# Reduplication in Virtualization of Images in Cloud

**K. Hari Hara Kumar**[1]
[1]Department of Computer Science and Engineering
[1]Saveetha School of Engineering Saveetha University

***Abstract***— Information de duplication is a specific information pressure strategy for dispensing with double duplicates of rehashing information. This system is utilized to enhance stockpiling usage and to open-source cloud successfully diminish the plate space for putting away multi-gigabyte virtual machine (VM) pictures. Lived is one of the lived duplication document frameworks which empowers deduplication stockpiling of VM pictures in anopen-source cloud.cloud registering has developed as a standout amongst the most persuasive innovations in the IT business and is quickly changing the way IT assets are overseen and utilized.livedfs has a few different characteristics, including spatial region, prefetching of metadata, Live Deduplication Storage of Virtual Machine Images in an Open-Source Cloud

**Keywords:** Deduplication, open-source cloud system ,implementation, virtual machine image storage.

## I. INTRODUCTION

The open source cloud can be detached by using the operating system and low-cost commodity to the users. Computing and storage resources for the users on demand. the cloud computing provides users with vm on modern applications as file cache of a host level and accessing web servers for various versions of Arrangements like 32/64 bit, working framework each one document is utilized by the one terabyte. The duplication is to decreases excess information pieces by seeming little pointer in which it is as of now put away information obstruct that has indistinguishable substance. One primary requisition of de duplication is putting away the information reinforcement in Content Addressable Storage (CAS) frameworks in which every information piece is recognized by its finger impression.

### *A.* EXECUTION OF VM OPERATIONS

The employments working in every virtual machine doesn't include in the other virtual machines is working or not, yet in the break down model. Then again, it accept the first start things out server (FCFS) planning arrangement for the numerous classes model. in a few cases ist stays as execution of existing operation as vmstartup.

### *B.* SUPPORT OF GENERAL FILE SYSTEM OPERATIONS

To do the vm administration more successful it permits general operations as information change and erase processes.for the reinforcement frameworks in the current strategy it takes information to be forced a compose once approach for anticipation of information to be adjust or erase.

## II. LIVE DFS DESIGN

Live dfs is a document framework that executes in procedure duplication for VM stockpiling and is disengaged as a stockpiling layer for an open-source cloud. It is intended for current requisitions as item strategies and working frameworks. For item equipment, it reuires 32 64-bit equipment frameworks with an one tera bytes of memory. tecnically, we look forto reduce the required memory capacity to reduce the hardware cost. For commodity operating systems, we consider Linux, on which LiveDFS is developed.

We make the following assumptions for LiveDFS design. LiveDFS is deployed in a single storage partition. It only applies deduplication to the stored data within the same partition, but not for the same data stored in different partitions.

### *A.* Spatial Locality

Live dfs stores partial deduplication of metadata i.e., by using the fingerprints for storing in the memory and it is used in indexing on disk. LiveDFS exploits spatial locality by carefully placing the metadata next to their corresponding data blocks with respect to the underlying of disk layout. Fingerprint store. LiveDFS exploits *spatial* locality for storing fingerprints with respect to the disk layout of the file system. Our insight is that LiveDFS follows and organizes blocks into block groups, each keeping the metadata of the blocks within the same group. In LiveDFS, each block group is allocated a fingerprint store, which is an array of pairs of fingerprints and reference counts, such that each array entry is indexed by the block number (i.e., block address) of the respective disk block in the same block group. Thus, each block group in the disk partition has its corresponding fingerprint store. Live DFS deploys a fingerprint store in a block group. We place the fingerprint store at the front of the data block region in each block group. When Live DFS writes a new block, we write the content to the disk and update the corresponding fingerprint and reference count for the block. A key observation is that all updates are localized within the same block group, so the disk seek overhead is minimized.

