

Btrfs and XFS manage free lists (free inode list, free blocks list and free extents) with logarithmic cost properties, cf. eqs. (4) and (5), where *n* is number of objects in Free Lists (blocks, extents, inodes). Ext4 employs a linear bitmap for its free lists and has linear cost properties, cf. eq. (10).

Direct file block access consists of different time components: item retrieval, item reading, item appending, item writing, and item deletion. Each file is made up of data extents, thus the direct file block manipulation is practically extent-Trees manipulation. Direct file block access is performed with a logarithmic cost, both for Btrfs and XFS cf. eqs. (4) and (5), where *n* is the number of file extents. Ext4 employs H-Trees for extents and thus suffers from H-Tree penalties, cf. eq. (11).

The housekeeping operations on Btrfs depend on data, meta-data, and journaling CRC, cf. eq. (12), which means that they are very intensive, especially in the case of a large number of write operations. The housekeeping operations on XFS and Ext4 depend only on the journaling CRC, cf. eq. (13). Thus, the housekeeping costs are small for Ext4 and XFS, but can be significant for Btrfs.

Table 1. *Performance Costs.*

Operation / FS	Ext4	cost	XFS	Cost	Btrfs	cost
Update method (writing)	Overwrite	C1.1	Overwrite	C2.1	CoW	C3.1
Directory operations	H-Tree	C1.2	B+ Tree	C2.2	B+ Tree	C3.2
Meta-data operations	Linear inode table	C1.3	B+ Tree	C2.3	B+ Tree	C3.3
Free lists operations	Linear bitmap	C1.4	B+ Tree	C2.4	B+ Tree	C3.4
File block access	H-Tree	C1.5	B+ Tree	C2.5	B+ Tree	C3.5
House keeping	Journaling CRC	C1.6	Journaling CRC	C2.6	Data, meta-data, and journaling CRC	C3.6
Small file embedding	None	C1.7	Moderate performance	C2.7	Good performance	C3.7

5. THE HYPOTHESES

This research employs Linux both as the guest and host OS. Different features of Ext4, XFS and Btrfs as host and guest OS underlying FSs are analyzed, and the following assumptions about the expected I/O performance are adopted.

B+ Trees boost data retrieval, cf. eqs. (4) and (5). Each B-Tree based FS has its own B-Tree organization, which has a direct impact on performance. This hypothesis applies to all B+ Tree-based FS reading operations, including directory, meta-data, and free-list operations as well as direct file-block access (Table 1, costs C2.2, C2.3, C2.4, C2.5, C3.2, C3.3, C3.4, and C3.5). Small file embedding is expected to have a major positive impact on random performance (Table 1, costs C1.7, C2.7, and C3.7).

CoW has a negative impact on write performances due to changed pages and CoW-ed extents (cached and written elsewhere) (cost C3.1) when compared to overwrite update method (costs C1.1 and C1.2). Garbage collection is also required for CoW.

CoW turns small, random updates into sequential cycles, thus providing the workload with more sequentially. The negative impact of CoW on sequential writing is expected to be more significant when compared to random writing operations (cost C3.1).

Housekeeping is expected to provide the largest negative impact on the Btrfs performance when compared to the Ext4 and XFS performance (costs C3.6 related to cost C1.6 and C2.6).

For the interpretation of the FS performance, we consider interactive FS pairs, and we think that each FS type in FS pair cannot be analyzed separately, but only as an interactive FS-pair. The gOS FS has a specific behavior in the FS-pair. It works similarly to physical conditions by generating a sequence of requests (files/directory operations) for the gOS FS, which operates based on the gOS FS features, gOS caching, and virtual disk drivers. For FHV, virtual disk drivers are identical to physical disk drivers, whereas virtual disk is represented as a large VMI file. Each gOS FS generates a specific WL-Gout which is mapped to a large VMI and then the WL-Hin is generated. For each gOS FS observed in the same benchmark, a quite different WL-Hin is generated. Each hOS FS works specifically, it gets a WL-Hin that does not look like any application, and it is the output of the whole FS with a benchmark as input. The hOS FS works in real physical conditions – processing a sequence request for one large file, VMI. The gOS FS operates based on the hOS FS features, hOS caching, physical disk drivers, and physical disks. In short, gOS FS generates WL-Gout, a complex sequence of requests for hOS FS, which hOS FS processes through a large VMI file. Complex interaction of two FS is the reason why both FSs must be viewed integrally as a pair.

Clear indications of the best/worst host and guest OS FS pairs are expected, where one or more of those pairs will provide the highest/weakest I/O performance.

The aforementioned assumptions are experimentally validated with a set of performance measurements (synthetic benchmarking), and the interpretation of the results is presented in the next section of the paper.

