Information Sciences 379 (2017) 3–22

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Towards an efficient snapshot approach for virtual machines in clouds

Jianxin Li*, Yangyang Zhang, Jingsheng Zheng, Hanqing Liu, Bo Li, Jinpeng Huai

School of Computer Science and Engineering, Beihang University, Beijing, 100191, China

ARTICLE INFO

Article history:
Received 20 October 2015
Revised 17 July 2016
Accepted 3 August 2016
Available online 8 August 2016

Keywords:
Cloud computing
High-availability
Virtual machine
Live snapshot
Virtual disk image

ABSTRACT

High-availability of virtual machines is of significant importance for a cloud computing environment, because cloud services may be compromised due to the system maintenance, malicious attacks, and hardware and software failures. The virtual machine (VM) snapshot can effectively back up the state, disk data, and configuration of a machine at a specific time point. However, the existing VM snapshot methods suffer from performance issues such as long downtime and I/O performance degradation during a live snapshot. To address such issues, we proposed an efficient VM snapshot system, `iROW` (improved Redirect-on-Write), based on the `qemu-kvm` virtual block device driver. `iROW` employs the following techniques. (1) A bitmap-based light-weight index method is designed to reduce the query cost of the existing two-level index table structure compared with qcow2. (2) An *index-free* approach is used for VM state data to improve the performance of data saving and loading operations of VM state. (3) VM state data is separated from disk image data in a snapshot. (4) The *free page detection* (FPD) is designed using *virtual machine introspection* to identify and skip saving free pages in the guest OS during snapshotting, thus reducing the VM state snapshot creation time and the snapshot disk space usage. Our experimental results demonstrate that `iROW` is evidently advantageous over `qcow2` in performance, in terms of the disk snapshot, the state snapshot and the disk I/O.

1. Introduction

IN the recent years, the cloud computing has emerged as an efficient paradigm enabling a ubiquitous and on-demand service access to a reliable and shared computing and storage resources [5,16]. In a cloud, the virtualization technology is popularly employed to run multiple virtual machines (VMs) with strong isolation on a single physical machine. A VM encompasses resources to facilitate the execution of a complete operating system (OS), and both user data and application data can be stored in the VM [13,19]. Unfortunately, the data reliability and service availability of the VM might be compromised by system maintenance, malicious attacks, and hardware and software failures [1,10].

The virtual machine snapshot is a file-based approach greatly enhancing the availability of the data and services in the VM. It can back up a VM at a specific time point by the disk snapshot saving the disk of the VM, and the state snapshot which saves the running state of the VM including the states of the virtual CPUs and of all connected virtual devices of the VM [2,6,10,26]. Moreover, the VM can be easily reverted to any previously saved state, and then can be allowed to run continuously [7,8].

* Corresponding author: Fax: +86 10 82339274.
E-mail addresses: lijx@act.buaa.edu.cn (J. Li), zhangyy@act.buaa.edu.cn (Y. Zhang), zhengjs@act.buaa.edu.cn (J. Zheng), liuhq@act.buaa.edu.cn (H. Liu), libo@act.buaa.edu.cn (B. Li), huaijp@buaa.edu.cn (J. Huai).

http://dx.doi.org/10.1016/j.ins.2016.08.008
0020-0255/

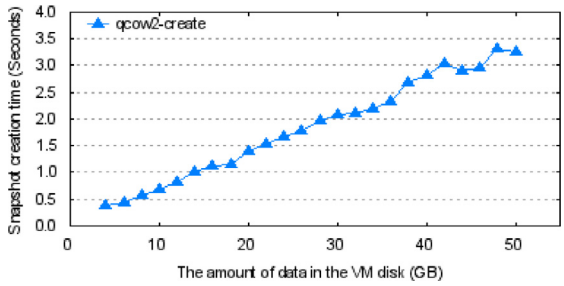


Fig. 1. The disk snapshot creation time of qcow2.

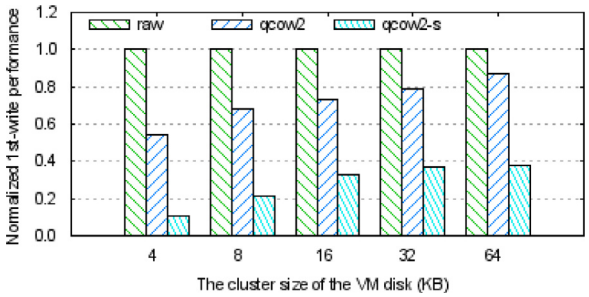


Fig. 2. The disk I/O performance loss of qcow2.

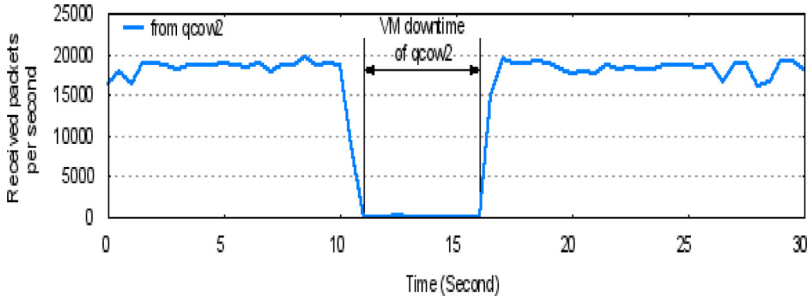


Fig. 3. The downtime during the VM live snapshot with qcow2 format.

Supported by the QEMU processor emulator used by several hypervisors including KVM, `qcow2` (QEMU *copy-on-write* version 2) is a widely adopted VM image format allowing the disk snapshot and the VM state snapshot. This format adopts *copy-on-write* (COW) for the disk snapshot, and a simple *stop-and-copy* method for the state snapshot, as well as a two-level tree index tree to store its metadata. However, this structure brings extra overheads to the snapshot performance and the I/O performance whilst data clusters of the VM are accessed. (1) The disk snapshot creation time grows dramatically with the increase of the VM disk size (Fig. 1). (2) The write performance loss is about 15 $\sim$ 45% of the `raw` format (another plain format of VM disk) in different cluster sizes. As a result of this inherent drawback of the COW-based snapshot, the 1st-write performance loss of `qcow2` after the snapshot creation reaches up to 60% $\sim$ 90% (Fig. 2). (3) The VM suffers from a long downtime during the live snapshot (Fig. 3) because of the *stop-and-copy* state snapshot of `qcow2`. (4) The sophisticated metadata structure of qcow2 designed to allocate the disk space on demand overlaps with sparse file, the similar function provided by host file systems, such as ext2/ext3/ext4, NTFS, reiserFS, etc. Therefore, the performance of VM snapshot creation and management is significantly affected, as well as the normal I/O performance of the VM disk.

In this paper, we design a bitmap-based light-weight index method to boost the operation performance of both the VM disk snapshot and VM disk I/O.

To boost the state snapshot creation, many techniques have been proposed, such as memory image compression [21], duplicated data elimination and unused memory elimination. Nevertheless, memory image compression has been found ineffective for checkpoint images [25]. Moreover, duplicated data elimination [18], tracks all disk I/O operations to maintain a map between memory pages and the identical blocks on the external storage. When a snapshot is captured, such duplicated pages are excluded. However, the I/O tracking burdens the VM I/O operations, and the time required to restore a VM from an optimized checkpoint image may increase remarkably, because it has to read the disk image discontinuously to restore those memory pages excluded before. Furthermore, unused memory elimination is not easily applicable to fully-virtualized guests

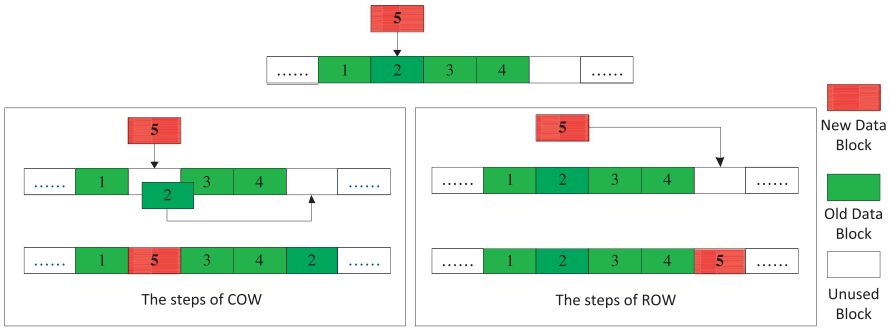


Fig. 4. The comparison between the COW and ROW method.

because the virtual machine monitor (VMM) has no prior knowledge of the memory organization of a VM. An alternative approach to obtain the guest memory information is *Virtual machine introspection* (*VMI*). In this paper, we detect the free pages of a running VM by virtue of the *VMI* technique, and skip saving these pages in snapshots.

