MLFS:A Multiple Layers Share File System for Cloud Computing

Xiao Zhang #1, Wan Guo #, Zhanhuai Li #, Xiaonan Zhao #, Xiao Qin *2

#School of Computer Science Northwestern Polytechnical University 127 Youyi xi Road, Xi'an Shaanxi China
1zhangxiao@nwpu.edu.cn

*Department of Computer Science and Software Engineering Auburn University, AL USA 36849-5347
2xqin@auburn.edu

***Abstract*—Cloud computing systems offer computing and storage resources at a low price. The users only pay for it when the VM(short for Virtual Machine) is running. While after VMs shut down, the vendors also need disk space to store multi-gigabyte VM images. Past studies use deduplication to reduce the space requirements, but it needs to calculate hash values of each block and store them. It also causes performance overhead costs to find identical content. In this paper, We propose a multiple layers file sharing system(MLFS) designed for cloud computing. It stores OS data, Application data, and user data in different layers. Multiple users share OS data layers. Application data are constructed according to users' selection, and they are deleted after VM shut down. We compare our solution with ordinary storage systems, storage systems with deduplication, the results show that MLFS can save more space than others, while achieving reasonable performance in file system I/O operating.**

I. Introduction

Cloud can be formally defined as a parallel and distributed system of interconnected virtualized nodes, with the capability to dynamically provision new resources based on the service level agreement between the client and the service provider[1]. Virtualization is the key technology for Cloud Computing. Most of the modern cloud providers apply virtualization in their data centers, running hypervisors such as Xen[2], KVM[3], VMware, and Orcale hypervision.

The pay-as-you-go model of cloud computing saves money for users. The key point is many users share computing resources together, which is so called virtualization. Users get computing resources(CPU, memory, and network) when they boot up their VMs. After they shutdown their VMs, the computing resources will be released and allocated to other users. While CPU and memory are only needed during the runtime, storage resources cannot be released even user shutdown their VMs and leave the system. From the perspective of service providers, it's critical to promote the utilization of persistent storage for cloud computing users. There are many same data between different VM images.

Deduplication technology tries to find the identical content and stores it once to save space requirement.[4][5]. But deduplication needs extra storage space to store index information; it needs computing resources, extra read/write operating. It also decreases the data dependability and causes security risks. Qcow and other "copy on write" technologies share the image between different VM disk images[6]; the base OS image is overlayed with changes. But they cannot change OS image and do not handle the same blocks between different user data.

In this paper, we present a new approach to solve this problem. We propose a multiple-layers file system to store OS, Application, and user data in different layers. The top layer stores all user data and changed data; OS data and application data are stored on lower layers. Compared with traditional deduplication methods, our approach have the following benefits:

1) After deploying OS image on different sites, migrating VM between data centers only needs move user data instead of the whole image. It also accelerates the VM recovery speed.
2) It's easy to install patches for all users in a cloud computing environment. Once patches are installed in OS image, all users will automatically use the new OS images.
3) We distinguish changed data during write operation inside VM, avoiding massive computing to check identical blocks between different images like deduplication.
4) Deduplication will cause read/write performance overhead especially when there are many users accessing their data at the same time. Our solution can avoid this by creating multiple OS image copies. And all of these copies can be reused after user shutdown their VM.
5) Deduplication has security risks when comparing data that belong to different users. While in our solution, we will not read the content of user data.

Our method has performance overhead when the VM is running, our study shows that it has less than 5% overhead. The rest of this paper is organized as follows. Section II describes and explains how MLFS works and why it can facilitate the adoption of Cloud Computing. For Service providers, MLFS can use storage resourced more efficiently, and it can makes cloud computing infrastructure more robust, efficient, and available. Section III explains how MLFS is implemented.Section IV illustrate the performance and capacity evaluation results of MLFS. In section V, we review the related work to reduce space requirements in cloud computing.

978-1-4799-7470-2/14/$31.00 ©2014 IEEE
Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 22:59:31 UTC from IEEE Xplore. Restrictions apply.

Finally, we conclude this paper and present the future work of this study.

II. DESIGN

As a step to use the MLFS in the cloud service, we investigate how much benefit can be obtained. Chulmin illustrates fixed deduplication algorithms can get 77% deduplication ratio, and Rabin's FP can get 70% deduplication ratio for VM images[4]. Meyer collected file system content data from 857 desktops; their study show that whole-file deduplication achieves about three quarters of the space saving of the most aggressive block-level deduplication for storage of a live file system[7].