6. EXPERIMENTAL EVALUATION

We consider three different case studies of the KVM hypervisor-based VE. In each case study, we apply the same protocol but different hardware. We employ two different CPUs, two magnetic hard drives, and two kernel versions of the same Linux distribution. For the first case study (CS1), experiments were performed on the Intel Xeon E3110 @ 3.00GHz, 8GB DDR3-RAM, with Seagate Barracuda 500GB SATA-3 hard disk (7200 rpm, 6 GB/s). Centos 7.2 with Linux Kernel 3.10.0-327.36.3.el7.x86_64 is chosen as the native host for KVM hypervisors and the guest operating systems. For the second case study (CS2), experiments were performed on the Intel Xeon E3110 @ 3.00GHz, 8GB DDR3-RAM, with Toshiba DT01ACA050 500GB SATA-3 hard disk (7200 rpm, 6Gb/s). Centos 7.9 with Linux Kernel 3.10.0-1160.21.1.el7.x86_64 is chosen as the native host for KVM hypervisors and the guest OSs. For the third case study (CS3), experiments were performed on the dual core AMD Ryzen 5 3400G @ 3.7GHz, 8GB DDR4-RAM, with Toshiba DT01ACA050 500GB SATA-3 hard disk (7200 rpm, 6Gb/s). Centos 7.9 with Linux Kernel 3.10.0-1160.21.1.el7.x86_64 is chosen as the native host for KVM hypervisors and the guest OSs.

Because of the relatively small amount of available RAM (8GB), each of the three VMs was assigned 2GB of RAM, thus allowing the host OS to operate with the remaining RAM. The experiment was performed on one, two, and three VMs simultaneously to examine the impact of the host OS caching on the KVM virtualization, proceeding with the experiments for the writeback (WB) cache mode.

The hypervisor's random and sequential performances are tested in the Filebench benchmark environment, setting four different application workloads.

For practical reasons, the obtained experimental results are just partially presented in this paper, i.e., we discuss the third case study (CS3), in which we consider the performance for four WL, for native performance, and 1VM, (cf. Figs. 2–5).

7. DISCUSSION

Testing of each server workload is briefly discussed below. Measurements are performed for a Web-server, mail-server, and file-server workloads as well as for a random file access. One Virtual Machine is used for each test. Different pairs of the aforementioned filesystems were considered during the test.

For each evaluated workload, starting with the WL-Bin, we have measured the characteristics of the WL-Hout (Figure 1), taking into consideration the processing time and the overall throughput. For each workload, we use the following main criteria: the best and worst FS pair results (gOS FS on hOS FS) considering all pairs, and presence of FSs in the best and worst combination on the guest and host side. For FS names, we will use abbreviation, Btrfs as B, Ext4 as E, XFS as X.

In the Web-WL, there are 100 threads, where each thread selects 10 of the 1000 files in directory-Tree, makes the 10 sequences of open-read-close operations and 11th sequence as log append operation. The web-workload is characterized by the dominant data and metadata random reads. There are small components of the data and metadata random

writes in log file, which are synchronous. The dominant random reads indicate a small influence of the WB cache mode on the guest/host side. Results for native and 1VM performance for web WL are depicted in Fig. 2.

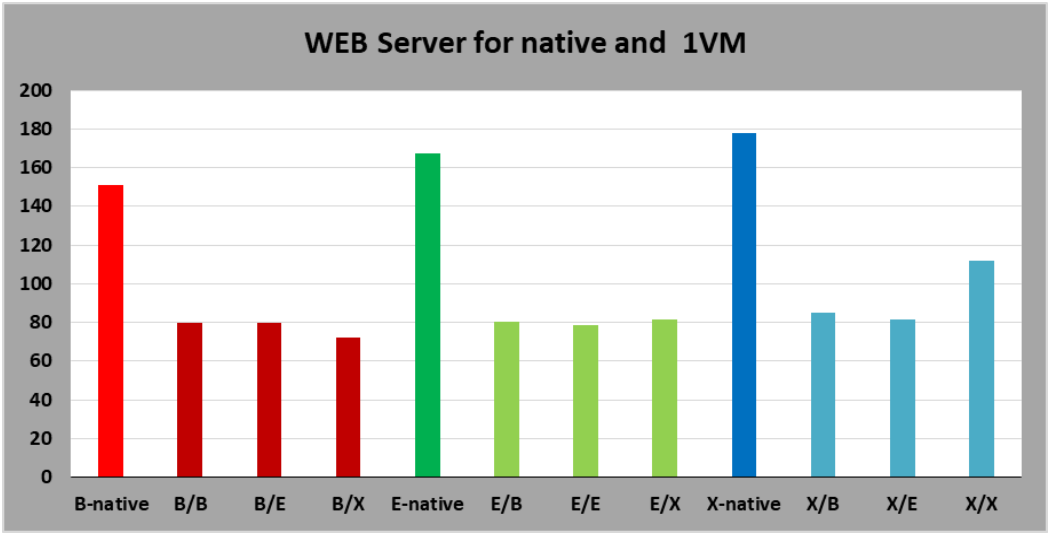

Figure 2. *WebServer throughput [MB/s] for native and 1VM environment.*

For all three case studies, we detected the best FS pairs for Web-WL: X/X, E/E, E/B, B/E, E/X, X/E, among which X/X, E/E, E/B stand out. In the best FS pairs, XFS and Ext4 dominate as gFS and hFS.