### *B.* Prefetching Of Metadata

Live DFS prefetchesthe deduplication of the data blocks in the same block group into the page cache . This further reduces the seek time of updating both metadata and data blocks on the disk. Whenever LiveDFS writes a block, the fingerprint and the reference count of that block has to be updated. As a result, Live DFS has to access the corresponding fingerprint store Live Deduplication Storage of Virtual Machine Images in an Open-Source Cloud 9 on disk every time Live DFS is writing a data block. We improve the performance of our deduplication design by extending the notion of spatial locality for caching fingerprints in memory. Our key observation is that a VM image file generally consists of a large stream of data blocks. If a data block is unique and cannot be deduplicated with existing blocks, then it will be written to the disk following the previously written block. Thus, the data

 

blocks of the same block group are likely to be accessed at about the same time.

In order to further reduce the number of disk seeks, Live DFS implements a finger-print prefetching mechanism. When Live DFS is about to access the fingerprint store of a block group, instead of accessing only the target block (which contains the target fingerprint), Live DFS prefetches the entire fingerprint store of the corresponding block group and store it into the page cache, the disk cache of the Linux kernel. Therefore, subsequent writes in the same block group can directly update the fingerprint store in the page cache. The I/O scheduler of the Linux kernel will later catches the page cache into the disk. This further reduces the disk seeks involved.

*C.* Journaling.

LiveDFS supports journaling, which keeps track of file system transactions and enables crash recovery of both data blocks and fingerprints. In addition, LiveDFS exploits the underlying journaling design to combine block writes in batch and reduce disk seeks, Live DFS supports *journaling*, a feature that keeps track of file system transactions in a the file system can be recovered to a stable state when the machine fails, e.g., power outage. LiveDFS extends the journaling design in Ext3FS. In particular, we treat every write to a fingerprint store as the file system metadata and have the journal process modifications to the fingerprints and reference counts.LiveDFS updates a fingerprint store for data block P to be written to the file system, while a similar set of procedures are taken when we delete a data block from the file system. Our goal is to update the fingerprint and the reference count associated with the block P. First, we obtain the handle that refers to the journal and perform the updates of metadata . If P is a new block that cannot be deduplicated with any existing disk block, then LiveDFS first loads the corresponding block that stores fingerprint into memory and notify the journal that the fingerprint block is about to be updated via the function .After the fingerprint block is updated, we notify the journal that the modification is finished via the function ext3 journal dirty metadata(Lines 6-7). Then, we update reference count similarly (Lines 9-12). When LiveDFS releases the journal handle the journal will update the fingerprint store on disk atomically.

Function LiveDFS Fingerprint Store Update Block
Input: data block P to be written to the file system
1: handle = ext3 journal start()
2: Perform metadata writes via the journal handle
3: if P cannot be deduplicated with an existing block then
4: Load a fingerprint block fp into memory
5: ext3 journal get write access(handle, fp)
6: Update fp with the fingerprint of P
7: ext3 journal dirty metadata(handle, fp)
8: end if
9: Load P's reference count cp into memory
10: ext3 journal get write access(handle, cp)
11: Increment the reference count cp by one
12: ext3 journal dirty metadata(handle, cp)
13: ext3 journal stop(handle)

Pseudo-code of how Live DFS updates the fingerprint store through the journaling system.

*D.* Live DFS Implementation and Deployment

Live DFS is a kernel-space file system running at the top of the Linux. We implemented Live DFS as a kernel driver module for the Linux kernel 2.6.32 version, and it can be loaded to the kernel without requiring any modification or recompilation of the kernel source code. The deduplication logic is implemented by extending the virtual file system (VFS) address space operations. In particular ,operations we perform fingerprint computation and determine if a block can be deduplicated in the function writepage() which is called by a kernel thread and will flush dirty pages to the disk.

