SED: SCALABLE & EFFICIENT DE-DUPLICATION FILE SYSTEM FOR VIRTUAL MACHINE IMAGES

Author Name: Poonguzhali. S B. Tech,

Information Technology,

M.A.M College of Engineering,

Tiruchirapalli.

kuzhalit@gmail.com

Abstract **— Virtualization is becoming widely deployed in servers to efficiently provide many logically separate execution environments by reducing the demand for physical servers, so this approach reserves physical CPU resources. Nevertheless, it still consumes large amounts of storage because each virtual machine (VM) instance, needs its own multi-gigabyte disk image. Existing systems take efforts to reduce VM image storage consumption by means of de-duplication within a storage area network (SAN) cluster. Nonetheless, a SAN cannot satisfy the increasing demand of large-scale VM hosting to cloud computing because of its cost limitation. The system proposes a SED (scalable & Efficient De-duplication) file system that has been predominantly designed for large-scale VM consumption. Its design provides hasty VM deployment with peer-to-peer (P2P) data transfer and low storage consumption by means of de-duplication on VM images. It also provides an inclusive set of storage features including on-demand fetching through a network, instant cloning for VM images, and caching with local disks by copy-on-read techniques. Experiments show that proposed system's features perform well and introduce minor performance overhead. It shows that simply identifying zero-filled block, even in ready-to-use virtual machine disk images available internet can provide considerable savings in storage.**

Index Terms — Cloud computing, de-duplication, file system, peer to peer data transfer, file storage, virtual machine

1. INTRODUCTION

Cloud computing is highly used domain in IT industry. The cloud computing provides simplified file system maintenance and scalable resource management of the virtual machine. VM is a program or OS, usually an environment. It does not physically exist, but is created within another environment. In commonly VM known as gust. VM is the basic technology of cloud computing, so managing virtual machine performance is the big issue in recent years. The main drawback of VM images is a performance overhead of a fully virtualized architecture. But the CPU industry provides the advanced hardware and hypervisors for the problem.

The merging of VM images begins moving to increase the burden on the underlying storage system. To manage the underlying storage system problem commonly uses NAS and SAN, while the host machine's direct attached (DAS) is only used for ephemeral storage. The network storage systems cost is several times more than DAS, so thousands of VM image could be an extremely challenging problem for network storage

systems because of the significant scale of storage consumption.

Many researchers provide the solution for managing the storage consumption issue brought by a large number of VM images could be addressed by de-duplication techniques. This approach most used the archival systems. The de-duplication techniques are implemented to the VM image through the SAN in the existing system. SAN work in de-centralized manner, such that de-duplication is done on the host machines running VMs and unique data blocks are then stored on the SAN cluster. SANs are mainly helpful in backup and disaster recovery technique. Within a SAN, data can be transferred from one storage device to another without interacting with a server [1].

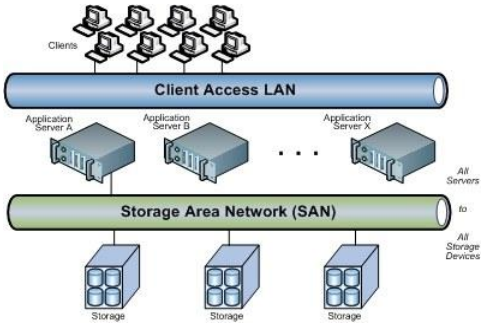

Fig.1. SAN Cluster

The SAN implementation cost is very high and thus difficult to satisfy the ever growing needs of VM image storage in future. The above addresses problems are overcome through the SED file system, which is a distributed file system also face in the large scale VM deployment. The process of SED file systems is, first the client side created VM image is split into several small data blocks. These data blocks are identified through their fingerprint (using Hash algorithm) and using de-duplication techniques to avoid storing redundant data blocks so it improve the storage space. The de-duplicated data blocks are stored in the data sever and their fingerprints are saved into the meta server. When the client access VM the SED file system first retrieve the set of fingerprints from the meta server and the fingerprints are matched data server matched data blocks are downloaded from the data server and it also use peer to peer data block transfer from nearest clients.

In P2P protocol reduces the request time through direct access of data servers, uses DAS on each host machine more effectively, and guarantees high scalability of the whole system. The SED file system provides fast VM deployment with fast cloning and on demand data block fetching, so the VM image is created a few milliseconds. SED also reduce network IO through copy-on- read technique, which is cached previously accesses data blocks on DAS. The main target of this approach is high availability of data blocks for P2P transfer.

The main purpose of this paper is to provide de-duplication file system with low storage consumption and high performance IO, which satisfies the requirement of VM hosts. The P2P data block sharing protocol provides high scalability for large-scale VM image. The additional techniques (i) VM image cloning, (ii) on-demand data block fetching, (iii) copy-on-read (iv) fault tolerance all are provided to reduce the network IO and expediting the VM creation and high availability.

The remaining part of this paper is explained as follows section 2 explain background information. Section 3 provides the design and implementation of SED. Section 4 provides related work. Section 5 dicribe the performance and section 6 concludes the paper.

2. BACKGROUND

2.1 VM Image Patterns

Virtual machine image has two basic patterns, (i) raw image pattern, (ii) sparse image pattern. The raw image pattern is defined by copying byte-by-byte content from the physical disk. The main advantages of raw image pattern is that have better IO performance why because their byte-by-byte mapping is a simple process for identifying content present in the physical disk, so the raw image pattern contains very large size data blocks.

The second pattern that is sparse image pattern is nothing to simply doing a byte-by-byte copy, but it has a complex mapping between content blocks and physical disks, data blocks in VM images. In this pattern contain a special block, who contain only zero bytes. The special mark is attached to the zero byte blocks on block mapping, so that the blocks are not stored, since their content could be easily regenerated when necessary. This procedure reduces the size of newly created VM images, since most blacks inside the images would not be used, which blocks only contain zero bytes, and hence, do no need to be stored. It reduces the storage, but the sparse image pattern does not give a good performance of

network IO compared to the raw image pattern [20]. And the complex block mapping produces the additional overhead. All types of hypervisors use both formats of VM images. The most of the hypervisors support raw image pattern. The Xen and KVM use the qcow2 sparse format and virtual box to support the vdi sparse format.