Particularly bad FS pairs for Web-WL are: B/X, B/B, X/B. In the worst FS pairs, Btrfs dominates as gFS and hFS.

When considering the best/worst FS pairs, for Web-WL X/X or Ext4 should be selected with all hFS, and B/X and B/B should be avoided. XFS and Ext4 are good choices for making FS pairs, whereas Btrfs should be avoided, especially on the guest side.

For RR components, on both sides (guest/host), B-Trees have the best performance for data retrieval. B+Trees of XFS used for directories [C2.2], inodes [C2.3], extents [C.2.5], FreeList [C2.4] and FileBlock [C2.5] are crucial for good random read performances. Similar insights regarding random read performances hold for Btrfs [C3.2, C3.3, C3.4, and C3.5]. However, there is a small random write component, hence the write methods still have an important role. Because of the overwrite method applied instead of CoW, XFS outperforms Btrfs [C2.1 and C3.1] on the guest/host sides.

We argue that the Btrfs CoW penalty is the main reason underlying the bad FS pairs such as B/X, X/B, B/B. Ext4 with its relatively simple but fast technologies (H-Tree, extent-Tree C1.1, C1.5), exhibits good characteristics in the best combinations (E/E and E/B) and behaves quite well on both the guest/host sides.

In the Mail-WL, there are 16 threads, each of which selects four of the 1000 files in a single directory. With these files, thread makes the following sequences of operations: delete, create-append-fsync-close, open-read-append-fsync-close, open-read-close. In the case of the varmail-workload, synchronous random writes are dominant for both data and metadata. A large amount of the random reads for both data files and metadata are noticeable. Random writes are synchronous, so the data writing must reach the disk drive. Due

to the dominant RR/RW-synchronous cycles, there is a small impact of WB caching on the guest/host sides. Housekeeping is also important to consider as it generates many writes. The results for native and 1VM performance for mail WL are depicted in Fig. 3.

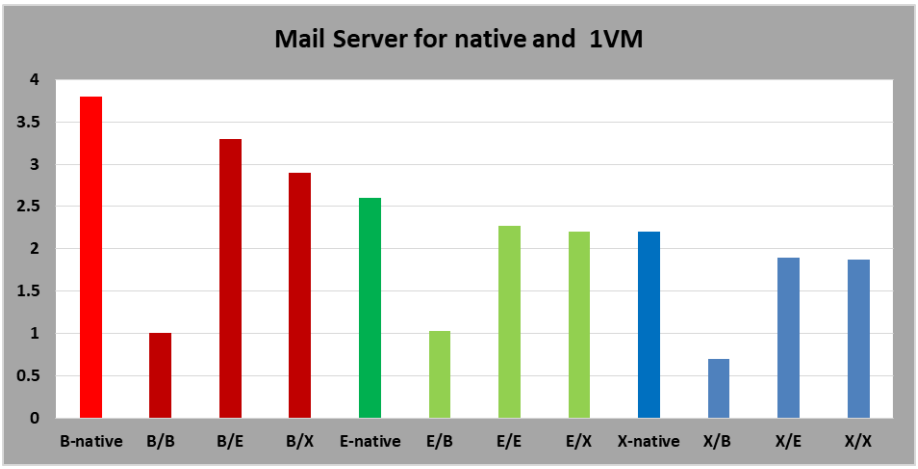

Figure 3. *MailServer throughput [MB/s] for native and 1VM environment.*

For all three case studies, we detected the best FS pairs for Mail-WL: X/E, E/X, E/E, B/E, B/X, among which B/E, E/X, E/E stand out. In the best FS pairs, Btrfs and Ext4 dominate as gFS, while Ext4 and then XFS dominate as hFS.

In addition, we detected the following bad FS pairs for Mail-WL: X/B, E/B, B/B, X/X, among which X/B stands out. In the worst FS pairs, XFS dominates as gFS, while Btrfs dominates as hFS.

When considering the best/worst FS pairs, for Mail-WL, B/E, E/X, E/E should be selected, and B/X, B/B should be avoided. Btrfs and Ext4 are good choices for gFS, and XFS should be avoided on the guest side. Btrfs should be avoided on the host side. Btrfs performs very well as gFS, and very bad as hFS, XFS is relatively bad as gFS, and Ext4 performs well in pairs on both sides.