In our prior work `iROW` [12,15], we introduced an efficient live snapshot system for VM disk. In this paper, we extend `iROW` with the additional VM state snapshot support, and further improve the disk snapshot design of `iROW`. `iROW` here refers to the extended work except for special statements, and its main contributions of `iROW` are as follows.

- A bitmap-based light-weight index structure, designed in `iROW` to replace the commonly used two-level index table structure in qcow2, reduces the amount of metadata, and improves the performance of both the VM disk snapshot and the VM disk I/O.
- `iROW` separates the VM state data from the VM disk data, enhances the performance of VM state data saving and loading operations, and reduces the VM downtime during the VM live snapshot using an *index-free* method for the VM state data.
- `iROW` identifies and skips saving free memory pages in the guest OS using the *VMI* technique when capturing a snapshot, which reduces the VM state snapshot creation time and the disk space.
- In the following, the results are presented regarding a series of experiments to compare `iROW` with `qcow2`. For the disk snapshot, the time incurred during the `iROW`'s disk snapshot creation, rollback and deletion is 17x, 35x, and 47x faster than that of `qcow2` respectively, when the VM disk size is 50GB and cluster size is 64 KB. Moreover, the superiority of `iROW` is more evident with the increase of the VM disk size. Further, `iROW`'s state snapshots creation time reduces to 60% or less than that of `qcow2` through the data storage mechanism and the *free page detection* (FPD) method, which depends on the workloads. For the live snapshot, the VM downtime of `iROW` can be reduced to 45% of that of `qcow2` in general. When the VM disk cluster size is 64 KB, `iROW`'s I/O performance loss is 10% less than that of `qcow2`; the performance of `iROW`'s first write operation after snapshot is approximately 250% of `qcow2`'s. At the same time, the space usage of `iROW` is similar to that of `qcow2` with the support by the sparse files.

The remainder of this paper is organized as follows. Section 2 briefly describes the related work. Section 3 details the design and implementation of `iROW`. Section 4 presents `iROW`'s performance evaluation results and analysis. Section 5 concludes this paper along with our future work.

2. Related work

In this section, we analyze two commonly used snapshot methods, and then introduce some related systems to the VM snapshot.

2.1. Snapshot methods

The Copy-on-Write (COW) snapshot and the Redirect-on-Write (ROW) snapshot are the two most commonly used snapshot methods [9,23].

As shown in Fig. 4, in the first write to a data block for the COW method, the original data is copied to another empty block, and then the new data is written into the block where the original data exists. The data copying operation brings an extra I/O operation, and thus the I/O performance in the first write operation decreases, which is the major drawback of the COW method.

In contrast, in the first write to a data block for the ROW method, the original data in the block is reserved and the write operation is redirected to another unused block. Thus, this method avoids the extra I/O operation of the COW method. However, as the new data is written to other blocks, the modified data leads to data fragmentation since the data after the snapshot is discontinuous to that of the unchanged data. The fragmentation problem may become intensified, and may result in the normal I/O performance degradation with the system running over a long time.

2.2. VM disk snapshot

Parallax [17,22] is a virtual storage system designed for Xen [3], adopting a three-level radix tree, a high-cost structure, to map virtual block addresses to physical block addresses. A complete radix tree takes up more than 1GB storage space (1 first-level page, 512 second-level pages, and 262,144 third-level pages, where each page is 4 KB). Although the radix tree may be cached in memory, this strategy lacks scalability. Additional I/O operations are needed to read the radix tree from the disk, decreasing the I/O performance. In order to solve the cost-inefficiency of Paralax metadata, SNPdisk [24] replaces the radix tree with a sparse tree by merging the index nodes, which allows increased amounts of metadata to be cached in memory, and thus improves the I/O performance. However, SNPdisk utilizes the COW snapshot method, which inevitably requires extra I/O operations in the first write operation.

As a COW-based image format and a block device driver for QEMU, Fast Virtual Disk (FVD) [20] boosts the I/O performance, thus yielding benefits based on the `raw` disk block. In addition, FVD supports the internal virtual disk snapshot that stores all snapshots in a single image. It also adopts *copy-on-read* and a prefetching technique to fetch blocks from backing image to improve the VM mobility. However, the COW method brings extra overheads to FVD and many of its features are not evaluated.

As the most important method supported by QEMU, `qcow2` (QEMU *copy-on-write* version 2) [4], another VM disk image format, supports snapshot, compression, encryption and other functions. It maps virtual block addresses to VM disk image addresses through a two-level index table. Besides, it utilizes a *reference-count* table to record quantities of snapshots sharing the same data clusters. Owing to the large size of its metadata, `qcow2` can only cache a part of the index table, so it constantly needs to read the index tree from the VM disk image, which leads to the performance loss of the VM disk I/O and disk snapshot operation. `qcow2` employs the same format to save VM state data and VM disk data, leading to performance degradation of the VM state data saving and loading operations. In addition, `qcow2` incurs great performance loss in the first write operation after the snapshot.

Our prior work, `iROW` [12,15], is also a VM disk image format designed for QEMU. In the prior `iROW`, a bitmap-based light-weight index method is adopted to replace the existing two-level index table structure to reduce the query cost. Moreover, the prior `iROW` can not only avoid extra copy operation in the first write operation with the COW method, but also handle the file fragmentation problem caused by the ROW method after a long-term usage, by means of a combination of the ROW and Copy-on-Demand (COD) method. The experimental results have shown that the prior `iROW` demonstrates obvious performance advantages in the snapshot creation operation, as well as better I/O performance. However, in the prior `iROW`, the time of a snapshot deletion is longer than that of qcow2 due to the data merging and the file deletion operations. Besides, the disk read performance may be affected by long snapshot lists since every read request traverses the current state and its ancestors. In particular, the prior `iROW` is designed only for the disk snapshot, not including the state snapshot which affects the live snapshot performance significantly.

2.3. VM state snapshot

The state snapshot comprises the volatile memory, the states of the virtual CPUs and the states of all connected virtual devices of a VM. To boost the state snapshot creation, many techniques have been proposed, including memory image compression [21], duplicated data elimination, and unused memory elimination. Nevertheless, memory image compression has been found ineffective for check-point images [25].

To remove the duplicated data from the memory image, Park [18] has proposed a technique that transparently tracks I/O operations of the guest and maintains a list of memory pages whose contents are duplicated on non-volatile storage. In a snapshot, these pages are excluded from the snapshot image. With this technique, the size of snapshot image reduces significantly, and less time is needed to capture a snapshot. However, the I/O tracking burdens the system I/O operation. Besides, the time required to restore a VM from an optimized snapshot image may increase enormously, because it has to read the disk image discontinuously to restore those memory pages excluded before. For a fully-virtualized environment, this technique cannot detect the free pages.

The effectiveness of unused memory elimination depends on the amount of memory actually used by the VM. This technique cannot be easily applied to fully-virtualized guests since the VMM has no prior knowledge of the VM memory organization. *Virtual machine introspection (VMI)*, which converts memory byte values into semantically meaningful data structures, is an alternative approach to obtain the guest memory information. *VMI* was originally proposed to detect malware intrusion in guests. In this paper, we detect the free pages of a running VM by *VMI* technique, and skip saving these pages in a snapshot.

3. Design and implementation of iROW

In order to solve the problems in the existing VM snapshot systems, we design an efficient VM snapshot system, `iROW`, based on the `qemu-kvm` virtual block device driver framework.

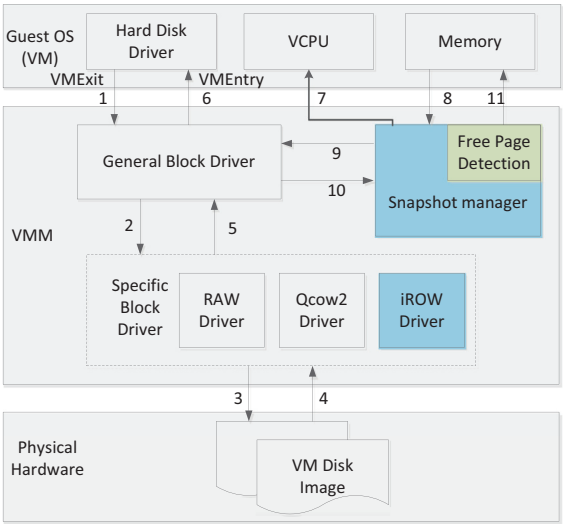


Fig. 5. The workflows of I/O and snapshot in `qemu-kvm`.