2.2 De-duplication Techniques

Data de-duplication is a specialized data compression technique for eliminating duplicate copies of repeating data [8], which is improving storage utilization. The de-duplication process uses the fingerprint to identify the unique blocks of data. The fingerprint is calculated during the compression process. When the data blocks are stored the fingerprint calculated and compared with previously stored data block's fingerprint. If the fingerprints are matched mean that data blocks are not saved to the data server. In this way, storage space is improved. For archival systems, research has shown that de-duplication could more effective than conventional compression tools. Tfhe de-duplication technique is implemented based on the whole file, or sections in a file. The section wise de-duplication process is more effective. Basically two methods are followed to break a file into small sections; there is (i) fixed size section method, (ii) variable size section method.

The fixed size section method divides the whole file into same size blocks. The variable size section method follows a more complicated dividing process. In this method, divide the file based on Rabin fingerprint on a sliding window and detects more natural boundaries inside the file. Compared with variable size section method, fixed size section method is simpler and have good read performance, but it cannot handle non-aligned insertion in the files effectively. Many researches have proved the fixed size section method is good for VM images de-duplication process.

3. DESIGN AND IMPLEMENTATION

3.1 System Architecture

The SED file system has three components, that is a single meta server with hot backup, multiple data servers, and multiple clients (see Fig. 2). VM images are divided into fixed size data blocks. Each data block is represented by its unique has a value, which is calculated during the de - duplication process. The SED file system represents a VM image through a sequence of hash value which refer to the data blocks inside the VM image. The meta server maintains information about file system layout. This includes file system namespace, fingerprint of data blocks in VM images, mapping from fingerprints to data servers, and reference count for each data block. To ensure high availability, the meta server is mirrored to a hot backup shadow meta server. The data servers are in charge of managing data blocks in VM images. They are organized in a distributed hash table (DHT) [27] fashion, and governed by the meta server. Each data server is assigned a range in the fingerprint space by the meta server.

The meta server periodically checks the health of data servers and issues data migration or replication instructions to them when necessary. A SED file system client provides a POSIX compatible file system interface via the FUSE [4] toolkit. It acts as a transparent layer between hypervisors and the de-duplicated data blocks stored in the SED file system. The client is a crucial component, because it is responsible for providing de-duplication of VM images, P2P sharing of data blocks, and features like fast cloning. When starting a new VM, the client side of SED file system file system fetches VM image meta info and data blocks from the meta server, data servers and peer clients, and provides image content to hypervisors.

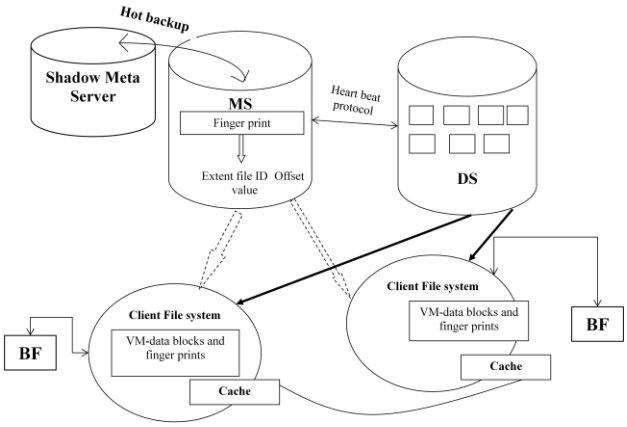

Fig. 2. System Architecture

After the shutting down of VMs, the client side uploads modified metadata on meta server, and pushes new data blocks to data servers, to make sure that the

other client nodes can access the latest version of image files. The SED file system offers fault tolerance by mirroring the meta server, and by replication on stored data blocks. When the meta server crashes, the backup meta server will take over and ensure functionality of the whole system. Replicas of data blocks are stored across data servers, thus crashing a few data servers will not impair the whole system.

3.2 De-duplication in SED

3.2.1 Fixed Size Method

The SED file system selects fixed size method applies well to the OS and software application data, why because the OS and software application data are read-only so they will not be modified once written in a VM image. The main advantage of the fixed size method is very simple. The data blocks are in same size so the mapping process is very easy. The address offset value is calculated in a simple manner so the de-duplication ratio is high in fixed size method.

3.2.2 Block size selection

The important factor to manage both de-duplication ratio and IO performance. The smaller block size will lead to higher de-duplication ratio, because modifications of the VM images will result in a smaller amount of additional data to be stored. So select the smaller block size [16]. Few more features are the smaller block size leads to data blocks to be analyzed. It also improves IO performance greatly. The smaller block size is achieved arbitrary seeks within an average search time when accessing a VM image, which is also not acceptable.

Mostly select block size smaller than 4 KB why because several OS align files on 4 KB boundaries, so smaller than 4 KB blocks improve the performance. Smaller block size improves the de-duplication ratio, but IO performance is reduced using smaller block size. Mostly select block size is multiplication of 4 KB to 256 KB and 1 MB, which is providing high IO performance and de-duplication ratio.

3.2.3 SED's Fingerprint Calculation

The fingerprint is a collision-resistant hash value that is calculated from data block contents. The fingerprint value is mainly used to identify the redundancy of data blocks. Normally MD5 and SHA-1 are two cryptography, hash functions used for calculating the fingerprint. The calculation of the fingerprint is relatively expensive. It is a major bottleneck for real time de-duplication. To avoid such expensive fingerprint calculations while a VM image is being modified, SED file system delay fingerprint calculation for recently modified data blocks, runs de-duplication lazily only when it is necessary. The client side of SED file system file system maintains a shared cache. That shared cache contains recently accessed data blocks. Data blocks in a shared cache are read-only and shared among all VM images currently opened. When being requested for a data block, the SED file system first tries to look it up in a shared cache. A cache miss will result in the requested data block been loaded into the shared cache. When the shared cache is filled up, cached data blocks are replaced using the least recently used (LRU) [9] policy. This layer of caching mechanism improves reading performance of VM images and ensures smooth operation of VMs.