For the RR component, the best are the B-Trees of XFS [C2.2, C2.3, C2.4, and C2.5] and Btrfs [C3.2, C3.3, C3.4, and C3.5]. However, for RW-synchronous cycles, XFS already has the well-known problem of low performance for random writes, whereas the Btrfs implements the CoW method. That is why Ext4 is the best selection for mail-WL. On the guest side, Btrfs and Ext4 perform quite well, while XFS generates an unfavorable WL-Gout sequence, which exhibits the worst behavior on the host side. In addition, on the host side, synchronous RW transfers pass through a large VMI file, so Ext4 performs as well as XFS, whereas Btrfs shows very bad performance.

Due to RW-synchronous accesses through a large image file, there is a growing influence of the extent_read [C1.5, C2.5, and C3.5] and extent_write operations [C1.5, C2.5, and C3.5] for the image file manipulation. XFS/Btrfs is at an advantage due to the B-Trees, and especially because of the use of the B-Trees for extents [C2.5 and C3.5], which are more efficient than the Ext4 extent-Trees [C1.5]. In addition, due to the dominant RW-synchronous, the influence of the writing method costs [C1.1, C2.1, and C3.1] is crucial, while due to the dominant RW, there is a growing negative impact of CoW and enhanced HK. The-

refore, the dominant random write component makes Btrfs the worst hostOS file system [C3.1].

In FS-WL, there are 50 threads, each of which selects 5 of the 10,000 files in the directory-Tree, making the following sequences of operations: create-write-close, open-write-close, open-read-close, delete, and stat. In the case of fileserver-workload, all the throughput components are equally presented (random reads, sequential reads, random writes, sequential writes, create/delete/metadata operations). The impact of the WB cache mode is significant, as the sequential reads and writes are intensive. Writes are not synchronous, thus the impact of the cache can be exceptionally significant. The results for native and 1VM performance for fileserver WL are depicted in Fig. 4.

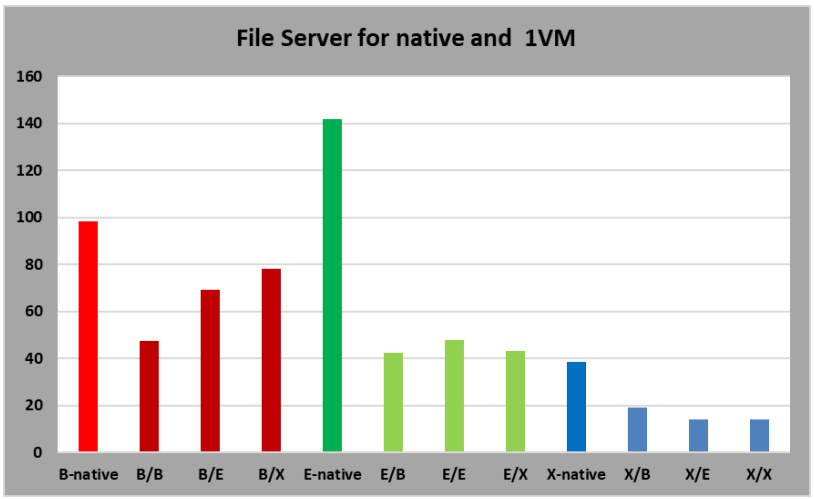

Figure 4. *FileServer throughput [MB/s] for native and 1VM environment.*

For all three case studies, we detected that the best FS pairs for FS-WL are B/X, B/E, E/X, B/B, E/B, among which B/X and B/E stand out. In the best FS pairs, Btrfs and then Ext4 dominate as gFS, while XFS slightly dominated as hFS, but all three FSs perform well. For all three case studies, we detected the following bad FS pairs for FS-WL: X/X, X/E, X/B. In the worst FS pairs, XFS dominates as gFS, while there is no dominance for hFS.

Considering the best/worst FS pairs for FS-WL, the best selection is either B/X or B/E, while it is recommendable to avoid XFS as a guest FS.

Each gFS (with its features) generates a unique WL-Gout, which passes through a large VMI file. For complex FS-WL the WL-Gout becomes also very complex. In FS-WL, WL-Gout with XFS performs the worst when processed through a large VMI file in hFS, whereas WL-Gout with Btrfs fits very well on all hosts hFS.

For sequences in which all the components are equally presented (random/sequential, reads/writes, file-data/metadata), the file block access components (Btrfs B-Trees [C3.5], XFS B-Trees [C2.5] and Ext4 extent-Trees [C1.5]) can have a significant impact on the performances. Also, due to a large number of files in a workload Tree, directory operations play a more important role (Btrfs B-Trees [C3.3], XFS B-Trees [C3.2] and Ext4 H-Trees [C3.1]). Due to the sequential SW components, the negative impact of CoW on Btrfs is reduced.