3.1. I/O and snapshot workflow in QEMU-KVM and the of *iROW* driver

The I/O and snapshot workflows in `qemu-kvm` are shown in Fig. 5, where the VM monitor (VMM) runs in *root mode*; the VM runs in *non-root mode* [11].

In the following, the I/O workflow is presented. When the VM receives an I/O request, a VMExit event is triggered. Then the I/O request is intercepted and sent by the VMM to the general virtual block device driver (Step 1) forwarding the I/O request to the specific block driver (Step 2) which operates on the VM disk image to obtain the return value or data (Step 3&4&5). Afterwards, the VMM begins to execute the VM code (Step 6). The simulation is transparent to the VM.

In the following, we present the snapshot workflow. When creating a live snapshot, the snapshot manager first pauses the execution of the VM by stopping virtual CPU (Step 7). Then it calls the specific driver through the general block driver to create the disk snapshot (Step 9&2&3) and obtains the disk snapshot result (Step 4&5&10). Afterwards, the snapshot manager begins to create the state snapshot. It collects and sends the VM state information to the general virtual block device driver (Step 8&9) which calls the specific driver to save the VM state to the VM disk image (Step 2&3). When the snapshot manager receives the state snapshot result (Step 4&5&10), the live snapshot is generated successfully. Then the snapshot manager resumes the stopped execution of the VM by restoring the virtual CPU (Step 7).

As shown in Fig. 7, based on the QEMU virtual block device driver framework, `iROW` is transparent to the general block driver. Thus, adding the `iROW` driver to the VMM exerts no impacts on other drivers. In addition, it is fully transparent to the OS, file system, and applications that run on the existing virtual block device driver. To reduce the size of the memory image, we add a *free page detection (FPD)* module to the snapshot manager to obtain a bitmap which indicates the free pages in the VM memory, using the *VMI* technique (Step 11), as is detailed in Section 3.7.

3.2. iROW VM disk image

As shown in Fig. 6, the `iROW` VM disk image consists of a *meta* file, several *snapshots* and its current state. The meta file contains metadata of the disk image. A *snapshot* includes a *bitmap file* (`btmp` file) and a *VM disk data file* (`irvd` file). The current state shares the same structure with a *snapshot*, so we use a *snapshot* to represent the current state.

The details of these files are introduced as follows.

- **The meta file** includes a *meta header* and the *snapshots information*. The *meta header* stores the basic information of a VM disk image, and the *snapshots information* sequentially stores the *name*, *id* and other related information of every snapshot. The iROW uses a 32-bit unsigned integer to store *snapshot id*, so it can support up to 2^{32}-1 snapshots (The current state of the VM disk occupies snapshot 0) which are sufficient for most cases.
- **The btmp file** is comprised of a *bitmap* and the *VM state data*.
  The bitmap indicates whether the clusters exist in the corresponding irvd file or not, where each bit represents a cluster in the VM disk image. Using a bitmap greatly reduces the metadata size of a VM disk image. Fig. 7 shows the bitmap size of iROW for various VM disk sizes and VM cluster sizes. We vary the VM cluster sizes from 512B to 2 MB, covering all possible cluster sizes in QEMU, and vary the VM disk sizes from 1GB to 1TB, covering the existing typical VM disk sizes. The results show that in most cases, the bitmap size of iROW is less than 1 MB (see the area surrounded by a bold red quadrangle), and thus can be completely cached in host memory.

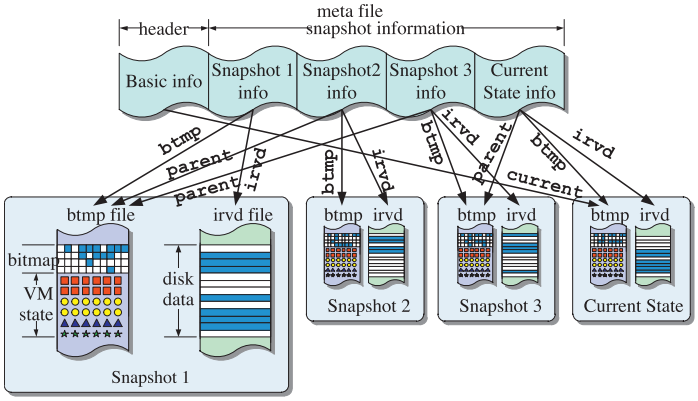


Fig. 6. The format of iROW image with three snapshots.

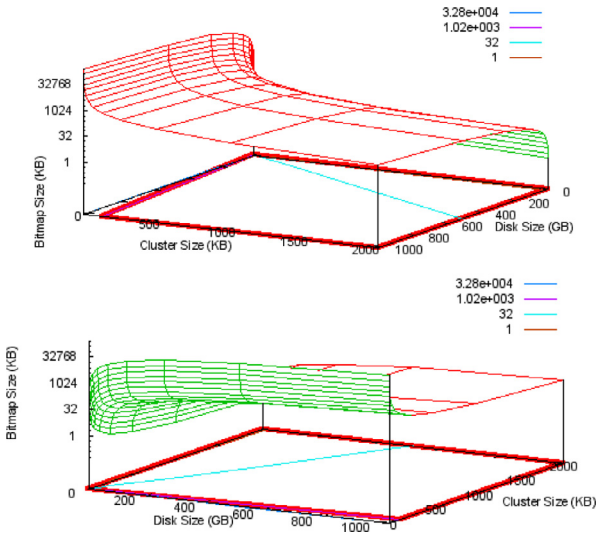


Fig. 7. The bitmap size distribution of iROW.

The *VM state data* includes the VM memory data and the virtual CPU registers data, etc. The *VM state data* is collected and saved during a live snapshot. In the offline mode, the btmp file does not store *VM state data*.

- **The irvd file** stores the actual data of a VM disk image. The basic unit of VM disk image storage is *cluster*. Users can specify the cluster size when creating an iROW disk image. iROW does not conduct complex costly address mapping as qcow2 does, because the virtual block address of the cluster is consistent with the VM disk image address in iROW. Besides, iROW gradually enlarges the VM disk image in accordance with the actual disk usage by virtue of the support of the host file system for sparse files.

3.3. *iROW disk I/O*

In the following, we detail the iROW disk I/O procedure. When receiving a read request from the VM, iROW first checks the *bitmap* to see whether the requested data exists in the current irvd file. If so, the data will be read from the current irvd file (Label ① in Fig. 8); otherwise, it will be read from the parent snapshots recursively (Label ② in Fig. 8). In this phase, two issues have to be addressed. (1) If the snapshot list is too long, the recursive data retrieving process may traverse too many bitmap tables, and hence the read operation performance may be affected. Therefore, we design a *merged bitmap table* method (detailed in Section 3.6) to overcome such an issue. (2) The I/O performance will be degraded due to the file fragmentation after a long-term usage of the ROW method. In order to eliminate such overheads, Copy-on-Demand (COD) is adopted in iROW. If COD is enabled, those data absent from the current irvd file will be copied from parent snapshots to the corresponding clusters in the current irvd file on the first read (Label ③ in Fig. 8). This method can improve the VM disk I/O performance in case of moderate data redundancy.

When receiving the write request from the VM, iROW conducts special handling of the data in the first and the last cluster, and the remaining data can be directly written to the corresponding clusters of the current state, shown as Label

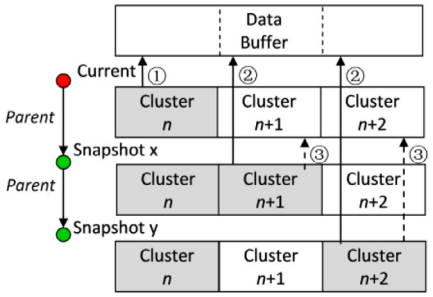


Fig. 8. The reading operation of iROW.

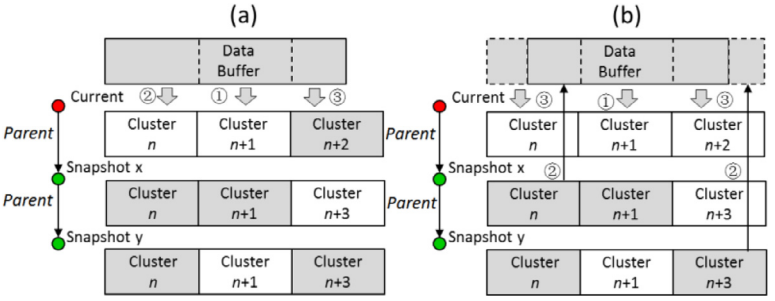


Fig. 9. The writing operation of iROW.