Another portion of memory is used by the client side of SED file system file system as private cache, which contains data blocks that are only accessible from individual VM images. Private cache is used to hold modified data blocks, and delay fingerprint calculation on them. When a data block is modified, it is ejected from the shared cache if present, added to the private cache, and assigned a randomly generated private fingerprint instead of calculating a new fingerprint on-the-fly. This modified data block will then be referred to by the private fingerprint, until it is ejected from the private cache. The private fingerprint differs from normal fingerprint only by a bit flag, and part of it is generated from an increasing globally unique number, which guarantees that no collision will occur. The modified data block will be ejected from the private cache when a hypervisor issue a POSIX flush () request, or the private cache becomes full and chooses to eject it based upon the LRU policy. Only then will the modified data block's fingerprint be calculated.

This layer of caching mechanism improves writing performance of VM images and avoids repeated invalid fingerprint calculation to ensure the effectiveness of de-duplication. In order to speed up a de-duplication process, the SED file system uses multiple threads for fingerprint calculation. One of the concurrent threads

calculates the fingerprint for one data block at one time, so multiple threads will process different data blocks currently. When ejecting modified data blocks from private cache, instead of ejecting a single element as in conventional LRU implementations, the SED file system ejects multiple data blocks in one round. The ejected data blocks are appended into a queue, and analyzed by multiple fingerprint calculation threads. With this approach, we achieved linear speedup by increasing the number of fingerprint calculation threads, until the memory bandwidth is reached. Since fingerprint calculation is CPU intensive, and would probably contend with hypervisors, it is also possible to do fingerprint calculation on the GPU instead of on CPU [18]. Based on our experience, 256 MB of shared cache and 256 MB of private cache would be sufficient for most cases. A group of four fingerprint calculation threads will provide good IO performance.

3.2.4 Storage for Data Blocks

Fixed size method is the basic process of de-duplication, so several numbers of data blocks are stored. All the blocks are stored directly into a local file system is one of the solution. But it increases the burden of managing local file system. For example ReiserFS [6] and XFS [7] have been designed to be friendly to small files, there will still be overhead on the frequent close () and open () system calls and Linux kernel 'v' node layer. Maximum of the Linux file system use linked list storage method for storing meta data of files under a directory [19], so file 'look-up' time complexity is O (n).

To overcome the above mentioned problem, use another one solution that is to combine multiple small data blocks together into a single file, and managing them within the file. But the complex transaction process of database systems increases unnecessary overhead. The SED file system implements its own storage module for data blocks. Data blocks are divided into groups based on their fingerprints. Mainly three files are created for each group, that is (i) an extent file, it contains all the data block's content (ii) an index file, it is used for mapping a fingerprint to corresponding data block's offset and reference count, last one is (iii) bitmap file, it is used to indicate if the extent file is valid or not.

The bitmap file and index amount of memory for loading. For example, with 256 KB data block size, and 1 TB size of unique data blocks, only have an index file size of 80 MB, and a bitmap file size of 512 KB. Fig 3 shows, look-up by fingerprint could get the block's address with hashing, and the whole time complexity is O (1). Deleting a data block is a simple process that is just flipping a bit flag is the bitmap file. But inserting a new data block has some special process. First SED file system seeks for unused slots by checking the bitmap file, and reuses any invalid slot. The result is no slot mean; a new slot is added to hold the data block. After a new data block is added, its corresponding bit flag in the bitmap file will be set, indicating the slot to be under use. SED borrows much from dynamic memory allocation algorithms, and uses a free list to track unused slots with O (1) time complexity.

3.3 SED's File System

All file system metadata are stored on the meta server. The individual meta server's local file contains the each VM image's metadata, and structured in a conventional file system tree.

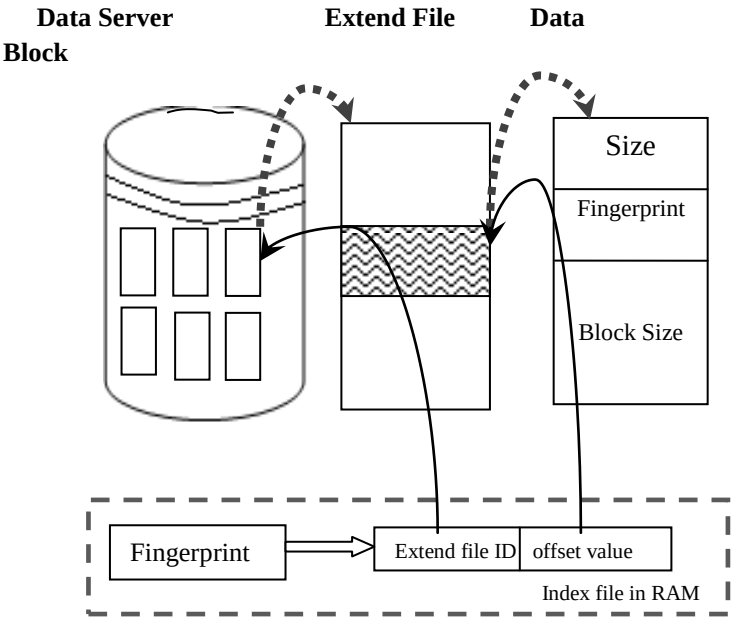

Fig.3. Process of look-up fingerprint

User-level applications are provided to fetch VM meta data files stored on the meta server. The client side of the SED file system could cache portions of file system metadata for fast accesses. In most cases, only metadata

of a single VM image or a subdirectory containing VM images will be cached on client side. SED follows the common practice in source code management systems, where a client modifies a local copy of portions of the whole project, and publishes the modification to others by committing to a central repository.