The experiment shows that WL-Gout sequences for Btrfs are optimal for all hFS, whereas XFS generates unfavorable sequences for all evaluated hFS. For Btrfs as gFS, when such optimal WL-Gout sequences are performed on a large VMI, all three hFS perform well. It is only necessary to ensure an optimal FS pairing on the guest/host side.

In the RFA-WL, five threads are created, where each thread selects five of the 10,000 files in the directory-Tree, making the following sequence of operations: open, open, open, append-close, read-read, close, close. In the case of the RFA-workload, random reads and writes are dominant for both data and metadata. Random writes are non-synchronous, so the impact of WB cache mode is significant. The housekeeping operations abound with writes, which is important for consideration. The results for native and 1VM performance for RFA WL are depicted in Fig. 5.

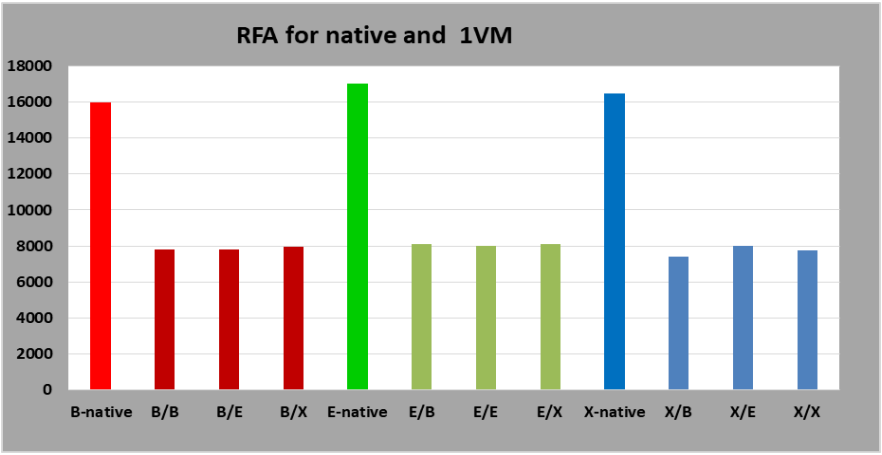

Figure 5. *RFA throughput [MB/s] for native and 1VM environment.*

For all three case studies, we have detected that the best FS pairs for RFA-WL are X/X, E/X, E/E, E/B, among which X/X, E/X stand out. In the set of the best FS pairs, Ext4 dominates as gFS, while XFS and Ext4 dominate for hFS. In addition, we detected inefficient FS pairs for the case of the RFA-WL: B/X, X/B, among which the pair B/X stands out. In the worst FS pairs, Btrfs dominates as gFS, while there is no dominance for hFS.

Considering the best/worst FS pairs for RFA-WL, the best selection is either X/X or E/X, whereas B/X should be avoided. Ext4 is very good as gFS, whereas FS pairs formed by both Btrfs and XFS should be avoided (B/X and X/B).

RFA-WL is dominated by RR and RW-asynchronous components. Due to RW-asynchronous components, WB caching has a large positive impact on the guest/host sides. B+Trees of XFS [C2.2, C2.3, C2.4, and C2.5] and B+Trees of Btrfs [C3.2, C3.3, C3.4, and C3.5] are the most promising candidates for RR components. Ext4 is very good as gFS, and it generates the most favorable WL-Gout sequences, which perform best for all hFS. XFS generates WL-Gout sequences that can well perform with XFS and Ext4 as hOS, whereas Btrfs makes the least favorable WL-Gout. Although the WB cache absorbs most RW accesses, some RW still passes to disk drivers, and this will have the most negative effect on Btrfs. In general, for RFA-WL, the rather unfavorable WL-Gout (Btrfs) sequences perform worst on XFS hFS. Due to the WL-Gout sequences of XFS, as well as CoW and HK of Btrfs as hFS, the combination of XFS on Btrfs is also unfavorable.

As the main results, we present the best and the worst FS pairs for individual workloads. The obtained results are given in Table 2, where the fundamental results are shown in columns 4 and 5. The column 2 contains the information on the best/worst guest FS detected in the corresponding best/worst FS pairs. The column 3 contains the information on the best/worst hFS detected in the corresponding best/worst FS pairs.