When designing MLFS, our goal was to share same files between different VMs. We believe that our approach is more practical than deduplication. We identify the same files before VM instances creating, while deduplication finds same blocks between VM instances after they shut down. As shown in figure 1 MLFS combines OS data and user data to a VM image. The image is interpreted by a Virtual Machine Monitor as a root file system. When a user applies for a new VM, he can select OS type, then we choose corresponding OS data; He then needs to choose applications, such as gcc, mysql, or apache. According to the user's selection, we create a disk image including selected applications use a previous image; At last, we prepare an empty space for user data. MLFS will map these three layers together, the user data layer can be modified and the other two layers are read-only. IT administrators and software developers can create these three images through offline operations using built-in or third-party tools. After a user shuts down his virtual machine, the OS image can be used by other users. The application data can be deleted and reused by the storage resources; only the user data should be kept for the next run. It's easy to migrate VM from one data center to another, because we only need to move user data to a remote site.

A. Architecture

From the view of OS, MLFS is implemented under VFS as shown in figure 2. It provides file APIs as another UNIX file system. MLFS unites three layers as one file system similar to unionfs[8]. User data, application data, and OS data can be located on any kind of file systems. In MLFS, data is divided into three layers: User layer, Application layer, and OS layer. Each layer is assigned a precedence. A layer with higher precedence overrides a layer with a lower precedence. If a directory exists in two layers, VMShareFS will merge these directories into one. If a file exists in two layers, the file in the higher layer will be shown in the final view, and the file in the lower-priority layer will be ignored. If users try to delete some files in the Application layer or OS layer, we will create a mask dummy file in the User layer to make sure the file will not appear at the final view.

We apply a stackable file system in the cloud environment in order to improve the utilization of storage with performance guaranteed. We compare the following three storage solutions:

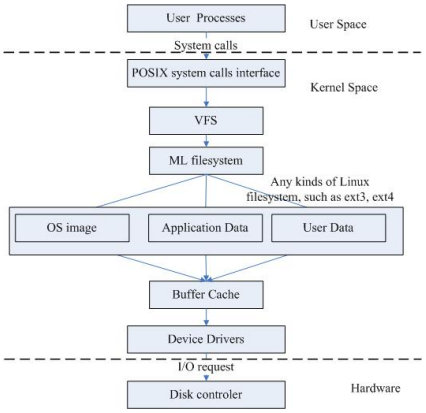


Fig. 2. ML file system in OS stacks

the first is the traditional storage solution without any optimization technology, the second is the deduplication solution, the last is our stackable file system solution.

B. Traditional Storage Solution

As shown in figure 3(a), cloud computing vendors allocate storage space for each instance separately. There aren't any technologies to reduce the redundant data between images of VMs. If m users created n VMs with a 4GB disk, then it needs 4*n GB spaces to store these images. Some vendors try to use tools(e.g. resize2fs) to resize the file system after VM shuts down. These tools rearrange the postion of files and remove the unused space from images. In this solution, VM images are difficult to migrate between data centers because of high bandwidth requirement.

C. Deduplication Storage Solution

Deduplication technology is widely used in backup systems; it is also useful to save storage requirements for cloud platforms. There are many same files between VM instances. Deduplication technology can find these same blocks and save them only once. As shown in 3(b), the first image shares several blocks with other images. This solution can reduce about 70% storage requirement according to past studies. Deduplication technology is often used in secondary storage systems, because it will decrease read and write speed in massive storage systems. When users create a new instance, it needs to calculate the md5 of each pieces of the images and compare it with existing instances. When the VM boots up, files in corresponding disk image may be stored in separated blocks; this will change the sequence read to disperse read. When multiple VMs using the same blocks are running, the share blocks may be hot data and cause high delay in reading and updating operations. Deduplication technologies reduced the space requirement for storing the VM images. It is also difficult to migrate a VM between data centers. Deduplication needs extra space to save indexes of each block and extra compute resource to calculate the hash code and compare it with existing index values.

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 22:59:31 UTC from IEEE Xplore. Restrictions apply.