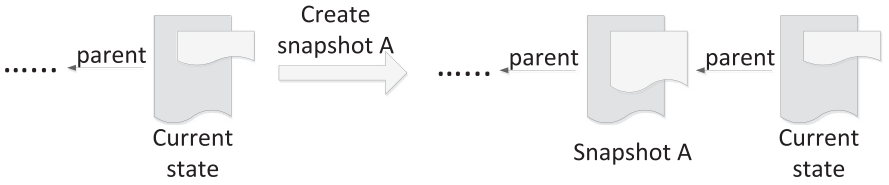


Fig. 10. The snapshot creation operation of iROW.

① in Fig. 9(a) and (b). If the data is aligned with the cluster (Label ② in Fig. 9(a)), or the target clusters are present (Label ③ in Fig. 9(a)), the data can also be directly written to the clusters. Otherwise, iROW reads these clusters from the parent snapshots recursively (Label ② in Fig. 9(b)), merges them with the data to be written, and then writes the data to the target clusters (Label ③ in Fig. 9(b)). For a real world application, if the cluster size is too large, each unaligned data write operation (Label ③ in Fig. 9(b)) needs to read a whole cluster (Label ② in Fig. 9(b)) before writing, which may cause write performance degradation. For better write performance, iROW allows users to specify the optimized cluster size.

3.4. *iROW snapshot operation*

All the iROW snapshot operations are implemented according to the format of the iROW disk image and the disk I/O workflow.

3.4.1. *Snapshot creation operation*

When creating a snapshot, iROW first performs the disk snapshot; during this process, iROW generates a btmp file and an irvd file as the new current state, and the *parent* pointer of the new current state points to the previous state as shown in Fig. 10. Then iROW performs the state snapshot by saving the VM state data to the old btmp file. Up to this phase, the previous state becomes a snapshot image, and the snapshot creation operation is completed. The disk snapshot creation only requires the file creation operation. Therefore, neither the VM disk size nor the cluster size affects the disk snapshot creation time of iROW. Note that the live snapshot creation time is mainly affected by the state snapshot, which we will discuss in Section 3.7.

3.4.2. *Snapshot rollback operation*

To roll back the current state to a target snapshot in an offline snapshot, we only change the parent pointer of the current state to the target snapshot, and clear the btmp file, as shown in Fig. 11. For a live snapshot, we need to load the state data from the target snapshot to the VM. Similar to the creation operation, the disk snapshot rollback time is independent of the disk size and the cluster size, while that of the state snapshot depends on the saved memory size.

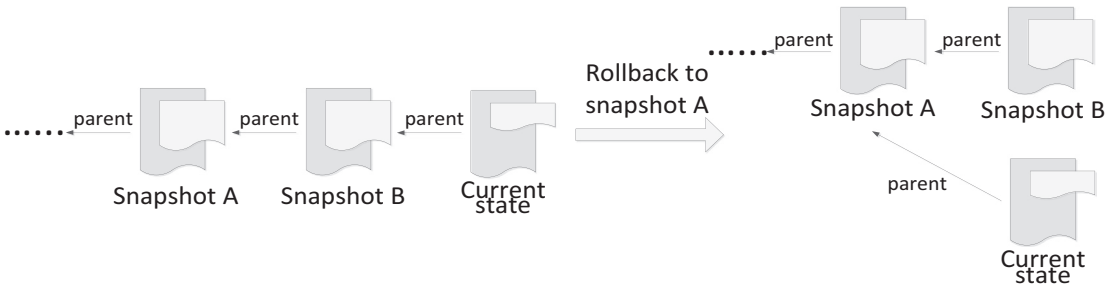


Fig. 11. The snapshot rollback operation of iROW.

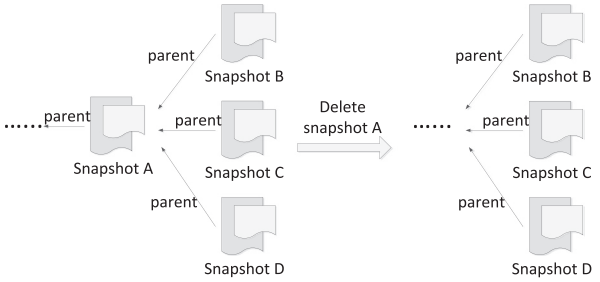


Fig. 12. The snapshot deletion operation without optimization.

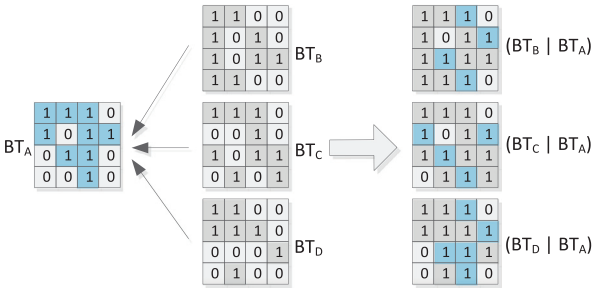


Fig. 13. The physical disk usage increase problem of the un-optimized snapshot deletion operation.

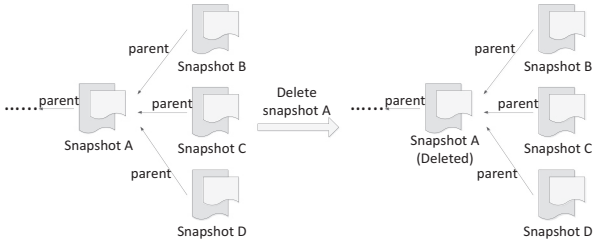


Fig. 14. The "*virtual deletion*" scheme of snapshot deletion operation.

3.4.3. Snapshot deletion operation

The iROW snapshot deletion is the most complex operation. Because of the dependency of snapshots, we have to merge the disk data from the snapshot to its children snapshots before deleting it in the prior iROW. This leads to a long deletion time and may result in physical disk over-usage if the snapshot has multiple children, as shown in Figs. 12 and 13. In Fig. 12, *A* is the snapshot to be deleted, and *B*, *C*, *D* are its children. In Fig. 13, BT_X denotes the bitmap table of the snapshot *X*, and this figure shows the bitmap tables of the snapshots correspondingly shown in Fig. 12. We must merge snapshot A with *B*, *C* and *D*, respectively, and then delete it. The blue clusters in Fig. 13 are the new allocated clusters, which occupy more physical disk space than *A* does. Therefore, iROW adopts a "*virtual deletion*" method to solve this problem: the deleted

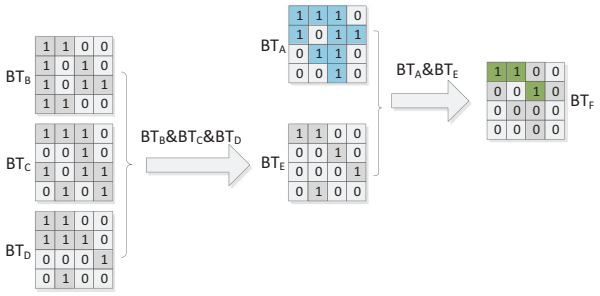


Fig. 15. Delete a snapshot in consideration of freeing the disk space as well as reducing the redundant data.

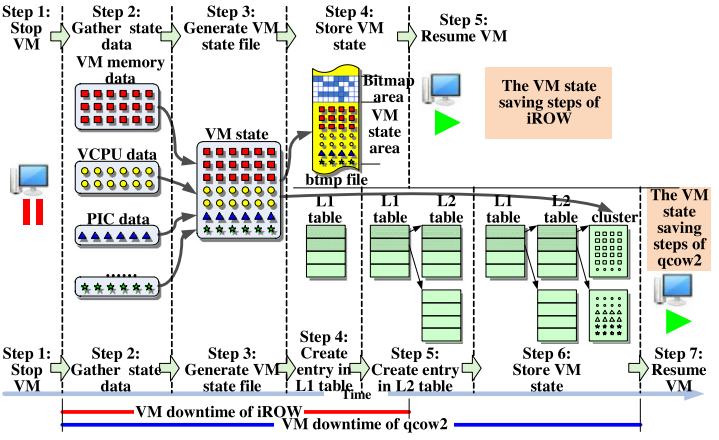


Fig. 16. Comparison of VM state saving procedures between `iROW` and `qcow2`.

snapshot is not actually deleted from the host file system, but marked as "*deleted*" in the metafile, as shown in Fig. 14. The "*virtual deletion* "method greatly reduces the VM snapshot deletion time, and avoids physical disk over-usage.

In addition, we implement a function to free the disk space insisted upon by the user, in consideration of freeing the disk space as well as reducing the redundant data. It will consume some deletion time; nevertheless, this amount of time is much less than the wasted time of duplicating the same data to every child. The method is shown in Fig. 15, where BT_A is the bitmap table of snapshot A to be deleted, and BT_B, BT_C and BT_D are the bitmap tables of snapshot B, C and D, respectively, which are the children of A. Firstly, we conduct bitwise-AND operation (BT_B & BT_C & BT_D) to obtain the result BT_E where the gray clusters are possibly redundant. Then we perform (BT_E & BT_A) and obtain the result BT_F where the green clusters can be actually deleted. Therefore, we can reduce the redundancy to a great extent and free the space simultaneously.