Since LiveDFS is POSIX-compliant, it can be seamlessly integrated into an open source cloud platform that runs at the top of the Linux. In this work, we integrate LiveDFS into OpenStack [22], an open-source cloud platform backed by Rackspace and NASA, such that LiveDFS serves as a storage layer for hosting VM images with deduplication. We bring up the a comparative building design as Openstack, so we expect that the sending of Livedfs in Eucalyptus takes after a comparable methodology.

Openstack outline. Openstack is based on three sub-projects compute (named Nova), Object Storage (named Swift), and Image Service (named Glance). A disentangled perspective of an Openstack cloud, which comprises of Nova and Glance just. Nova characterizes a structural planning that uses a few controller benefits that facilitate the VM occurrences running on diverse Compute hubs. Look is a VM picture administration framework that is answerable for enlisting, looking, and recovering Vmimages. It Apis for getting to a stockpiling backend, which could be Object Storage (Swift)

## III. LIVEDFS SENDING.

This shows how Livedfs is sent in an Openstack cloud. Livedfs serves as a stockpiling layer between Glance and the VM picture stockpiling backend. Overseers can transfer VM pictures through Glance, and the pictures will be put away in the Livedfs partition. when a client needs to begin a VM example, the cloud0 controller administration of Nova will allot the VM occurrence to run on one of the Compute hubs focused around the current asset utilization. At that point the doled out Compute hub will bring the VM picture from Glance, which then recovers the VM picture by means of Livedfs.

*1)* Experiments

In this section, we evaluate our LiveDFS prototype. We first measure the I/O throughput performance of LiveDFS as a disk-based file system. We then evaluate the deployment of LiveDFS in OpenStack-based cloud platform. We justify the performance overhead of LiveDFS that we observe. We compare LiveDFS with which does not support deduplication.

*2)* I/O Throughput

We measure the file system performance of different I/O operations using synthetic workload based on LiveDFS . In LiveDFS, we assume that the index key length n is approximately 19 bits and the bucket key length k is 24 bits . Note that LiveDFS is built on different design components, including (i) spatial locality, in which we allocate fingerprint stores in different block groups (ii) prefetching of a

 

fingerprint store and (iii) journaling. We evaluate different LiveDFS variants that include different combinations of the design components, as shown in Table 1, to see the performance impact of each component. When spatial locality is disabled, we simply place all fingerprint stores at the end of the disk partition; when prefetching is disabled, we bypass the step of prefetching a fingerprint store into the page cache; when journaling is disabled, we use alternative calls to directly write fingerprints and reference counts to the fingerprint stores on disk. Spatial locality Prefetching Journaling

- LiveDFS-J × × p
- LiveDFS-S p × ×
- LiveDFS-SJ p × p
- LiveDFS-all p pp

**Table 1.**Different LiveDFS variants evaluated in Section 4.1 (p = enabled, × = disabled).

## IV. EVALUATION METHODOLOGY.

We use Linux basic system calls read() and write() to measure the I/O performance. Each experimental result is averaged over 10 runs.

In each run, we use the system call gettimeofday() to obtain the duration of an operation, and then compute the throughput. At the beginning of each run, we clear the kernel page cache using the command echo 3 > /proc/sys/vm/drop caches so that we can accurately evaluate the performance due to disk accesses.

### *A.* Experiment A1: Sequential write.

We first evaluate the sequential write performance of Live DFS by writing a 16GB file with all unique blocks (i.e., all blocks cannot be deduplicated with others). The file size is larger than the 8GB RAM in our test machine, so that not all requests are kept in the kernel buffer cache. The throughput of different Live DFS variants. First, considering the Live DFS-J and Live DFS-SJ, we observe that Live DFS-SJ has a higher throughput than Live DFS-J by 16.1MB/s (or 37%). Thus, spatial locality by itself can improve the throughput, by putting the fingerprint stores close to their blocks on disk. We note that Live DFS-S (without journaling) has the lowest throughput among all variants. Specifically, Live DFS-SJ increases the throughput of Live DFS-S by 34.6MB/s (or 78.6%). Since journaling combines write requests and flushes them to the disk in Batch, journaling can effectively minimize the disk accesses in addition to providing robustness against system crashes.