Table 2. *Performance Results Summarized.*

Workload	gOS FS best/ worst	hOS FS best/ worst	best pairs	worst pairs
Web	[E, X]/B	[E, X]/B	X/X, E/E, E/B,B/E, E/X, X/E	B/X, B/B, X/B
Mail	[B, E]/X	[E, X]/B	X/E, E/X, E/E, B/E,	X/B, E/B , B/B, X/X, B/X
File Server	B/X	X/no	B/X, B/E, E/X, B/B, E/B	X/X, X/E, X/B
RFA	E/B	[X, E]/no	X/X, E/X, E/E, E/B	B/X, X/B, X/E

The analysis of the results presented in Table 2, without taking into account the workloads characteristics, can be summarized as follows: there is neither the best nor the worst option for the guest or host. Their performance depends on workload characteristics (i.e., each FS can perform either as the best or as the worst depending on the workload), and they are very sensitive to the guest/host FS pairs.

Based on the reported three case studies and four WLs for Centos gOS, we provide the following recommendations. For Web WL, we recommend the Ext4/XFS as gFS and Ext4/XFS as hFS, while Btrfs as gFS and Btrfs as hFS should be avoided. For Mail WL, we recommend Btrfs/Ext4 as gFS and Ext4 as hFS, while XFS as gFS and Btrfs as hFS should be avoided. For FileServer WL, we recommend Btrfs as gFS, while XFS as gFS should be avoided. For RFA WL, we recommend Ext4 as gFS, while Btrfs as gFS should be avoided.

For individual FS, we recommend Btrfs for Mail as gFS and for FileServer as gFS, while Btrfs should be avoided for Web as gFS/hFS, for Mail as hFS, and for RFA as gFS. In addition, we recommend XFS for Web as gFS and for Web as hFS, while XFS should be avoided for Mail as gFS and for Fileserver as gFS. Finally, we recommend Ext4 for Web as gFS, for Web as hFS, for Mail as gFS, for Mail as hFS and for RFA as gFS. There is no recommendation for avoiding Ext4.

Regarding solid-state drives, their examination will follow this research in the field on hypervisors case studies. Although the authors conducted some research on this topic, the results are not yet mathematically explainable. Certain studies regarding Microsoft Windows environment are available in [32]. More on this topic will be discussed in the research that follows.

Regarding real environments, not virtual ones, a reader may conclude that virtualization takes a toll on performance, i.e., dropping I/O throughput. However, typical small-to-medium-sized service providers would apply virtual environments, such as providing a machine with sufficient processing power and storage devices to reduce electricity consumption, providing rack spaces for servers, etc.

8. CONCLUSION

Virtualization techniques have a significant impact on operating system performance, and in this paper, we examine this impact on specific components, such as different FS pairs. When considering a hypervisor virtualization environment based on different filesystems on the host and guest sides, a number of FS pairs could be included in the analysis. We analyzed the behavior of three 64-bit Linux FS, both on the guest and on the host side, resulting in nine pairs for examination. We introduced the model for FS pairs and validated it for the KVM hypervisor, but our model is also applicable for most Linux-based hypervisors, such as Xen, ESXi, and Proxmox.

Our KVM-based case studies showed that FS pairing is quite sensitive and that there is no optimal FS pair. The choice of a suboptimal FS pair depends on WL, due to a complex VE with a large number of input variables, complex structures in FS itself, and the complex FS pair interaction. One of the important conclusions is that the FS performance in a VE can be interpreted by analyzing the FS pair as a whole instead of an individual FS analysis.

The reported results are quite consistent with the best practices obtained in experiments related to environments with and without virtualization. The results also indicate that Btrfs is not a good solution if the workload is generating big amounts of random writes. Ext4 is still good enough when compared to XFS and Btrfs, although it uses relatively simple technologies and does not have a sophisticated B+ Tree structure. Ext4 shows excellent performance.

Our model and KVM results show that optimal pairing of FS types on the host and guest sides may be rather challenging. Thus, an additional contribution of this paper is that it provides insight into the performance of selected FS pairs for typical application WL, based on three case studies in a KVM hypervisor-based VE.

We think that the optimal pairing of FS types on the host and guest sides is particularly complex. Administrators of VE face a complex problem in determining the optimal VE for their applications of interest. First, for each type-1 hypervisor (except Hyper-V) that will be included in VE, a pool of hOS FS with different types should be created, on which they can place VMs and migrate if necessary. Secondly, administrators should determine the optimal FS pair for applications of interest. Our model can be used for hypotheses about expected behavior, whereas good benchmark or real-app testing can provide real validation. For VMs with gOS, possible application-based adjustment of gOS FS types should also be provided.

Beside the model, another contribution of this paper is the beginning of Knowledge Data Base (KDB) creation that is related to FS performance of FS pairs in hypervisor-based VE. We consider the KDB to be another significant theoretical contribution of this paper. Based on three case studies in KVM VE, we get the knowledge about optimal and very bad FS pairs for typical application WL. For now, KDB includes three case studies for KVM VE, with 9 FS pairs, Centos 7 as gOS, Filebench as WL generator. KDB is open for extension with new case studies, which include another FS pairs (with NTFS, F2FS, JFS), other hypervisors (ESXi, Xen, Proxmox, Hyper-V), other hOS and gOS, other benchmarks or real applications, and new versions of all of these components. We consider that our KDB can serve administrators to create VE for specific system case.