Next, we will discuss the long snapshot list issue in "virtual deletion". Even though COD is introduced to improve the read performance, the user may insist on using "real deletion" to cut down the snapshot list for saving the disk space. The process of replicating the data of the snapshots to be deleted to the children consumes an increased amount of disk space. Another possible solution is to merge the continuous snapshots into the last snapshot, thereby avoiding extra replicas. It is easy to combine `iROW` snapshots by merging the bitmaps in `btmp` files and replicating data in older `irvd` files into a single `irvd` file according to the bitmaps.

3.5. *iROW data storage mechanism*

The `iROW` separates the VM state data from the VM disk data, and we use an *index-free* method that directly stores the VM state data at the end of the `btmp` file. Therefore, any other metadata for the VM state data is not needed. Through the separation data storage and *index-free* method, the VM downtime of `iROW` is significantly reduced during the VM live snapshot.

Fig. 16 shows the comparison of the VM state saving procedure between `iROW` and `qcow2`. `iROW` directly stores the VM state data at the end of `btmp` file. `qcow2` stores the VM state data using the same method for storing the VM disk data. Therefore, it must create the corresponding metadata for the VM state data, before saving the VM state data to `qcow2` VM disk image. The unified data storage of `qcow2` greatly prolongs its VM downtime during the VM live snapshot.

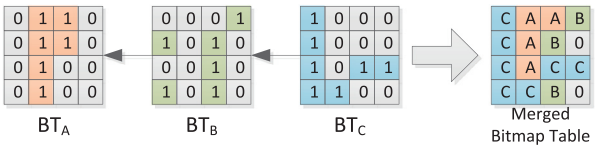


Fig. 17. Merging bitmap tables to a merged bitmap table.

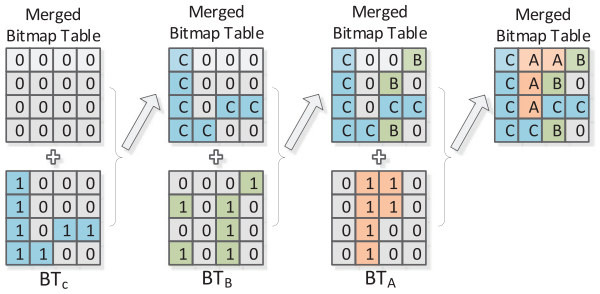


Fig. 18. The steps of merging bitmap tables to a merged bitmap table.

3.6. Merged bitmap table

In iROW, the I/O latency is very low, since all the bitmap tables used to locate the requested data can be cached in memory. However, if the VM snapshot list is too long, it will be costly to frequently and recursively read many bitmap tables. Hence, we use a merged bitmap table to avoid the multiple tables reading problem Fig. 17.

Fig. 18 illustrates the process of merging a bitmap table of the current snapshot with its ancestors to form a merged bitmap table, a three-node snapshot list as an example. All the values of merged bitmap tables are initialized as zero. Then, we merge the given bitmap table with the bitmap table of the latest snapshot (here is snapshot C or the current state). For every bit traversing through the original table of snapshot C and the merged table simultaneously, if the values are "1" and "0", we assign to the merged table a corresponding tag (here is "C") indicating the `irvd` file from which the data can be read. Lastly, we obtain a new merged bitmap table which can be merged with other snapshot bitmap tables (BT_B and BT_A) until we gain the final result. The merged bitmap table can improve the read performance, because data can be directly read from snapshot files using, instead of traversing the current state and its ancestors. Moreover, it can also save memory space used to cache individual bitmap tables.

3.7. The *iROW* VM state snapshot with free page detection

Since the size of the VM memory affects the state snapshot creation time significantly, we optimize the state snapshot method by means of *free page detection* (*FPD*). The key idea of iROW VM state snapshot is to avoid storing pages unnecessary for the system after restoring the VM. Such pages mainly comprise the free pages not used by the guest OS, so discarding them from the memory image does not hamper the correctness after the VM is restored.

It is not easy to obtain complete knowledge of the free memory information of the guest VM in a fully-virtualized environment. Therefore, we adopt *virtual machine introspection* (*VMI*) converting memory byte values into semantically meaningful data structures.

Linux kernel utilizes a page descriptor to maintain the state information of a page frame. All page descriptors are stored in the `mem_map` array starting with the address defined by a global macro `VMEMMAP_START`. Through this macro and the *page frame number* (`PFN`) of the memory page, we can obtain the page descriptor of each page. As a usage reference counter for the page, another field named `count` in the descriptor, helps identify the free pages in the guest memory. If it is non-negative, the page frame is assigned to one or more processes or kernel data. If it is `-1`, the corresponding page frame is free. Fig. 19 exhibits the principle of locating the page descriptor table in VMM.

When capturing a state snapshot, we first obtain a bitmap which indicates the free pages in the memory, using the aforementioned *VMI* technique. Then we preserve the full memory image except the free pages with the iROW block driver. Discarding the free pages can reduce the saved memory size, the VM downtime and disk space usage.

4. System evaluation

We have conducted a series of experiments to evaluate the performance of iROW. The experimental machine is a DELL Precision T1500 workstation with Intel Core i7-860 2.8 GHz CPU, 4GB DDR3 memory and SATA2 hard disk. The host and guest OS are both Debian 6.0. By default, all the experiment results are repeated at least ten times, and we obtain their

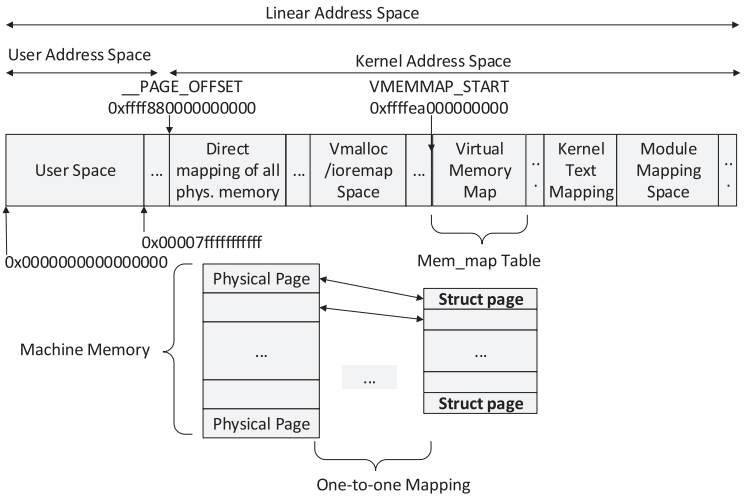


Fig. 19. The principle of locating the page descriptor table in VMM.

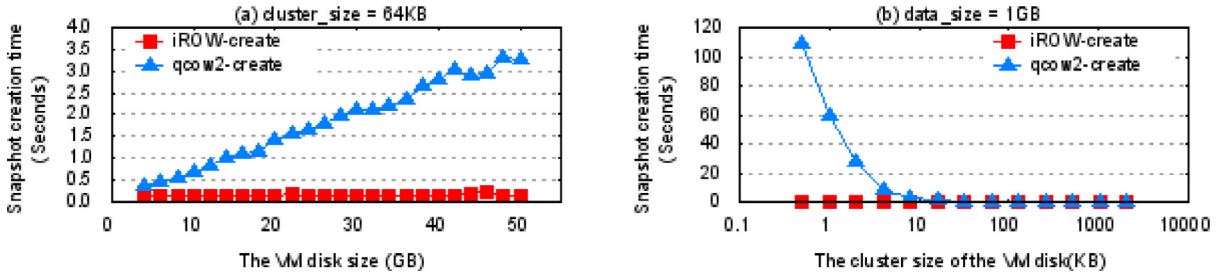


Fig. 20. The snapshot creation time of iROW and qcow2.

average value to improve accuracy. In our experiments, we focus on the performance of the disk snapshot, the state snapshot, disk I/O, and disk usage.