We observe that Live DFS-all further improves the throughput of Live DFS-SJ via prefetching (by 13.8MB/s, or 18%). Now, comparing Live DFS-all and Ext3FS, we observe that Live DFS-all is slightly less than Ext3FS's throughput by 3.7MB/s (or 3.8%), mainly due to the increase in the block group accesses. Because we introduce a fingerprint store to each block group, Live DFS has fewer data blocks per block group than Ext3FS. However, we show that each design component of Live DFS can reduce disk accesses and increase the throughput of Live DFS close to that of Ext3FS.

### *B.* Experiment A2: Sequential read.

In this experiment, we evaluate the sequential read performance of Live DFS by reading the stored 16GB file created in Experiment A1.

### *C.* Experiment A3: Sequential duplicated write.

In this experiment, we write another 16GB file that has the identical content to the one in ExperimentA1. Figure 9 shows the results. We observe that all Live DFS variants (except Live DFS-S without journaling) significantly boost the throughput of Ext3FS by around 120MB/s. The reason is that Live DFS only needs to read the fingerprints of data blocks and update the reference counts, without re-writing the same data blocks (with larger size) to the disk. We note that the write-combining feature of journaling plays a significant role in boosting the throughput of Live DFS, as Live DFS-S does not achieve the same improvement.

### *D.* Experiment A4.1: Crash recovery with journaling.

In this experiment, we consider How duplication affects the time needed for a file system to recover from a crash using journaling. We set the journal commit interval to be five seconds when the file system is first mounted. We then write a 16GB file with unique blocks sequentially into Live DFS, and unplug the power cable in the middle of writing. The file system is therefore inconsistent. After rebooting the machine, we measure the time needed for our file system check tool which is modified from the Ext2FS utility to recover the file system.

We observe that Live DFS ensures the file system correctness using journaling. However, it generally uses a longer recovery time than Ext3FS. On average (over 10 runs), Live DFS uses 14.94s to recover, while Ext3FS uses less than 1s. The main reason is that Live DFS needs to ensure that the fingerprints in the fingerprint store match the new data blocks being written since the last journal commit interval. Such additional fingerprint checks introduce overhead. Since system crashes are infrequent, we expect that the recovery time is acceptable, as long as the file system correctness is preserved.

## V. OPEN STACK DEPLOYMENT

We now evaluate Live DFS when it is deployed in an Open Stack-based cloud. Our goal is to justify the practicality of deploying Live DFS in a real-life open-source cloud for VM image storage with deduplication. Specifically, we aim to confirm that Live DFS achieves the expected storage savings as observed in prior studies [13, 11],while achieving reasonable I/O performance of accessing VM images.

System configuration.Our experiments are conducted in an Open Stack cloud platform consisting of three machines: a Nova cloud controller, a Glance server, and a Compute node. The Nova cloud controller is equipped with an Intel Core 2 Duo E7400 2.8GHz CPU, while both the Glance server and the compute node are equipped with an Intel Core 2 Quad Q9400 2.66GHz CPU. All machines are equipped with 4GB DDR-II RAM, as well as two harddisks: a 1TB 7200RPM harddisk for storing VM images and a 250GB harddisk for hosting the operating system and required software. All machines use Ubuntu 10.04.2 server 64-bit edition as the operating system. Furthermore, all three

 

machines are inter-connected by a Gigabit Ethernet switch, so we expect that the network transmission overhead has limited performance impact on our experiments.

All VMimages are locally stored in the Glance server.We deploy either LiveDFS or Ext3FS within the Glance server, which can then access VM images via the deployed file system using standard Linux file system calls. We also deploy Kernel-based Virtual Machine (KVM) as the default hypervisor in the compute node. By default, we assume that LiveDFS enables all design components (i.e., spatial locality, prefetching, and journaling).