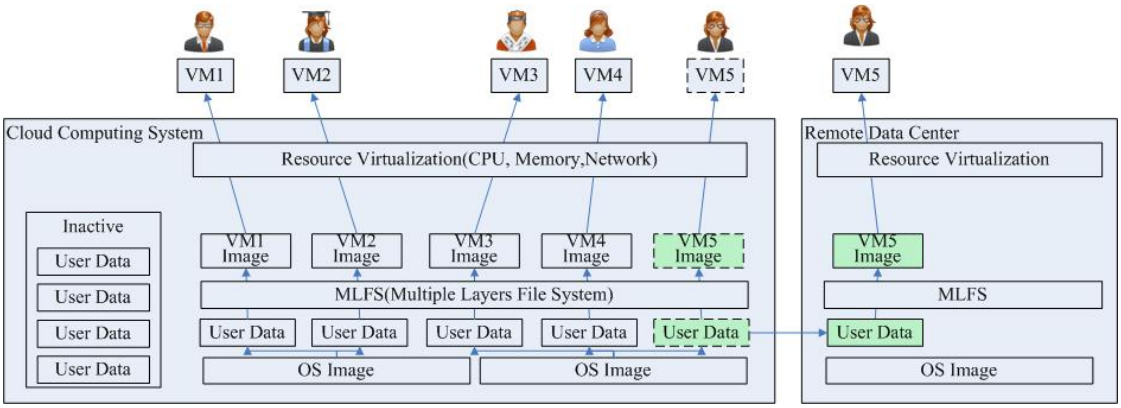
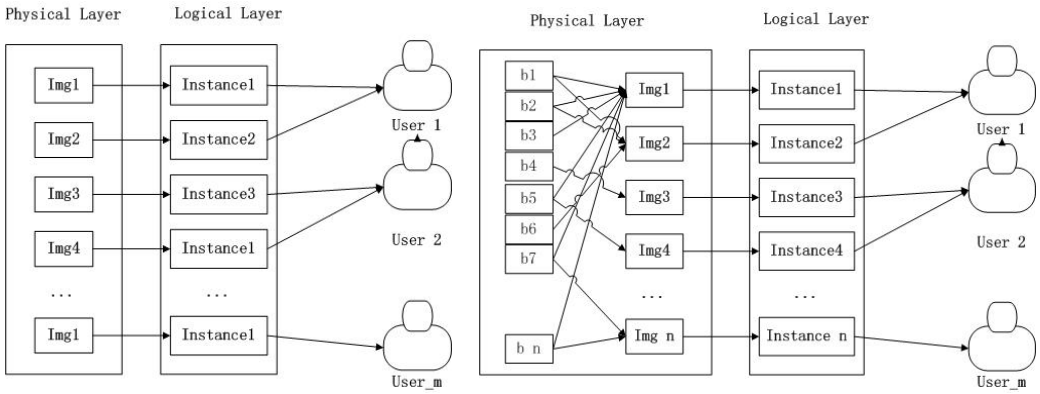


Fig. 1. ML file system usage in cloud environment



(a) Traditional Storage Solution (b) Deduplication Storage Solution

Fig. 3. Previous Solutions

D. Multiple Layers Storage Solution

We present a new solution to store VM images in the cloud computing environments. We divided the files in VM images into three categories as shown in Figure 4. The first is OS data, it includes all files created during OS installation and all patches for OS. The second is Application data, it refers to the files created by certain applications, such as Apache, gcc, and Mysql. We install Apache on a clean OS prepared in step 1, and then we compare and save the changed files in a separated location. Apache and its dependencies are saved in the location. We can prepare other applications in the same way. But when users select several applications, there may be a conflict between dependencies. In that condition, users have to solve the conflict by selecting correct one. The last part is User data, it refers to the files created or modified by the users. We put these three different kinds of data onto three different partitions and use MLFS to map them into one file system. These three parts have different priorities, user data layer is highest and OS data layer is lowest. All modifications on these file systems will be saved in user data layers. When a user applies a VM instance, he can choose the operating system and applications. According to the selection, we prepare OS data images as the bottom data layer. And we generate an application data layer from the template of each applications. For example, if a user wants use Apache and Mysql in his VM instance, we create image and copy files from Apache and Mysql templates. These templates can be created by the following steps:

1) Install OS in a VM instance.
2) After shutting down the instance, make a copy of the VM image, named with clean_VM_image.
3) Boot up VM instance and install one application and related packages.
4) Shut down the instance; rename the VM image with app_VM_image.
5) Use an offline tool to compare clean_VM_image and app_VM_image; create the directory tree and different files in a new partition. The difference between these two images can be viewed as an application template.