4.1. Disk snapshot performance

Experiment Setup: In order to examine the performance of disk snapshot creation, rollback and deletion, we create VMs with different disk sizes and cluster sizes for iROW and qcow2 images, respectively. Qcow2 and iROW are almost similar in performance with cluster_size = 64 KB and data_size = 1GB. Therefore, these can be considered as baseline settings. For every key snapshot operation, the experimental configurations are divided into two groups. We fix either *cluster_size* or *data_size* as one variable and vary the other to observe how each variable affects the performance. In the first group, we generate VM cases with *cluster_size* = 64 KB (default size used by qcow2 format) and vary the disk size from 2GB to 50GB by 2GB progressively. In the second group, we generate VM cases with disk *data_size* = 1GB and choose some typical cluster sizes from 0.5 KB to 10,000 KB. For every VM image, we first run a test program to write data to the disk till it is full, and create a snapshot. Then we randomly write data to the disk again, roll back to the previous snapshot, write data to the whole disk again, and delete the previous snapshot. The time of disk snapshot creation, rollback and deletion for iROW are denoted as $T_{iROW\text{-}c}$, $T_{iROW\text{-}r}$ and $T_{iROW\text{-}d}$, respectively, and as $T_{\text{qcow2-c}}$, $T_{\text{qcow2-r}}$ and $T_{\text{qcow2-d}}$ for qcow2, respectively.

The experimental results are shown in Figs. 20, 21 and 22, where comparisons are made between $T_{iROW\text{-}c}$ and $T_{qcow2\text{-}c}$, $T_{iROW\text{-}r}$ and $T_{qcow2\text{-}r}$, and $T_{iROW\text{-}d}$ and $T_{qcow2\text{-}d}$ respectively. We observe that the VM disk size and cluster size do not affect the time of the key disk snapshot operations of iROW, but they impacts the performance of qcow2. Furthermore, when the VM disk size is large or the cluster size is small, iROW obviously outperforms qcow2 in terms of VM disk snapshot operations.

When creating, rolling back or deleting a snapshot, iROW only updates a small amount of metadata, such as the *parent pointer* of the current state, the children number of the parent snapshot, and the deletion mark of the target snapshot, etc. Neither VM disk size nor VM cluster size would affect the amount of these metadata. Thus, the disk snapshot performance of iROW shows excellent stability in terms of various VM disk sizes and VM cluster sizes. However, when qcow2 performs disk snapshot operations, it needs to update the *reference count* of each cluster in the qcow2 VM disk. These operations should constantly read the level-2 index tree, reference count table and reference count block from the qcow2 VM image, and should write the new reference count value back to the qcow2 VM disk, which incurs significant performance penalties. Note that the number of clusters in VM disk is proportional to the VM disk size, and inversely proportional to the VM cluster

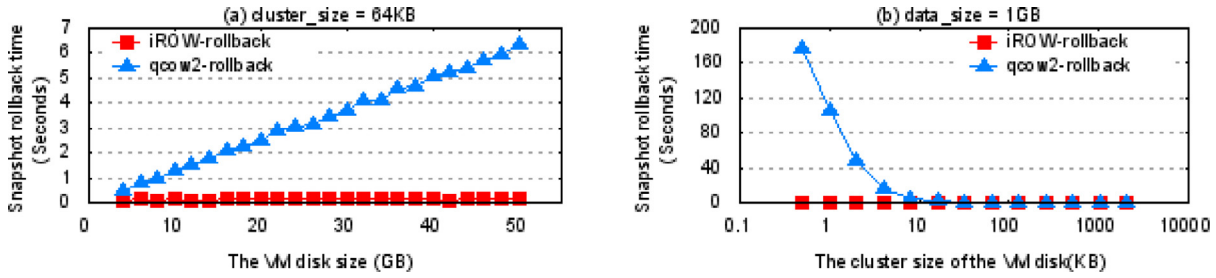


Fig. 21. The snapshot rollback time of iROW and qcow2.

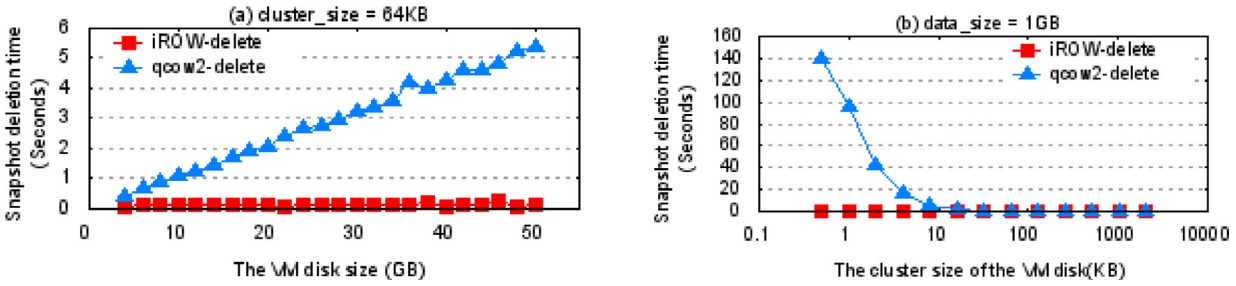


Fig. 22. The snapshot deletion time of iROW and qcow2.

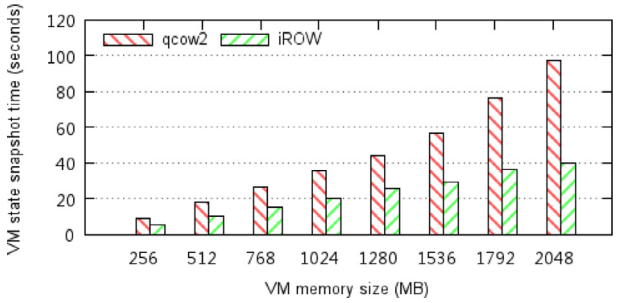


Fig. 23. The effect of the iROW data storage mechanism.

size. Therefore, more clusters result from the VM disk when the VM disk size grows larger, or the VM cluster size becomes smaller. As a result, the disk snapshot performance of qcow2 degrades with the increase in the VM disk size or the decrease in the VM cluster size.

4.2. State snapshot performance

In this section, we focus on the VM state snapshot mainly affecting the live snapshot creation time. We have conducted the following experiments demonstrating the advantages of our method.

Experiment Group 1: This experiment is to display the effect of the iROW data storage introduced in Section 3.5. First, we use the raw VM disk images as base images to create an iROW and a qcow2 VM disk with 64 KB cluster. Then we assign different memory sizes when booting up these VMs, capture a live snapshot, and record the VM state snapshot time, simultaneously.

Fig. 23 compares iROW with qcow2 in terms of VM state snapshot creation time when the VM state data increases. These results show that the VM state snapshot creation time of both iROW and qcow2 increases with increasing VM memory. However, iROW's snapshot time increases more slowly than that of qcow2; for instance, the snapshot time is 10.5 s for iROW when the size of memory is 512 MB, but it is 17.8 s for qcow2. The reason is that iROW does not assign any metadata to VM state data, but qcow2 does. The metadata of qcow2 assigned to the VM state incurs more performance penalties. Thus, as shown in Fig. 23, when VM memory increases from 256 MB to 2GB, the iROW's VM state snapshots time decreases from the equivalent of 60.3% of that of qcow2 to 40.9%.

Experiment Group 2: This experiment is to show the effect of the iROW state data snapshot designed in Section 3.7. We first choose the following workloads inside the VM.

- **Idle** workload means that the VM runs only the tasks of the OS after boot-up.

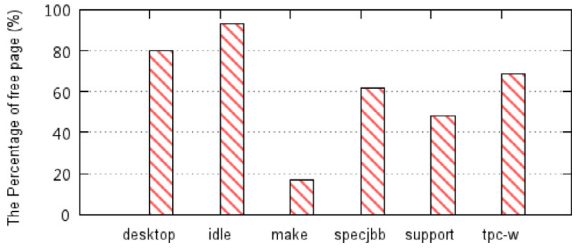


Fig. 24. The percentage of free pages in VM running different workloads.

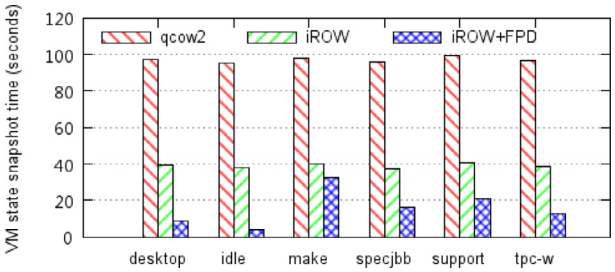


Fig. 25. The VM state snapshot creation time.

- **Desktop** represents a user session with several applications running at the same time. We run two Firefox web browsers each of which running four tabs, and OpenOffice Writer.
- **Kernel Compilation** represents a development workload involving memory and disk I/O operations. We compile the Linux 2.6.32 kernel along with all modules.
- **SPECjbb** is a SPEC benchmark that emulates a three-tier client/server system and is designed to evaluate the performance of server-side Java applications.
- **SPECweb2009_Support**, designed to evaluate the web server performance, is one workload of the Specweb2009.
- **TPC-W** is a transactional benchmark that simulates the activities of a business web server. We simulate a mix of R/W scenario through a shopping site workload with 600 concurrent accesses.