A SED client fetches portions of file system metadata from the meta server, which contains directory listing, file attributes, and set of data block fingerprints needed by the VM image being accessed. After meta data are prepared, SED clients, then fetches data blocks from data servers and from peer client nodes (see Section 3.4.2 for details). Note that only data blocks not present on local cache will be fetched. When a VM is stopped, modified meta data and data blocks will be pushed back to the meta server and data servers, which ensures that the modification of the VM image is visible to other client nodes.

3.4 Communication Protocols

3.4.1 Heartbeat Protocol

The meta server in SED is incharge of managing all data servers. It interacts a typical heartbeat message with each data server, in order to keep an up to date vision of their health status. The meta server exchanges heartbeat messages with data servers in a round-robin fashion. This approach will be slow to detect failed data servers when there are many data servers. To speedup failure detection, whenever a data server or client encounters, connection problem with other data server, it will send an error signal to the meta server. A dedicated background daemon thread will immediately send a heartbeat message to the problematic data server and determines if it is alive. This mechanism ensures that failures are detected and handled at an early stage. The round-robin approach is still necessary since it could detect failed data servers even if no one is communicating with them.

3.4.2 P2P Data Block Sharing

One advantage of SED is its P2P data block sharing scheme. SED alleviates the burden on the data servers by sharing data blocks among all client nodes in a peer-to-peer fashion, eliminating network IO bottlenecks. However, existing peer-to-peer distribution protocol such as Bit Torrent [2] will not perform well in this case because of the sheer number of data blocks and corresponding metadata tracking overhead. Moreover, existing protocols do not utilize de-duplication info available for VM images. SED implements its own peer-to-peer distribution protocol with inspiration from Bit Torrent protocol. Each client node or a data server is a valid data block provider, and publishes a Bloom filter [3] where the fingerprints of all their data blocks are compacted into. A Bloom filter uses an array of m bits, all set to 0 initially, to represent the existence information of n fingerprints. K different hash functions must also be defined, each of which maps a fingerprint to one of the m array positions randomly with a uniform distribution. When adding a new fingerprint into the Bloom filter, the k hash functions are used to map the new fingerprint into k bits in the bit vector, which will be set at 1 to indicate its existence. To query for a fingerprint, we feed it to each of the k hash functions to get k array positions. If any of the bits at these positions is 0, the element is definitely not in the set; otherwise, if all are 1, then either the element is in the set, or the bits have been to 1 by chance during the insertion of other elements. An example is given in Fig. 4, representing the set x; y; z. The olid, dashed, and dotted arrows show the positions in the bit array that each set element is mapped to. The element w is not in the set x; y; z, why because it hash values to one bit-array position containing 0. Bloom filters have false positives. According to [3], the probability of false positive is given as

$$(1-(1-\frac{1}{m})^{kn})^{k} \sim (1-e^{-kin/m})^{k} \qquad (1)$$

In the case of 256 KB block size, 40 GB unique data blocks, 4 hash functions in Bloom filter, and a constraint of less than 1 percent false positive rate, we could calculate that the Bloom filter should contain at least around 1, 724, 000 bits, which is about 210 KB in size. In practice, we choose a Bloom filter size of 256 KB. Each client maintains connections to a set of peer clients tracked by the meta server, and periodically updates its copy of peer clients' Bloom filters. When fetching a data block by its fingerprint, it checks the existence of the fingerprint among peer clients' Bloom filters in a random order, and tries to fetch the data block from a peer if its Bloom filter contains the requested fingerprint. If the data block is not found among peers, the client will go back to fetch the data block from data servers. Checking peers and data servers in a random order to eliminate the

possibility of turning them into hot spots, and brings little additional cost because of its simplicity.

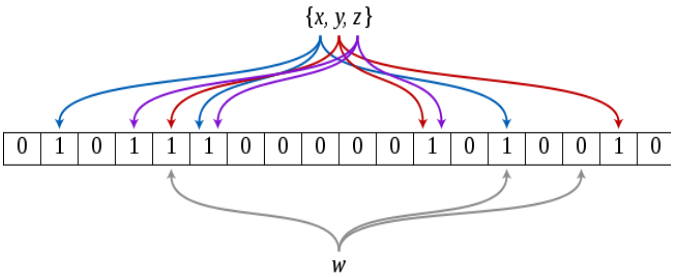

Fig. 4. Example of Bloom Filter

3.4.3 On-demand Data Block Fetching

SED uses the copy-on-read technique to bring data blocks from data servers and peer clients to local cache on demand as they are being accessed by a VM. This technique allows booting a VM even if the data blocks in the VM image were not all fetched into the local cache, which brings significant speedup for VM boot up process. Moreover, since only the accessed portion of data blocks is fetched, network bandwidth consumption is kept at a low rate, which is way more efficient than the approach of fetching all data blocks into local cache and then booting the VM. However, on-demand fetching has a relatively low read performance compared with fully cached VM images. This is because a cache miss will result in an expensive RPC call for fetching the data block being requested. This incurs several times longer IO delay compared with local disk IO. However, this problem will not impair IO performance greatly, since only the first access to such data blocks will affect IO performance. As the frequently used data blocks are fetched, IO performance will return to the normal level.

3.5 Fast Cloning for VM Image

The common practice of creating a new VM is by copying from a template VM image. Most VM images are large, with sizes of several GB. Copying such large images byte-by- byte would be time consuming. SED provides an efficient solution to address this problem by means of fast cloning for VM images. The VM disk images, as seen by SED, are represented by a metadata file containing references to data blocks. Simply by copying the metadata file and updating reference counting in data block storage, we could achieve cloning of a VM image. Since the underlying data block storage is a content addressable storage [23], the cloned VM image is by nature a copy-on-write product, which means that modification on the cloned image will not affect the original image. Due to the small sizes of meta data, VM images could be cloned in several milliseconds in the users' view.