Our cloud testbed consists of only one Compute node, assuming that in most situations there is at most one Compute node that retrieves a VM image at a time. We also evaluate the scenario when a Compute node retrieves multiple VM images simultaneously (see Experiment B3).

*A.* Dataset.

We use deployable VM images to drive our experiments. We have created 42 VM images in Amazon Machine Image (AMI) format. Table 2 lists all the VM images.

The operating systems of the VM images include ArchLinux, CentOS, Debian, Fedora, OpenSUSE, and Ubuntu. We prepare images of both x86 and x64 architectures for each distribution, using the recommended configuration for a basic server. Networked installation is chosen so as to ensure that all installed software packages are up-to-date.

Each VM image is configured to have size 2GB. Finally, each VM image is created as a single monolithic flat file.

*B.* Experiment B1: Storage efficiency.

We first validate that LiveDFS can save space for storing VM images. Note that each VM image typically has a large number of zerofiled blocks [11]. One main source of zero-filled blocks, according to our created VM images, is due to the unused space of the VM. To reflect the true saving achieved by deduplication, we exclude counting the zero-filled blocks in our evaluation. Figure 10(a) shows the cumulative space usage of storing the VM images using LiveDFS and Ext3FS. Overall, LiveDFS saves at least 40% of space over Ext3FS (for non-zero-filled blocks). If we count the zero-filled blocks as well, then LiveDFS still it uses around 21GB of space (as all zero-filled blocks can be denoted by a single block), while Ext3FS consumes 84GB of space for the 42 2-GB VM images. In this case, we can even achieve 75% of saving. Figure 10(b) shows the average space usage of a VM image for each Linux distribution.
The space savings range from 33% to 60%. It shows that different versions of VM images of the same Linux distribution have a high proportion of identical data blocks that can be deduplicated. Therefore, our LiveDFS implementation conforms to the observations that deduplication is effective in improving the storage efficiency of VM images [13, 11]. We do not further investigate the deduplication effectiveness of VM images, which has been well studied in [13, 11].

*C.* Experiment B2: Time for inserting VM images.

In this experiment, we evaluate the time needed for inserting VM images into our Glance server. We execute the commands euca-bundle-image, euca-upload-bundle, and euca-register1 to insert the VM images from the cloud controller to the Glance server (over the Gigabit Ethernet switch).We repeat the test five times and obtain the average. Our goal is to measure the practical performance of writing VM images using LiveDFS.

Figure 11(a) shows the average insert time for individual distributions (over five runs). Overall, LiveDFS consumes less time than Ext3FS in inserting VM images. The reason is that LiveDFS does not write the blocks that can be deduplicated to the disk, but instead it only updates the smaller-size reference counts. Figure 11(b) shows the average insert time for all 42 VM images using different LiveDFS variants as defined in Section 4.1. Although this time the differences among the different LiveDFS variants are not as significant as seen in Section 4.1, we observe that enabling all design components (i.e., LiveDFS-all) still gives the least insert time.

*D.* Experiment B3: Time for VM startup.

We now evaluate the time needed to launch a single or multiple VMinstances.We assume that all VMimages have been inserted into the file system. We then execute the euca-run-instances command in the Nova cloud controller to start a VM instance in the Compute node, which fetches the corresponding VM image from the Glance server. We measure the time from the command being issued until the time when the euca-run-instances command invokes the KVMhypervisor (i.e., when the VMstarts running).Our goal is to measure the practical performance of reading VM images using LiveDFS.

Figure 12(a) shows the time needed to launch a single VM instance for different distributions in the Compute node.We observe that LiveDFS has lower throughput performance than Ext3FS. The main reason is that deduplication introduces fragmentation [24]. That is, in deduplication, some blocks of a filemay be deduplicatedwith the blocks of a different file, so the actual block allocation of a file on disk is no longer in the sequential order as seen in the ordinary file system without deduplication. Fragmentation is an inherent problem in deduplication, as it remains an open issue to be solved [24].