Firstly, we use the `qcow2` and `iROW` disk image in experiment group 1, and assign 2GB memory to the VM. Then, we run the VM under six different workloads mentioned above respectively, and record the percentage of free pages, and the VM state snapshot creation time simultaneously. When the snapshot completes, we also record the space usage of the state snapshot.

Fig. 24 shows the percentage of free pages in the VMs under the six different workloads. The results present the characteristics of the workloads running inside the test VMs. It can be observed that Idle and Desktop workloads have large amounts of free pages, and that the web service benchmarks, including SPECjbb, SPECweb2009_Support, TPC-W, also have as many as 48% $\sim$ 68% free pages. Kernel compilation is I/O intensive, and hence the amount of memory committed to the page cache is large, and that of free pages is relatively small.

Fig. 25 shows the state snapshot time of `qcow2`, `iROW`, and `iROW` with *FPD*. In this figure, "`qcow2`" denotes the VM state snapshot time using the `qcow2` format image with the original state snapshot method, "`iROW`" indicates the time of the `iROW` format image with the proposed data storage mechanism, and "`iROW+FPD`" refers to the time of the `iROW` with the proposed *free page detection* method. Similar to the results of the group 1, `iROW` outperforms `qcow2`. Besides, with the proposed *PFD* approach, the state snapshot creation time can be further reduced, and the results vary under different workloads. Moreover, idle workload depicts the most obvious decrease since it contains the most amounts of free pages. As expected, the more free pages a VM contains, the shorter the VM state snapshot creation time is.

Fig. 26 presents the results of the space usage occupied by the snapshot, where "base" means the space usage by the un-optimized mechanism. Different workloads result in different space usages, which is consistent with the amount of free pages.

Experiment Group 3: This experiment is to show the VM downtime during the live snapshot. We install Debian 6 in an `iROW` VM image, and use the `netperf` tool inside the VM to continuously send TCP packets to a host. In this experiment, the `netperf` serves as a networking performance measurement benchmark. At the host, we use a test program to record the number of packets received per second. At the 10th second, we create a live snapshot of the VM. The variation of packets received per second reflects the impact of the VM downtime on the VM live snapshot. For the next phase, we repeat the same experiment using a `qcow2` VM image.

As shown in Fig. 27, we observe that the data packets received per second for `iROW` and `qcow2` VMs decline rapidly at the 10th second, and that the transmission is even interrupted for a few seconds. Although both `iROW` and `qcow2` have VM

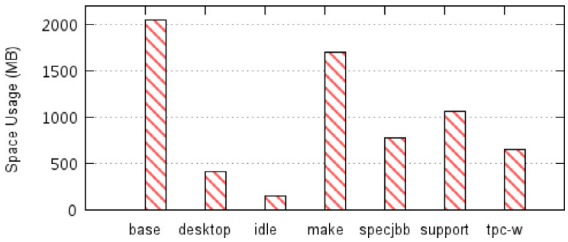


Fig. 26. The space usage of a state snapshot vary from different workloads running in the VM.

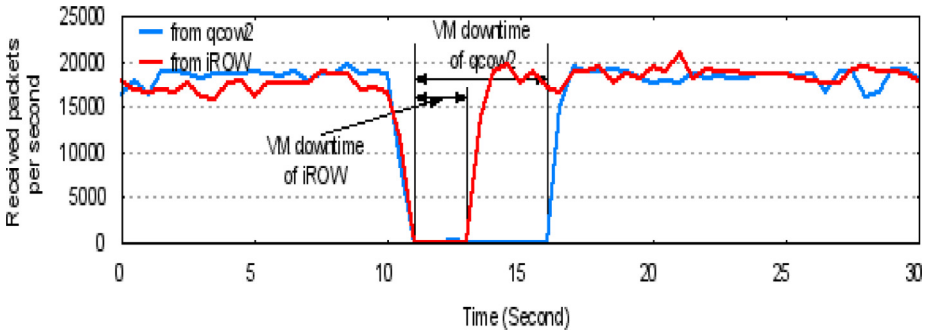


Fig. 27. The VM downtime of `iROW` and `qcow2` during live snapshot.

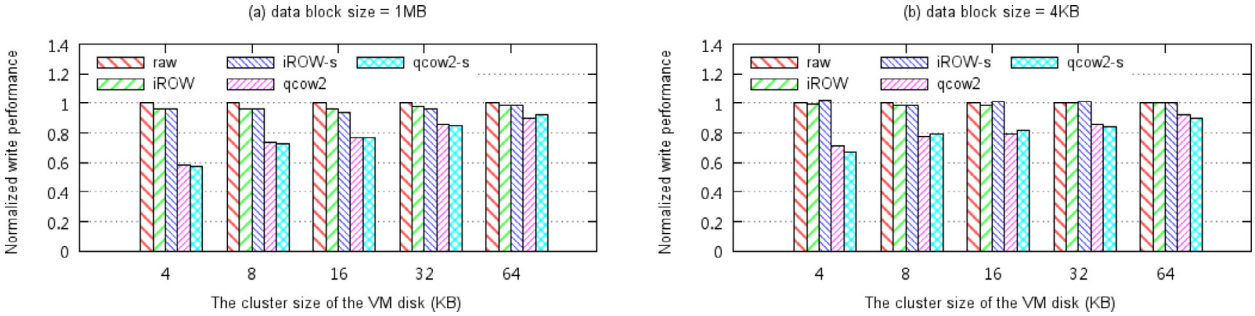


Fig. 28. The normalized write performance.

downtime, `qcow2` is much worse than `iROW`, and its downtime almost equals to 220% of that of `iROW`. This indicates that the live snapshot performance of `iROW` surpasses that of `qcow2`.

4.3. I/O performance

`qemu-io` is an I/O test and diagnostic tool of QEMU. It is mainly used to test the functionality of the virtual block driver. We add some I/O evaluation functions to the framework of `qemu-io`. These functions write random data with different block sizes to the VM disk image until the disk image storage reaches to the maximum. Then, these functions read data with different block sizes from the VM disk image, and record the time when the write and read I/O operation are performed simultaneously.

The results of these experiments are shown in Figs. 28–30. In these Figures, "`raw`" denotes the I/O performance of `raw` image which is of no snapshot function, "`iROW`" the I/O performance of `iROW` before snapshot, "`iROW-s`" the performance of `iROW` after snapshot, "`qcow2`" the I/O performance of `qcow2` before snapshot, and "`qcow2-s`" the performance of `qcow2` after snapshot.

Fig. 28 shows that the write performance of `iROW` is better than `qcow2`'s, and that the write performance of `iROW` is independent of the cluster size. When the data is written to the VM disk, `iROW` also writes the data to its original virtual offset in `irvd` file and updates the bitmap. However, `qcow2` needs to allocate the clusters offset in the VM disk image by querying the index table. In particular, the index table is too large to be fully cached in memory, meaning that during the write operations, `qcow2` needs extra I/O operations to read the index table from the VM disk image. When the VM cluster becomes smaller, the number of clusters increases and the index table grows larger. Therefore, the write performance of `qcow2` degrades when VM cluster size decreases.

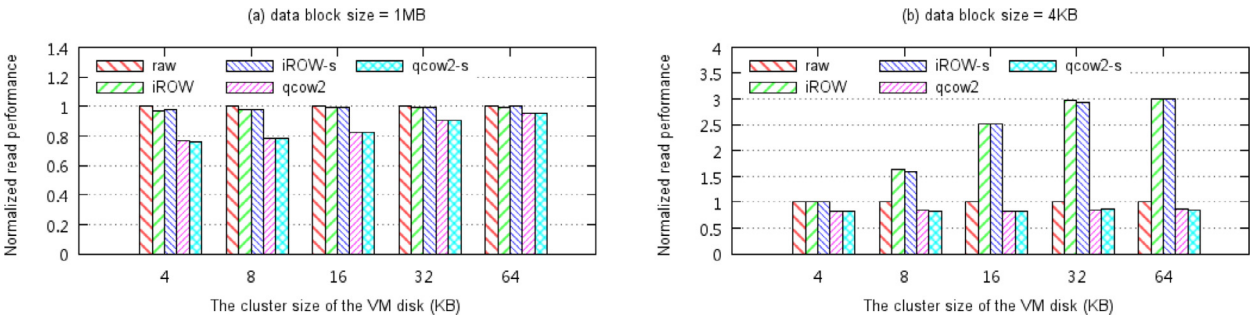


Fig. 29. The normalized read performance.