3.6 Fault Tolerance

SED provides fault tolerance through data replication, data migration, and hot backup of the meta server. Data blocks are stored in two replicas, in case some data server crashes. When the meta server detects failure of a data server, it immediately contacts other data servers containing replicated data blocks, and sends instructions to create new replicas. Most data blocks have a replication count of more than two, since they also exist on multiple client nodes. Even if all data servers crashes, those blocks are still available through the P2P block sharing protocol. When a data server is planned to be offline, its data blocks could be migrated to other data servers. This is done by simply copying the extent files to other data servers, and to merge with existing extent files on destination data servers. Taking a data server offline with planning enables SED to handle re-replication work gracefully. The meta server is a critical component in SED. It achieves high availability with a hot backup shadow meta server. Every meta data mutation is performed on both meta servers to ensure a consistent file system view. The shadow meta server exchanges heartbeat messages with the meta server periodically. When the primary meta server is failing, the shadow meta server will take over and operate in read-only mode until an administrator sets up a shadow meta server for it. The other components will discover the failure of the origin meta server, and switch to work with the new meta server.

3.7 Garbage Collection

SED uses reference counting extensively to track usage of each data block, and removes unused garbage data blocks when running out of space. For client side of the SED file system, the data block storage contains reference counting of each data block. When a data block's reference count drops to zero, it is considered to be a garbage data block. Garbage data blocks are not immediately removed, since they might be used again sometime later. Those garbage data blocks will only be removed when the client cache is nearly full, and the extent file containing data blocks will be compacted to reduce storage consumption. For data servers, they are not responsible for collecting reference counting of data

blocks. The reference counting of all data blocks is maintained by the meta server, and it periodically issue's garbage collection requests to data servers. Based on the data server's fingerprint range, the meta server will generate a Bloom filter containing all the valid data blocks inside the range. The Bloom filter is then randomly sent to one of the replicas, and an offline garbage collection is executed based on data block membership in the Bloom filter.

4. RELATED WORK

Propose the use of deduplication to both reduce the total storage required for VM disk images and increase the ability of VMs to share disk blocks [18]. Testing the effectiveness of deduplication, conducted extensive evaluations on different sets of virtual machine disk images with different chunking strategies. Deduplication is an efficient approach to reduce storage demands in environments with large numbers of VM disk images. As we have shown, deduplication of VM disk images can save 80% or more of the space required to store the operating system and application environment; it is particularly effective when the disk images correspond to different versions of a single operating system lineage, such as Ubuntu or Fedora.

The exact effectiveness of deduplication is data-dependent hardly surprising, given the techniques that deduplication uses to reduce the amount of storage consumed by the VM disk images. This technique showed which factors influence the amount of deduplication available when deduplicating VM disk images, but they do not address the issue of locality in deduplication. Believe that deduplication of VM disk images will exhibit both temporal and spatial locality because similar files in different disk images will contain many similar chunks in the same order. This technique currently works to evaluate the amount of locality available in disk images. If there is significant locality, we can improve deduplication performance by co-locating nearby chunks from one disk image, since those chunks will likely be near one another in other disk images as well.

A prototype for consolidating virtual disk images using a service oriented file system. It provides a hierarchical group, manages historical data of drive images, and takes steps to optimizing encoding based on partition type and file system. Present these experiences with building this prototype and using it to store a variety of drive images for QEMU and the Linux Kernel Virtual Machine (KVM) [19].

A content addressable storage (CAS) approach to coalescing duplicate data between multiple disk images would provide a solution to the incremental drift of virtual disks. Additionally, the nature of CAS would obviate the need for end-users to start with a template image as any duplication would be identified and addressed by the CAS back end. Furthermore, CAS solutions lend themselves to rapid cloning, snapshoting, and can be configured to implicitly provide temporal-based backups of images.

In order to obtain a better scheme of the performance and efficiency implications of using a CAS based image management system, constructed a prototype by CAS back end with a service-oriented file system to provide an organizational infrastructure and tested it with guest logical partition and running under QEMU which provides the Virtual I/O infrastructure for KVM.

Promising efficiency improvements, it is clear that the current performance in this environment is far below what would be desirable. It was primarily developed as a backup archive server, and as such its implementation is single threaded and not constructed to scale under heavy load. Additionally, its performance is primarily bottlenecked by the requirement of in-directing block requests via the index.

LiveDFS has several individual features, including spatial locality, prefetching of metadata, and journaling. LiveDFS is POSIXcompliant and is implemented as a Linux kernel-space file system. Deploy our LiveDFS prototype as a storage layer in a cloud platform based on OpenStack, and conduct extensive experiments. A live deduplication file system called *LiveDFS*, which enables deduplication storage of VM image files in an open-source cloud. In particular, target the open-source cloud platforms that are deployed in low-cost commodity hardware and operating systems.

LiveDFS supports general file system operations, such as read, write, delete, while allowing *inline deduplication* (i.e., on-the-fly deduplication is applied to data that is to be written to the disk) [24]. LiveDFS consists of several design features that make deduplication efficient and practical. 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. 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.

FVD is a holistic solution for both Cloud and non-Cloud environment. Its feature set includes storage thin provisioning without a host file system, elastic configurability, copy-on-read,compact image, internal snapshot, encryption, copy-on-write and an adaptive pre-fetching. The last two features enable instant VM creation and instant VM migration, even if the VM image is stored on DAS. The DAS is at least several times cheaper than SAN and NAS, but DAS limits the availability and mobility of VMs. To get the best out of the different technologies, a Cloud frequently offers a combination of block-device storage services to VMs. The solution in FVD is to do copy-on-read (CoR) and adaptive prefetching [27], in addition to copy-on-write (CoW). CoR avoids repeatedly reading a data block from Network Area Storage, by saving a copy of the returned data on DAS for later reuse. Adaptive prefetching uses resource idle time to copy from NAS to DAS the image data that have not been accessed by the VM. FVD also supports instant VM migration, even if the VM's image is stored on DAS. FVD can instantly migrate a VM without first transferring its disk image. As the VM runs uninterruptedly on the intention host, FVD uses CoR and an adaptive pre-fetching to gradually move the image from the source host to the target host, without user perceived downtime. This FVD only supports neither copy-on-read (CoR) nor adaptive pre-fetching. Some virtualization solutions do support CoR or pre-fetching, but they are implemented for specific use cases.