To see how fragmentation affects the practical performance, we note that in LiveDFS, its increase in VMstartup time ranges from 3s to 21s (or 8% to 50%) for a VMimage of size 2GB. Currently, the cloud nodes are connected over a Gigabit Ethernet switch. We expect that the VM startup penalty will become less significant if we deploy the cloud in a network with less available bandwidth (e.g., the cloud nodes are connected by multiple switches that are shared by many users), as the network transmission overhead will dominate in such a setting.

We now consider the case when we launch multiple VM instances in parallel. In the Compute node, we issue multiple VM startup commands simultaneously, and measure the time for all VM instances to start running. Figure 12(b) shows the time required for starting one to four VM instances in the Compute node. The overhead of LiveDFS remains consistent (at around 30%), regardless of the number of VM instances being launched. The observation conforms to that of launching a single VM instance.

 

*E.* Related data of vm images

Existing deduplication techniques aremainly designed for backup systems. On the other hand, LiveDFS applies deduplication in a different design space and is designed for VM image storage in a cloud platform. Also, LiveDFS seeks to be POSIX-compliant, so its implementation details take into account the Linux file system layout and are different.

*F.* Backup systems.

Venti [23] is the first work of content addressable storage (CAS) for data backup. Foundation [24] is another CAS system built upon Venti, and improves

Venti with the new compare-by-value mode. It uses the Bloom filter as an in-memory indexing structure to identify new or existing fingerprints. LiveDFS uses the fingerprint filter as an in-memory indexing structure. However, the fingerprint filter not only identifies the existence of fingerprints as in the Bloom filter, but it also specifies where the fingerprint is stored on disk. Data Domain [28] also uses the Bloom filter [4] for in-memory indexing, and uses locality preserved caching to reduce random I/Os for duplicated data. Sparse Indexing [14] trades deduplication opportunities for reduced memory usage in backup systems by only deduplicating data with a few of the most similar previous copies in different backup streams. Bimodal Content Defined Chunking [12] reducesthememory overhead of chunk indexing by using chunkswith different granularities in different regions of a backup. Note that the above systems do not consider the scenario where data can be modified or deleted, while LiveDFS addresses this by associating a reference count with each data block in its deduplication design.

*G.* Usage of flash memory.

Dedupv1 [15] and ChunkStash [8] use flash memory and solid state drives (SSDs) to relieve the memory constraints of deduplication. They exploit the fast random I/O feature of flash memory by putting the fingerprints into the flash rather than in main memory. They show that they achieve competent I/O performance while requiring a small memory capacity. On the other hand, LiveDFS targets a commodity server that is not necessarily equipped with SSDs.

*H.* Scalable storage.

Extreme Binning [3], HydraFS [27] and DeDe [7] are scalable storage systems that support data deduplication. Extreme Binning exploits file similarity rather than chunk locality. HydraFS is a file-system built atop HYDRAstor [9]. DeDe is a storage system that performs *out-of-order* deduplication. Their deployment platforms are based on a distributed environment, while LiveDFS is designed to be deployed on a single commodity server.

*I.* Deduplication-enabled file systems.

ZFS by Sun Microsystems [21] and SDFS by Opendedup [20] are file systems supporting inline deduplication. However, as stated 18 Chun-Ho Ng, Mingcao Ma, Tsz-Yeung Wong, Patrick P. C. Lee, and John C. S. Lui in Section 2.2, they need a large memory capacity for enabling deduplication, mainly because they assume that the entire set of fingerprints is loaded into memory. In terms of deployment, ZFS and SDFS are mainly designed for enterprise-scale systems, while LiveDFS is designed as a kernel-space file system for commodity servers. Btrfs [5] is an open-source file system developed by Oracle that supports deduplication.WhileBtrfs is a Linux kernel module like LiveDFS, it only supports offline deduplication [19] instead of inline at the time of this writing.