After the user shuts down the instance, OS data layer and application data layer can be used by other users or retrieved depending on the vendors' policy. We compare advantages and disadvantages of these three storage solutions in table I. Using MLFS in a cloud environment to organize images has the following advantages:

1) It uses less storage space than traditional solutions.

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 22:59:31 UTC from IEEE Xplore. Restrictions apply.

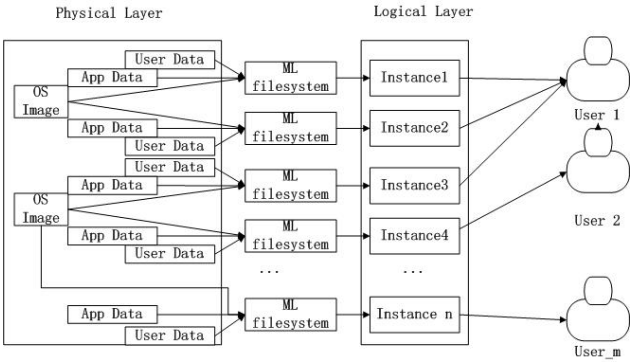


Fig. 4. MLFS storage solution

Because it checks different files instead of blocks, it uses more space than the deduplication solution.

2) In the situation when a user wants to transfer his VM instance to another data center, the traditional solution has to transmit all instance data; deduplication can calculate the hash index values for each block of instance to be transmitted and check the index list in the remote site to judge if the blocks should be transmitted. It decreases the transmit amount, but it needs to calculate and store the hash index values. In our solution, MLFS saves changed data in user data layer, so user data layer is enough to re-create a same instance in another data center.
3) It is easier to backup users' data in MLFS solution. In our solution, all OS data and application data are shared between users. We can regenerate OS image and application by installing media. So we only needs to backup the top-level user data to save the users' VM environment.
4) When a VM boots up, it updates files inside the guest OS. VMs in traditional solutions share nothing, so the performance is highest. VMs stored by deduplication share blocks, the higher deduplication ratio, the slower read/write performance. While in VMs using MLFS, vendors can decides how many VMs share one image; this is adjustable. And the application layer and user data layer is exclusively used by each VM.
5) In cloud computing systems, there are thousands of instances. Deduplication needs extra space to store increasing hash values of blocks in new instances. It also costs more time to identify same blocks in massive storage systems. MLFS stores same data(OS and application data) and different data(User data and changed files) separately, so it uses constant time to create and save a new instance.

Compared with other solution, MLFS also has some disadvantages:

1) Deduplication is a full virtualization solution, it can support any guest OS and does not change anything of the guest OS. MLFS must be supported by the guest OS; it can be implemented in Unix-like systems and does not support windows.
2) When users create a new instance, we need to generate the application data from templates, which delays the boot up time. Because the generation time depends on how many applications the users choose, we did not count the delay.
3) Our implement searches files in three layers when the guest OS opens a file and merge the changed file when closing the file. In these two operations, our solution has performance overhead.

TABLE I. COMPARISON OF THREE STORAGE SOLUTIONS

Comparision	Tranditional	Deduplication	Multi-Layers
Efficenty	low	middle	high
Backup	difficult	middle	easy
Migration	difficult	difficult	easy
Performance	high	low	middle
Expansibility	low	middle	high
virtualization	Full	Full	Paravirtualization

III. IMPLEMENTATION

One of the main special characters of MLFS is that it uses 3 file systems to handle different usage-level files, so the key point of MLFS implementation is to locate the position of a file or directory. As we mentioned in section II, there are 3 priorities for the file system layers, so when opening a given file or directory, it will search the object by the priority level from the user data layer to the app data layer and the OS image layer at last. We divided the file system APIs into two groups, which are file I/O operations and meta-data operations in order to conclude the API set that must be modified to implement its core functions. We will discuss the processing approaches in the following 3 parts in detail.