The production Data Domain deduplication file system to relieve the disk bottleneck. These techniques include: (1) Stream-Informed Segment Layout, a data layout method to improve on-disk locality for sequentially accessed segments; (2) the Summary Vector, a compact in-memory data structure for identifying new segments; and (3) Locality conserved Caching, which maintains the locality of the fingerprints of duplicate segments to achieve high cache hit ratios [29]. To implement a high-throughput Identical Segment Deduplication storage system at low system cost. The key performance challenge is finding duplicate segments.

An in-memory index of all segment fingerprints could easily achieve this performance, but the size of the index would limit the system size and increase system cost. To maintain an on-disk index of segment fingerprints and use a cache to accelerate segment index accesses. Early deduplication hashing to the space ABC ratios, not on high through application storage systems, you detect duplicate files and reclaim their storage systems also use file hashes to address files. Some call such systems, content addressed storage or CAS. Since their deduplication is at file level, such systems can achieve only limited global compression and not duplication in every single client FS.

5. PERFORMANCE EVALUTION

5.1 De-duplication Block Size

Data block size has a direct impact on disk IO performance and de-duplication ratio. A proper choice of the data block size needs to balance these two factors in order to achieve the best result. Fig. 5 provides an analysis of IO performance under different data block sizes. The statistic results are obtained by reading data blocks stored in the client side of SED file system's local cache.

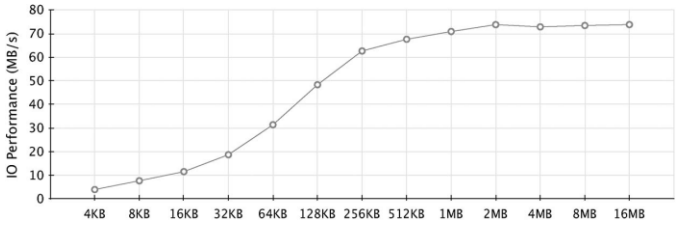

Fig. 5. IO performance under different block size

A smaller data block size results in more frequent random access operations, and in turn degrades IO performance. With the increase of the data block size, IO performance gradually improves, and stabilizes after it reaches 256 KB. On the other hand, smaller data block size has the benefit of better redundancy. Fig. 6 presents de-duplication ratio under different data block size. The results are obtained by running de-duplication on a set of 183 VM images totaling 2.31 TB. This collection of VM images includes OS of Microsoft Windows, Ubuntu, RedHat, Fedora, CentOS, and openSUSE. The number of each OS image is shown in Table 1. The applications such as Microsoft Office, MATLAB, Apache, Hadoop, MPI, etc. are installed in these OS images randomly. A disk cluster size of 4 KB is used in all VM, so de-duplicated data block size smaller than 4 KB will not bring more

advantages. For data block size larger than 4 KB, the de-duplication ratio drops quickly. Based on our experience, data block size in the range 256 KB ~1 MB achieves moderate IO performance and de-duplication ratio, satisfying common applications.

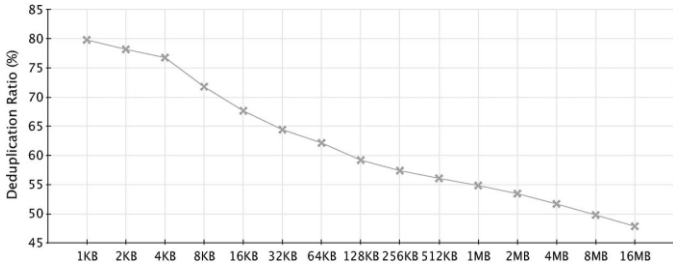

Fig. 6. De-duplication ratio inder different block size

5.2 VM Benchmark

In order to illustrate the difference in IO performance under different storage policies, we conducted a few experiments inside VM. The first benchmark is against VM disk IO performance. Bonnie++ is an open-source benchmark suite that is aimed at performing a number of simple tests of hard drive and file system IO performance. PostMark is also a common benchmark for file system developed by NetApp.

TABLE 1
Number of Each OS Image

OS Type	Number
Micro Windows	75
Ubuntu	32
Red Hat	22
Fedora	21
Cent OS	21
OpenUSE	12

We ran Bonnie++ and PostMark benchmarks on the native disk on the host machine, raw format VM image, qcow2 format VM image, and raw format image stored in SED with different data block size in range 16 KB ~2 MB. The VM images are created with a size of 50 GB, formatted as an ext4 file system. The VM is assigned 2 GB memory and 1 vCPU, running Ubuntu 10.10 with Linux kernel 2.6.35. The results are shown in Figs. 7 and 8. Native IO on host machine has the best performance. It is the criterion to evaluate other storage policies. The raw image format provides good read and write performance.

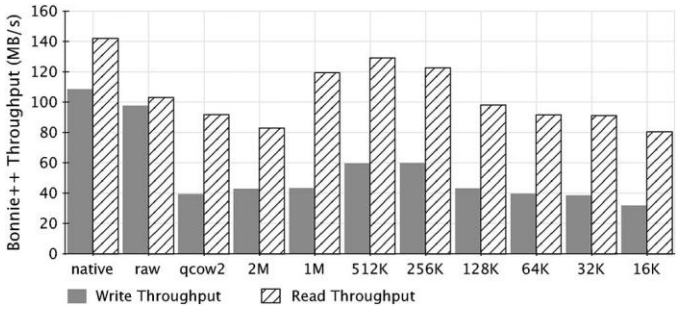

Fig. 7. Bonnie++ benchmark result

For qcow2 format images and raw images stored in SED, write performance is degraded due to the frequent need to allocate a new data block. However, SED writes a little faster than qcow2 because of its caching mechanism, even if it is running expensive de-duplication process concurrently. Read performance sees weaker impact compared to that of writing, because the data being accessed is likely to be cached by the OS.