Future work on optimal FS pairing in VE may include the introduction of new FS types in the analysis, such as NTFS, F2FS, and JFS on the guest/host side, and evaluation of the case studies based on other hypervisors, such as ESXi, Xen, Proxmox, and Hyper-V. As a part of the future work, we may consider the application of artificial intelligence as the possibility for intelligent management of FS pairs, assuming that large hOS storage pools based on different FS (Btrfs, Ext4, XFS, etc.) exist. The proposed system would be trained with information about the best/worst FS pairs supplied from KDB. Based on the detected application WLs on VMs and training data, the system should migrate guest VMs to another hOS FS pool, and realize optimal FS pairs.

FUNDING:

This research received no external funding.

INSTITUTIONAL REVIEW BOARD STATEMENT:

Not applicable.

INFORMED CONSENT STATEMENT:

Not applicable.

CONFLICTS OF INTEREST:

The authors declare no conflict of interest.

REFERENCES

[1] R. Y. Ameen, A. Y. Hamo, "Survey of server virtualization," *International Journal of Computer Science and Information Security (IJCSIS)*, Vol. 11, No. 3, 2013. arXiv preprint arXiv:1304.3557

[2] E. Correia, "Hypervisor-based server virtualization," In *Encyclopedia of Information, Science and Technology*, 3rd Edition, IGI Global, pp. 1182–1187, 2015. DOI 10.4018/978-1-4666-5888-2.ch112

[3]A. Varasteh, M. Goudarz, "Server consolidation techniques in virtualized data centers," *IEEE System Journal*, Vol. 2, No. 11, pp. 772–783, 2017. DOI 10.1109/JSYST.2015.2458273

[4] T. Imada, M. Sato, and R. Kimura, "Power and QoS Performance Characteristics of Virtualized Servers," In Proceeding of the 10th IEEE/ACM International Conference on Grid Computing (GRID), 2009, pp. 232–240. DOI 10.1109/GRID.2009.5353054.

[5] J. Liu, Y. Zhang, Y. Zhou, D. Zhang, and H. Liu, "Aggressive Resource Provisioning for Ensuring QoS in Virtualized Environments," *IEEE Transactions on Cloud Computing*, Vol. 3, No. 2, 2015, pp. 119–131.

[6] J. W. Lin, C. H. Chen, and C. Y. Lin, "Integrating QoS awareness with virtualization in cloud computing systems for delay-sensitive applications," *Future Generation Computer Systems*, Vol. 37, pp. 478–487, 2014. DOI https://doi.org/10.1016/j.future.2013.12.034.

[7] C. D. Graziano, "A performance analysis of Xen and KVM hypervisors for hosting the Xen worlds project," M. A. thesis, Iowa State University, Ames, IA, 2011. DOI 10.31274/etd-180810-2322

[8] S. Pawar and S. Singh, "Performance comparison of VMWare and Xen hypervisor on guest OS," *IJICSE*, Vol. 2, No. 3, pp. 56–60, 2015. ISSN 2393-8528. https://ijicse.in/index.php/ijicse/article/view/43/41

[9] A. Kumar and S. Shiwani, "Guest operating system based performance comparison of VMWare & Xen hypervisor," *International Journal of Science, Engineering and Technology*, Vol. 2, No. 5, pp. 286–297, 2014. ISSN 2348-4098. Available: http://ijset.in/wp-content/uploads/2014/06/IJSET.0620140075.1011.1906_Ankit_Kumar_286-2971.pdf

[10] A. Bhatia and G. Bhattal, "A comparative study of various hypervisors performance," *International Journal of Scientific and Engineering Research*, Vol. 7, No. 12, pp. 65–71, 2016.

[11] V. P. Singh, "Analysis of system performance using VMWare ESXi server virtual machines," M. Sc. thesis, Thapar University, Patiala, India, 2012. Available: http://hdl.handle.net/10266/1809

[12] H. Kazan, L. Perneel, and M. Timmermann, "Benchmarking the performance of Microsoft Hyper-V server, VMWare ESXi and Xen hypervisors," *Journal of Emerging Trends in Computing and Information Sciences*, Vol. 4, No. 12, pp. 922–933, 2013. ISSN 2079-8407

[13] M. Polenov, V. Guzik, and V. Lukyanov, "Hypervisors comparison and their performance," In Computer Science On-line Conference, 2018, pp. 148–157. DOI 10.1007/978-3-319-91186-1_16

[14] P. Kedia, R. Nagpal, "Performance evaluation of virtual environment with respect to physical environment," *International Journal of Computer Applications (0975 – 8887),* Vol. 89, No. 11, pp. 17–22, 2014. DOI 10.5120/15676-4425

[15] P. Vijaya V. Reddy, L. Rajamani, "Evaluation of different hypervisors performance in the private cloud with SIGAR framework," *International Journal of Advanced Computer Science and Applications (IJACSA)*, Vol. 5, No. 2, pp. 60–66, 2014. DOI 10.14569/IJACSA.2014.050210

[16] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. Lightweight Virtualization: A Performance Comparison," In 2015 IEEE International Conference on Cloud Engineering, 2015, pp. 386–393.