A. File Open

File open is the first step of file accessing. There are several modes for the open method which will lead to some different succeeding operations. According to different open modes, we will take different dealing procedures.

- RO(READ ONLY)
  RO mode means that the user will only get the information from the file instead of modifying it, so it only needs to search which layer the object file belongs to, and return the file pointer of it and other useful information when it is found. Otherwise, if it finds nothing or gets a mask-file from the user data layer, it returns directly.
- RW (READ and WRITE)
  RW mode indicates the object file will be changed after being opened, so the precessing steps depend on where the file is. When a RW-mode open request comes, the system sets the C_flg to 0 at first, then executes one of the following 3 processing steps, if the file is
  1) In the user data layer. Return necessary information directly.

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 22:59:31 UTC from IEEE Xplore. Restrictions apply.

2) In the non-user data layer(refers to App data layer or OS image layer). Set C_flg(this flag will be used in the "file close" processing steps) to 1 at first, copy the metadata from the original file to the user data layer and merge the metadata to the new file after it is created and return the file pointer and other useful information.
3) Not found or found as a mask-file. Create a new file and return its file pointer and other information.

- A (APPEND)
  Append mode needs the same processing steps as the steps for the RW mode exactly.

B. File Close

As we know, file migration will happen from the non-user data layer to the user data layer after a file is opened in RW or appending mode which is located in the non-user data layer originally. When the requirement of file close requirement approaches, the system checks the value of C_flg firstly, if it is 0, the file is closed in the user data layer directly, if not, it closes the file after copying its data from the original layer to the user data layer and merges it with another part of the same file which is created during the write/appending operation.

C. File(Directory) Remove

In the MLFS, the remove method includes two levels, directory removement and file removement. When a remove request is approached, the system performs the following algorithm:

① Check whether the operation object is a file or a directory, if it is a directory go to step ②, otherwise go to step ④;
② If the directory is in non-user data layer, create a mask file in the user data layer and go to step ⑤. If not, go to the next step;
③ If the directory is empty (it is also empty if it only has mask files), delete it directly and go to step ⑤. If not, perform this algorithm with a for loop execution to handle each object in this directory, and then go to step ⑤;
④ If the file is located in the user data layer, delete it directly. If it belongs to the non-user data layer, create a mask file in the user data layer. go to the next step;
⑤ stop and return.

IV. Analysis

In our experiments, we use three Linux servers as our testbed shown in table II. Our cloud computing platform uses OpenStack and KVM. We allocate VMs to students to do homework. We compare the capacity and performance of three different storage solutions.

TABLE II. Configurations of Testbed

Name	Description
Hardware	Inspur AS300N 1*Intel(R) Xeon(R) CPU E5502 1.87GHz(4 core) 1*16GBytes of RAM 4*160 GBytes SATA disk (Hitachi HTS545016B9A300)
Network	Gigabit Ethernet
Host OS	Ubuntu 12.04 Linux kernel 3.2.0
Guest OS	Ubuntu 12.04 Linux kernel 3.2.0
Cloud platform	OpenStack Essex 1.4.8
Virtualization	KVM (QEMU emulator version 1.0)

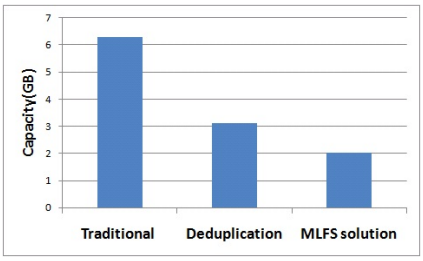


Fig. 5. Space Requirements for Different Storage Solutions

A. Capacity Comparison

We collected 8 Ubuntu 12.04 images of different users and a template image from OpenStack. The capacity of these images are 6.3GB.

We use Opendedup in our deduplication storage solution. Opendedup is an open source inline deduplication for block storage and files. It can dedup huge amounts of data and load balance across up to 126 independent nodes[1]. Opendedup presents a user space deduplication file system(SDFS) to storage data. As a file system, SDFS can de-duplicate data, but it is difficult to integrate the file system with backup and VM manager systems. After copying all eight images to SDFS, they use 3.1GB of disk space.