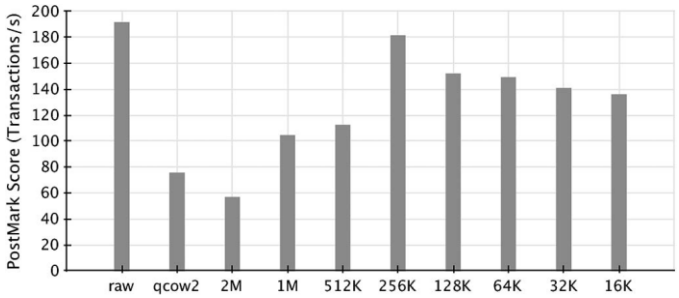

Fig. 8. PostMark benchmark result in VM

A moderate data block size (256 KB, 512 KB) will result in efficient use of cache, while larger data block size causes lower cache hit rate, and small data block size results in additional overhead due to frequent random seeks. In addition to Bonnie++ and PostMark benchmark, we also conducted a set of comprehensive benchmarks based on VM Booting and Linux source code compilation. This benchmark set is both CPU and IO intensive and is representative for most use cases. The Linux kernel source code used in these benchmarks is version 3.0.4. Benchmark results are shown in Fig. 9, with time normalized according to the longest sample.

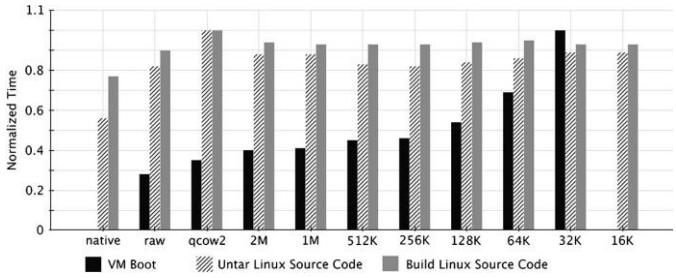

Fig. 9. Linux benchmark in VM

5.3 Network Transfer Rate

To evaluate the P2P data block sharing protocol implemented in SED, we record the total time used to transfer an 8 GB VM image from one node to another seven nodes. The VM image being transferred is a newly installed Ubuntu 10.10 instance. A moderate data block size of 256 KB is used by SED for this benchmark. Results are shown in Fig. 10. significant speedup compared to scp and NFS.

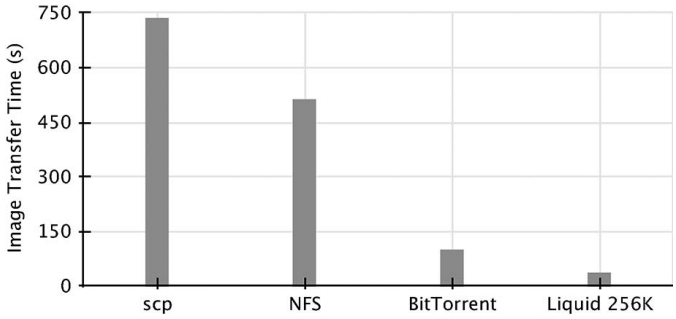

Fig. 10. Time used transfer a VM image

However, the redundancy is not utilized by Bit-Torrent protocol, and all the redundant zero data blocks are all transferred through a network. SED client avoids fetching redundant data blocks multiple times, and achieves much better distribution speed than plain Bit-Torrent protocol. The VM boot benchmark is used to evaluate on-demand fetching, with results shown in Fig. 11. Compared with normal VM booting where all data blocks inside the VM image have already been fetched into the local cache, VM is booting with on-demand fetching takes several times longer duration.

This is caused by the additional overhead to request missing data blocks from other nodes, which incurs a longer delay than local disk IO delay. As the data block size decreases, local cache misses increases and leads to more frequent IO through a network, thus results in a longer VM boot phase. However, the whole VM boot time (the downloading image time and the VM boot time) has been shortened while the data block size is between 512 k and 2 M. In fact, the on-demand fetching scheme also speeds up the virtual machine startup speed.

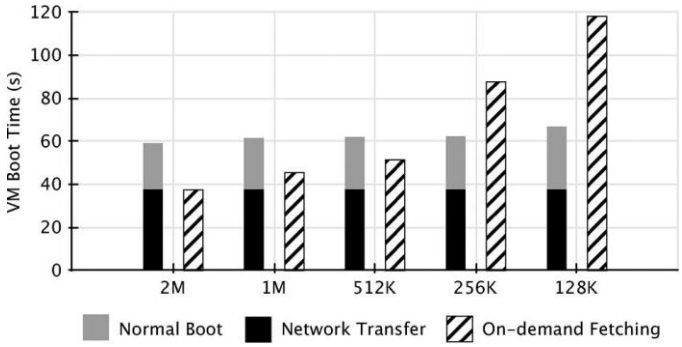

Fig. 11. VM boot time with on-demand fetching

6. CONCLUTION

The SED is a de-duplication file system with rich set of features and good IO performance. SED's IO performance is achived by implementing de-duplication process at mean time. This is achieved by caching frequently accessed data blocks in local memory cache, and the deduplication algorithms are run when it is necessary. By organizing data blocks into large group, SED avoids additional disk operations incurred by local file system. SED supports on-demand fetching through network and instant VM image cloning by copy-on-write technique, which enables fast VM deployment. The main perpose of P2P technique is to accelerate sharing of data blocks, and makes the system highly scalable. Occasionally exchanged Bloom filter's data block fingerprints enables accurate tracking with little network bandwidth consumption. De-duplication on VM images is to proved highly effective. Yet, special care should be taken to achieve high IO performance. The OS and applications are create the temporary files so the VM images modified their part of the images based on the temprory files. Caching the modified image's data blocks will avoid running expensive de-duplication technique frequently, thus increase IO performance. Making the system scalable by means of P2P technique is the challenging because of the more number of data blocks to be tracked. By using the Bloom filter for comparing data block fingerprints ,that avoid the management overhead and meta data transferred over network could.