*J.* VM image storage.

It has been shown [13, 11] that deduplication can significantly save the storage space for VM images. However, there remain open issues of deploying deduplication in a VMstorage system. Nath*et al.* [17] evaluate a deployed CAS system for storing VM images. They mainly focus on storage efficiency and network load, but do not evaluate the read/write throughput of accessing VM images. Liguori*et al.* [13] deploy a CAS system based on Venti [23] for storing VM images, but its read/write throughput is limited by the Venti implementation. Lithium [10] is a cloud-based VM image storage system that aims for fault tolerance, but it does not consider deduplication.
To our knowledge, LiveDFS is the first practical system that deploys deduplication for VM image storage in a real cloud platform.

## VI. CONCLUSIONS

We propose LiveDFS, a live deduplication file-system that is designed for VM image storage in an open-source cloud with commodity configurations. LiveDFS respects the file system design layout in Linux and allows general I/O operations such as read, write, modify, and delete, while enabling inline deduplication. To support inline deduplication, LiveDFS exploits spatial locality to reduce the disk access overhead for looking up fingerprints that are stored on disk. It also supports journaling for crash recovery. LiveDFS is implemented as a Linux kernel driver module that can be deployed without the need of modifying the kernel source. We integrate LiveDFS into a cloud platform based on OpenStack and evaluate the deployment. We show that LiveDFS saves at least 40% of storage space for different distributions of VM images, while its performance overhead in read/write throughput is minimal overall. Our work demonstrates the feasibility of deploying deduplication into VM image storage in an open-source cloud.

In this work, we mainly focus on deduplication on a single storage partition. Since a cloud platform is typically a distributed system, we plan to extend LiveDFS in a distributed setting (e.g., see [16]). One challenging issue is to balance the trade-off between storage efficiency and fault tolerance. On one hand, deduplication reduces the storage space by removing redundant data copies; on the other hand, it sacrifices fault tolerance with the elimination of redundancy. We pose this issue as future work.

## REFERENCES

[1] Amazon EC2. http://aws.amazon.com/ec2.
[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, Rabkin, I. Stoica, and M. Zaharia. A View of Cloud Computing. Comm. of the ACM, 53(4):50–58, Apr 2010.
[3] D. Bhagwat, K. Eshghi, D. Long, and M. Lillibridge. Extreme binning: Scalable, parallel deduplication for chunk-based file backup. In Proc. IEEE MASCOTS, pages 1–9. IEEE, 2009.

 

[4] B. H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. Communica-tions of the ACM, 1970.Btrfs.
[5] http://btrfs.wiki.kernel.org.
[6] M. Cao, T. Tso, B. Pulavarty, S. Bhattacharya, A. Dilger, and A. Tomas. State of the art: Where we are with the ext3 filesystem. In Proc. of the Ottawa Linux Symposium (OLS), 2005.
[7] A. Clements, I. Ahmad,M. Vilayannur, and J. Li. Decentralized deduplication in SAN cluster file systems. In Proc. USENIX ATC, 2009.
[8] B. Debnath, S. Sengupta, and J. Li. ChunkStash: Speeding Up Inline Storage Deduplication Using Flash Memory.In Proc. USENIX ATC, 2010.
[9] C. Dubnicki, L. Gryz, L. Heldt, M. Kaczmarczyk, W. Kilian, P. Strzelczak, J. Szczepkowski, C. Ungureanu, and M.Welnicki.Hydrastor: A scalable secondary storage. In Proc. USENIX FAST, 2009.
[10] J. G. Hansen and E. Jul. Lithium: Virtual Machine Storage for the Cloud. In Proc. of ACM SOCC, 2010.
[11] K. Jin and E. L. Miller.The effectiveness of deduplication on virtual machine disk images.In Proc. ACM SYSTOR, 2009.

 