As for MLFS, it is implemented inside the guest OS. We did not apply MLFS to existing VM images. We just compared files in these images to separate these files into three different layers and calculate how much space it needs. In the template image, there is only Ubuntu 12.04 installed. After comparing the other 8 images with the template image, we can get the changed and added files by users. In our environment, every student has the same homework. They installed the same application(gcc) in their VMs. The users did not change too many files in their VMs. We sum up the size of OS data, application data, and 8 different users' data, the capacity is 2.0117GB. The results are shown in figure 5.

B. Performance Comparison

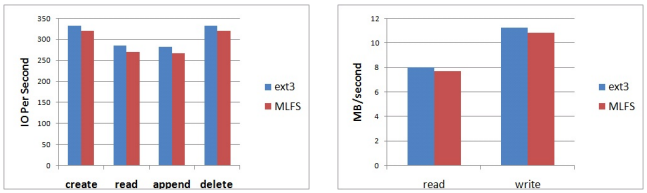
In order to compare the performance of different solutions, we boot up VM instance and use the benchmark to measure the I/O performance. In traditional and deduplication storage solutions, the VM images are saved in normal Linux file systems and SDFS. They are full virtualization solutions because they did not modify any content of guest OS. SDFS is integrated

[1] http://www.dedup.org

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 22:59:31 UTC from IEEE Xplore. Restrictions apply.

TABLE III. PERFORMANCE OF DIFFERENT SOLUTIONS

Solution	Workloads	Throughput (ops)	I/O speed (M/s)	CPU (cpu/op)	Latency (s)
Traditional	varmail	1846.832	6.7	820	24.7
	fileserver	824.094	19	1325	204.6
	webserver	12649.903	63.4	371	0.1
Deduplication	varmail	321.2	1.1	11403	147.8
	fileserver	34.815	0.8	30560	7462.3
	webserver	25.116	0.1	24782	9961.6
ML filesystem	varmail	1906.116	6.9	849	23.6
	fileserver	627.738	14.6	1375	239
	webserver	12625.166	63.3	373	0.1



(a) IOPS of ext3 and MLFS (b) MBPS of ext3 and MLFS

Fig. 7. Performance Overhead of MLFS

with guest OS. We use KVM to run a virtual machine with MLFS. KVM can use -hda/hdb/hdc options to support three IDE disk images. We save OS data, application data, and user data in each disk image.

We use Filebench[2] and test fileserver, varmail, and webserver for three different workloads inside VMs. Filebench reports Throughput, I/O transmit speed, CPU and Latency for each workload. We list the results in Table III and Figure 6 . From the table, we can find that the MLFS solution has little performance overhead comparing with the traditional solution, while deduplication solution has bad I/O performance. SDFS is an inline deduplication solution, which means that it calculates hash values of any updated blocks during any write operation inside the VMs. And SDFS is implemented as a user-space file system, the reason of bandwidth drop is not only because of dedup, but also because of data transport between user space and kernel.

C. Performance Overhead

We also compared the performance overhead of MLFS building upon ext3. We use Postmark [9] to measure the metadata operating rate and data transmit rate. We run Postmark on MLFS and user data layer(ext3) directly inside the VM. As shown in Figure 7, the results show that the performance overhead is less than 5%.

V. RELATED WORK

A. Stackable Filesystem

Stackable file systems were originally introduced in the early 1990s. They augment the functionality of existing file systems by transparently mounting a layer above them[10]. Stackable file systems can be stacked on top of each other to add multiple new features such as encryption [11], [12], anti-virus scanning (e.g., AVFS) [13], data consistency verification [14], secure deletion [15], or versioning [16] at the same time. Stackable file systems are commonly used on Linux, Windows, FreeBSD, and Solaris [17]. A different class of stackable file systems called fan-out stackable file systems, forward requests to a number of file systems mounted below (e.g., RAIF [18] and UnionFS [8]). Josef discussed issues and solutions to support stackable file system in kernel mode[19]. Aufs is a complete rewrite of the earlier UnionFS. It aimed to improve reliability and performance[20]. MLFS is a stackable file system implemented base on Unionfs.