REFERENCES

[1] AmazonMachine Image, Sept. 2001. [Online].Available: http://en.wikipedia.org/wiki/Amazon_Machine_Image

[2] Bittorrent (Protocol), Sept. 2011. [Online]. Available: http://en.wikipedia.org/wiki/BitTorrent_(protocol)

[3] Bloom Filter, Sept. 2011. [Online]. Available: http://en.wikipedia.org/wiki/Bloom_filter

[4] Filesystem in Userspace, Sept. 2011. [Online]. Available: http://fuse.sourceforge.net/

[5] Rabin Fingerprint, Sept. 2011. [Online]. Available: http://en.wikipedia.org/wiki/Rabin_fingerprint

[6] Reiserfs, Sept. 2011. [Online]. Available: http://en.wikipedia.org/wiki/ReiserFS

[7] Xfs: A High-Performance Journaling Filesystem, Sept. 2011. [Online]. Available: http://oss.sgi.com/projects/xfs/

[8] Data Deduplication, Sept. 2013. [Online]. Available: http://en. wikipedia.org/wiki/Data_deduplication

[9] A.V. Aho, P.J. Denning, and J.D. Ullman, ''Principles of OptimalPage Replacement,'' J. ACM, vol. 18, no. 1, pp. 80-93, Jan. 1971.

[10] R. Coker, Bonnie++, 2001. [Online]. Available: http://www.coker.com.au/bonnie++/

[11] D. Eastlake, 3rd, Us Secure Hash Algorithm 1 (sha1), Sept. 2001. [Online]. Available: http://tools.ietf.org/html/rfc3174.

[12] G.DeCandia, D.Hastorun,M. Jampani,G.Kakulapati,A. Lakshman, A. Pilchin, S. Sivasubramanian, P.Vosshall, andW.Vogels, ''Dynamo: Amazon's Highly Available Key-Value Store,'' in Proc. 21st ACM SIGOPS SOSP, New York, NY, USA, 2007, vol. 41, pp. 205-220.

[13] 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. 7th Conf. File Storage Technol., 2009, pp. 197-210.

[14] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica, ''Above the Clouds: A Berkeley View of Cloud Computing,'' Dept. Elect. Eng. Comput. Sci., Univ. California, Berkeley, CA, USA, Rep. UCB/EECS 28, 2009.

[15] K. Jin and E.L. Miller, ''The Effectiveness of Deduplication on VirtualMachine Disk Images,'' in Proc. SYSTOR, Israeli Exp. Syst. Conf., New York, NY, USA, 2009, pp. 1-12.

[16] M. Juric, Notes: Cuda md5 Hashing Experiments, May 2008. [Online]. Available: http://majuric.org/software/cudamd5/

[17] J. Katcher, ''Postmark: A New File System Benchmark,'' Network Applicance Inc., Sunnyvale, CA, USA, Technical Report TR3022, Oct. 1997.

[18] A. Liguori and E. Hensbergen, ''Experiences with Content Addressable Storage and Virtual Disks,'' in Proc. WIOV08, San Diego, CA, USA, 2008, p. 5.

[19] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier, ''The New ext4 Filesystem: CURRENT STATUS and Future Plans,'' in Proc. Linux Symp., 2007, vol. 2, pp. 21-33, Citeseer.

[20] M. McLoughlin, The qcow2 Image Format, Sept. 2011. [Online]. Available: http://people.gnome.org/markmc/qcow-image-format.html

[21] C. Ng, M. Ma, T. Wong, P. Lee, and J. Lui, ''Live Deduplication Storage of Virtual Machine Images in an Open-Source Cloud,'' in Proc. Middleware, 2011, pp. 81-100.

[22] K. Pepple, Deploying OpenStack. Sebastopol, CA, USA: O'Reilly Media, 2011.

[23] S. Quinlan and S. Dorward, ''Venti: A New Approach to Archival Storage,'' in Proc. FAST Conf. File Storage Technol., 2002, vol. 4, p. 7.

[24] P. Schwan, ''Lustre: Building a File System for 1000-Node Clusters,'' in Proc. Linux Symp., 2003, vol. 2003, pp. 400-407.

[25] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, ''TheHadoop Distributed File System,'' in Proc. IEEE 26th Symp. MSST, 2010, pp. 1-10.

[26] I. Stoica, R.Morris, D. Karger,M.F. Kaashoek, and H. Balakrishnan, ''Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications,'' in Proc. Conf. Appl., Technol., Architectures, Protocols Comput. Commun. (SIGCOMM), New York, NY, USA, Oct. 2001, vol. 31, pp. 149-160.

[27] C. Tang, ''Fvd: A High-Performance Virtual Machine Image Format for Cloud,'' in Proc. USENIX Conf. USENIX Annu. Tech. Conf., 2011, p. 18.

[28] J. Wei, H. Jiang, K. Zhou, and D. Feng, ''Mad2: A Scalable High-Throughput Exact Deduplication Approach for Network Backup Services,'' in Proc. IEEE 26th Symp. MSST, 2010, pp. 1-14.

[29] B. Zhu, K. Li, and H. Patterson, ''Avoiding the Disk Bottleneck in the Data Domain Deduplication File System,'' in Proc. 6th USENIX Conf. FAST, Berkeley, CA, USA, 2008, pp. 269-282.