[17] J. Hwang, S. Zeng, F. Wu, and T. Wood, "A component-based performance comparison of four hypervisors," In 13th IFIP/IEEE Int. Symposium on Integrated Network Management (IM) Technical Session, 2013, pp. 269–276. ISBN 978-3-901882-50-0 978-1-4673-5229-1, 978-3-901882-51-7

[18] A. Elsayed and N. Abdelbaki, "Performance evaluation and comparison of the top market virtualization hypervisors," IEEE International Conference on Computer Engineering and Systems, 2013, pp. 45–50. DOI 10.1109/ICCES.2013.6707169

[19] W. Graniszewski and A. Arciszewski, "Performance analysis of selected hypervisors (Virtual Machine Monitors-VMMs)," *International Journal of Electronics and Telecommunications*, Vol. 62, No. 3, pp. 231–236, 2016. DOI 10.1515/eletel-2016-0031

[20] S. A. Algarni, M. R. Ikbal, R. Alroobaea, A. S. Ghiduk, and F. Nadeem, "Performance evaluation of Xen, KVM, and Proxmox hypervisors", *International Journal of Open Source Software and Processes*, Vol. 9, No. 2, pp. 39–54, 2018. DOI 10.4018/IJOSSP.2018040103

[21] B. Djordjevic, N. Macek, and V. Timcenko, "Performance issues in cloud computing: KVM hypervisor's cache modes evaluation," *Acta Polytechnica Hungarica*, Vol. 12, No. 4, pp. 147–165, 2015. DOI 10.12700/APH.12.4.2015.4.9

[22] V. K. Manik and D. Arora, "Performance comparison of commercial VMM: ESXi, XEN, HYPER-V & KVM," In 3rd International Conference on Computing for Sustainable Global Development, 2016. ISBN Electronic ISBN 978-9-3805-4421-2, DVD ISBN 978-9-3805-4420-5, Print on Demand (PoD) ISBN 978-1-4673-9417-8

[23] D. Vujičić, D. Marković, B. Đorđević, and S. Randić, "Benchmarking Performance of Ext4, XFS, and Btrfs as Guest File Systems under Linux Environment," In Proceedings of 3rd International Conference on Electrical, Electronic and Computing Engineering IcETRAN 2016, Zlatibor, Serbia, June 13–16, 2016, pp. RTI1.3.1-5.

[24] K. V. Kumar, A. M. Cao, J. R. Santos, and A. Dilger, "Ext4 block and inode allocator improvements," In Proceedings of the Linux Symposium, Vol. 1, 2008, pp. 263–273.

[25] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Thomas, and L. Vivier, "The new Ext4 filesystem: current status and future plans," In Proceedings of the Linux Symposium, Vol. 2, 2007, pp. 21–33.

[26] M. Holton and R. Das, "XFS: a next generation journalled 64-Bit filesystem with guaranteed rate I/O," SGI Corp, Internet White Paper, 1995.

[27] O. Rodeh, "B-Trees, shadowing, and clones," *ACM Transactions on Storage (TOS)*, Vol. 3, No. 4, Article No. 2, pp. 1–27, 2008.

[28] O. Rodeh, "Deferred Reference Counters for Copy-On-Write B-Trees," IBM Corporation, Technical Report rj10464, 2010.

[29] O. Rodeh, J. Bacik, and C. Mason, "BTRFS: The Linux B-Tree filesystem," *ACM Transactions on Storage (TOS),* Vol. 9, No. 3, Article No. 9, pp. 1–32, 2013.

[30] H. Powell, "ZFS and Btrfs: a quick introduction to modern filesystems," *Linux Journal*, Vol. 2012, No. 218, Article No.: 5, 2012.

[31] Silicon Graphics Inc. XFS Filesystem Structure, Documentation of the XFS filesystem on-disk structures, 2006.

[32] B. Đorđević, V. Timčenko, and N. Maček, "NTFS fajl sistem u MS Windows i Linux okruženju," *Zbornik radova XIII međunarodnog naučno-stručnog Simpozijuma INFOTEH 2014*, 2014, pp. 805–808. ISBN 978-99955-763-3-2