B. De-duplicate

Data deduplication is a specialized data compression technique for eliminating duplicate copies of repeating data. Data deduplication has been widely used in disk based secondary storage systems, especially in environment of remote backup systems. Chun *et al.* propose LiveDFS, a live deduplication file system that enables deduplication storage of VM images in OpenStack. It can save at least 40% space for storing VM images[5]. Sun *et al.* present a deduplication storage system over cloud computing; they use HDFS to save files and use Hadoop database to build up an indexing system[21]. Chulmin illustrates fixed deduplication algorithms can get a 77% deduplication ratio and Rabin's FP can get a 70% deduplication ratio for VM images[4]

Our purpose is similar to data deduplication, but we did not try to find same chunks between different images, we design a mechanism to store different data at different positions, which avoids the process of analyzing the same data. Another difference is that data deduplication has bad performance when used in main storage, while our solution has little performance overhead.

C. File System for Cloud Computing

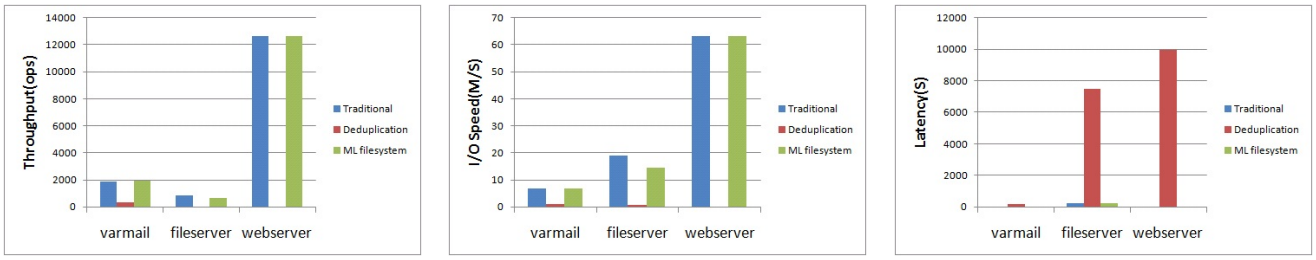
There are many file systems designed for cloud computing. Most past studies focused on how to integrate file systems with cloud storage systems. Our file system is designed for IaaS, which stores VM images efficiently. Michael *et al.* present a cloud-backed file system for the enterprise named BlueSky[22]. The file system stores data in a cloud storage provider such as Amazon S3. Liberios *et al.* propose a new file system as SaaS service of a cloud, it maps Google docs as a local file system[23]. Emil *et al.* present an authenticated file system(Iris) designed to store data in the cloud. It can enforce strong integrity guarantees[24]. qcow is a disk image format used in QEMU; it uses "'copy on write"' and optimizes allocation of storage until it is actually needed[6]. It optimize the storage usage for individual virutal machines, but it cannot find same blocks between virtual machines.

VI. CONCLUSIONS

In this paper, we present a multiple layers share file system for cloud computing environments. The file system tries to identify the same files between different VM images and store

[2]http://sourceforge.net/projects/filebench/

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 22:59:31 UTC from IEEE Xplore. Restrictions apply.



(a) Throughput (b) I/O Speed (c) Latency

Fig. 6. Performance of Different Solutions

them once to reduce space requirement. We store OS data, application data, and user data in different disk images. This design avoids hash value calculating and store in deduplication solution. According to our experiments, MLFS is faster than the deduplication solution. We also analyze the advantage and disadvantage of MLFS. MLFS is integrated with guest OS and cloud platform. It is convenient to backup or migrate user instances using MLFS. If many users update system files in their VMs, the benefits are lost. Our solution cannot handle too complex applications combines because of conflict of dependencies.

ACKNOWLEDGMENT

This work was supported by the National High-tech R&D Program of China(863 Program) under Grant No.2013AA01A215; the NFS of China under Grant No.61033007; the NFS of China under Grant No.61472323.

REFERENCES

[1] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 50–55, 2008.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5. ACM, 2003, pp. 164–177.

[3] "Kernel based virtual machine," http://www.linux-kvm.org/.

[4] C. Kim, K.-W. Park, K. Park, and K. H. Park, "Rethinking deduplication in cloud: From data profiling to blueprint," in *Networked Computing and Advanced Information Management (NCM), 2011 7th International Conference on*. IEEE, 2011, pp. 101–104.

[5] C.-H. Ng, M. Ma, T.-Y. Wong, P. P. Lee, and J. Lui, "Live deduplication storage of virtual machine images in an open-source cloud," in *Proceedings of the 12th International Middleware Conference*. International Federation for Information Processing, 2011, pp. 80–99.

[6] "qcow technology," http://en.wikipedia.org/wiki/Qcow/.

[7] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," *ACM Transactions on Storage (TOS)*, vol. 7, no. 4, p. 14, 2012.

[8] D. Quigley, J. Sipek, C. P. Wright, and E. Zadok, "Unionfs: User- and community-oriented development of a unification filesystem," in *Proceedings of the 2006 Linux Symposium*, vol. 2, 2006, pp. 349–362.

[9] J. Katcher, "Postmark: A new file system benchmark," Technical Report TR3022, Network Appliance, 1997. www. netapp. com/tech_library/3022. html, Tech. Rep., 1997.

[10] D. S. Rosenthal, "Evolving the vnode interface," in *Proceedings of the Summer USENIX Technical Conference*. Citeseer, 1990, pp. 107–18.

[11] C. P. Wright, M. Martino, and E. Zadok, "Ncryptfs: A secure and convenient cryptographic file system," in *Proceedings of the Annual USENIX Technical Conference*, 2003, pp. 197–210.

[12] M. A. Halcrow, "ecryptfs: An enterprise-class encrypted filesystem for linux," in *Proceedings of the 2005 Linux Symposium*, vol. 1, 2005, pp. 201–218.

[13] Y. Miretskiy, A. Das, C. P. Wright, and E. Zadok, "Avfs: An on-access anti-virus file system," in *Proceedings of the 13th USENIX Security Symposium (Security 2004)*, 2004, pp. 73–88.

[14] S. Patil, A. Kashyap, G. Sivathanu, and E. Zadok, "I3fs: An in-kernel integrity checker and intrusion detection file system," in *Proceedings of the 18th Annual Large Installation System Administration Conference (LISA04)*, 2004.

[15] N. Joukov and E. Zadok, "Adding secure deletion to your favorite file system," in *Security in Storage Workshop, 2005. SISW'05. Third IEEE International*. IEEE, 2005, pp. 8–pp.

[16] K.-K. Muniswamy-Reddy, C. P. Wright, A. Himmer, and E. Zadok, "A versatile and user-oriented versioning file system," in *Proceedings of the Third USENIX Conference on File and Storage Technologies (FAST 2004)*, 2004, pp. 115–128.

[17] E. Zadok, R. Iyer, N. Joukov, G. Sivathanu, and C. P. Wright, "On incremental file system development," *ACM Transactions on Storage (TOS)*, vol. 2, no. 2, pp. 161–196, 2006.

[18] N. Joukov, A. M. Krishnakumar, C. Patti, A. Rai, S. Satnur, A. Traeger, and E. Zadok, "Raif: Redundant array of independent filesystems," in *Mass Storage Systems and Technologies, 2007. MSST 2007. 24th IEEE Conference on*. IEEE, 2007, pp. 199–214.

[19] J. Sipek, Y. Pericleous, and E. Zadok, "Kernel support for stackable file systems," in *Proc. of the 2007 Ottawa Linux Symposium*, vol. 2. Citeseer, 2007, pp. 223–227.

[20] "Project of aufs," http://aufs.sourceforge.net/.

[21] Z. Sun, J. Shen, and J. Yong, "Dedu: Building a deduplication storage system over cloud computing," in *Computer Supported Cooperative Work in Design (CSCWD), 2011 15th International Conference on*. IEEE, 2011, pp. 348–355.

[22] M. Vrable, S. Savage, and G. M. Voelker, "Bluesky: a cloud-backed file system for the enterprise," in *Proceedings of the 10th USENIX conference on File and Storage Technologies (FAST'12). USENIX Association, Berkeley, CA, USA*, 2012, pp. 19–19.

[23] L. Vokorokos, A. Baláz, B. Madoš, and J. Radušovskỳ, "Cloud file system," in *Intelligent Engineering Systems (INES), 2013 IEEE 17th International Conference on*. IEEE, 2013, pp. 311–315.

[24] E. Stefanov, M. van Dijk, A. Juels, and A. Oprea, "Iris: A scalable cloud file system with efficient integrity checks," in *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM, 2012, pp. 229–238.

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 22:59:31 UTC from IEEE Xplore. Restrictions apply.