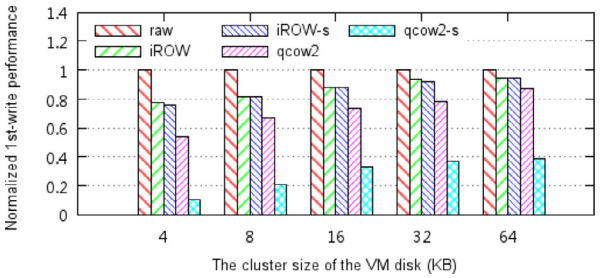


Fig. 30. the first write performance of `iROW` and `qcow2`.

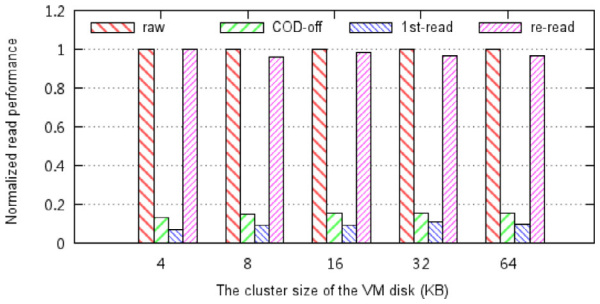


Fig. 31. The results of Copy-on-Demand mechanism.

Fig. 29 suggests that the read performance of `iROW` is superior to that of `qcow2` as well, for the same reason as above: `qcow2` requires extra I/O operations to read the index table, so that it can locate the clusters offset in VM disk image. The `iROW` caches the most recently read cluster, so in Fig. 29-(b), the performance of `iROW` is even better than that of `raw`, when the cluster size is larger than the data block size.

Fig. 30 compares different image formats in terms of the first write performance. The metadata of `iROW` is much simpler than that of `qcow2`, so that the first write performance loss of `iROW` is smaller than that of `qcow2`. In addition, `qcow2` has extra I/O operations on the first write after snapshot. Thus the first write performance of `iROW` far outperforms that of `qcow2` after snapshot.

In order to evaluate the effect of `iROW`'s Copy-on-Demand feature, we conduct the following experiments. First, we specifically design an unusual `iROW` VM disk. By alternately performing VM disk write operation and VM disk snapshot operation, we set all the $4n+1$, $4n+2$ and $4n+3$ clusters to be the captured snapshots 1, 2 and 3, respectively, and set all the $4n$ clusters to be in the current state. Then, we evaluate the read performance of this `iROW` VM disk with the Copy-on-Demand feature disabled. After that, we enable the Copy-on-Demand feature, and evaluate the first read performance and re-read performance of this `iROW` VM disk. Fig. 31 shows the results.

In Fig. 31, "`raw`" denotes the read performance of `raw` format, "`COD-off`" the read performance of `iROW` with Copy-on-Demand disabled, "1*st read*" the *first read* performance of `iROW` after Copy-on-demand is enabled, and "*re-read*" the re-read performance of `iROW` after Copy-on-Demand is enabled respectively. In this extreme case of fragmentation, when we disable Copy-on-Demand, the read performance of `iROW` becomes degraded. When we enable Copy-on-Demand on the first read operation, `iROW` needs to copy the clusters that are not in current state, so the *first read* performance of `iROW` further decreases. However, when the *first read* operation completes, the *re-read* operation performance of `iROW` increases above 95% of `raw`. Thus the effect of Copy-on-Demand feature is significant for I/O performance.

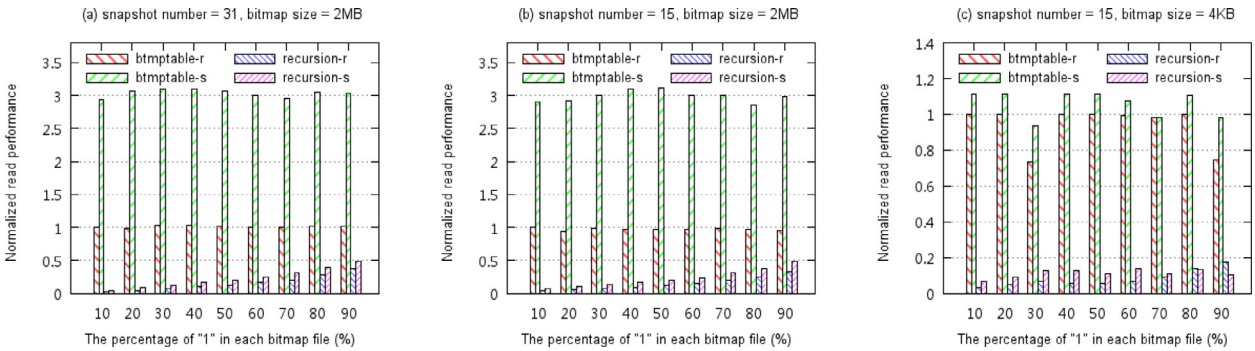


Fig. 32. The normalized read performance of the recursion method and the merged bitmap table method.

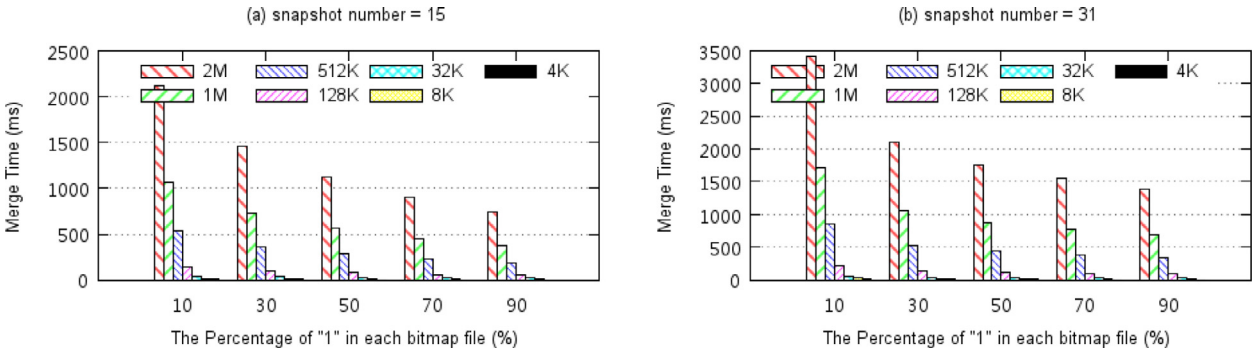


Fig. 33. The time of merged bitmap table generation.

We observe that "COD−off" and "1*st read*" underperform in Fig. 31. This is because we use an extremely fragmented artificial VM disk image. In this extremely fragmented situation, there exist no more than two clusters appearing continuously in the same snapshot. For the normal usage of VM, there is nearly no likelihood for the case of extreme fragmentation to occur. We use this case to demonstrate the effectiveness of the Copy-on-Demand feature of iROW. In real-world application scenarios, even when the Copy-on-Demand feature is disabled, iROW's read performance is much more preferable to that of "COD−off", as shown in Fig. 31.

4.4. Merged bitmap table

In order to verify the efficiency of the merged bitmap table, we conduct some simulations, in which we mainly measure the time consumed to find the target snapshot irvd file containing the reading data.

First, we generate 50 test cases randomly for every experiment by varying thee key parameters as follows.

- The size of data cluster: 4 KB or 2 MB.
- The percentage of "1" in each bitmap table: the value ranges from 10% to 90%, meaning that the real data is stored in the different data files.
- The length of the snapshot list: the value is 15 or 31.

Then we measure the read I/O performance in the following four cases. The first case is "*btmptable-r*" that denotes reading randomly in our merged bitmap table mode. The second case is "*btmptable-s*" that refers to reading sequentially in our merged bitmap table mode. The third case is "*recursion-r*" standing for reading randomly in the original recursion mode. The fourth case is "*recursion-s*" referring to reading sequentially in the original recursion mode. To compare the four group data, we normalize the final statistics results, as shown in Fig. 32. It suggests that the performance of disk read operation in the merged bitmap table mode is better than that in the un-optimized recursion read mode.

However, the generation of a merged bitmap table incurs additional time, which is evaluated in the further experiments. In these experiments, we set the number of snapshots to be 15 and 31, and the size of the bitmap table to be 4 KB and 2 MB, respectively. The percentage of "1" in each bitmap table ranges from 10% to 90%.

The experimental results are shown in Fig. 33. The merging time rises with the increase in the number of snapshots and the size of the bitmap table, but the cost is insignificant in comparison with the time saved in the merged bitmap table mode.

Note that a 2 MB bitmap table means that the size of the disk image is 1TB if the cluster size is 64 KB (the default cluster size), and that a 4 KB bitmap table corresponds to a 2GB disk image. With the increase in the number of snapshot list and

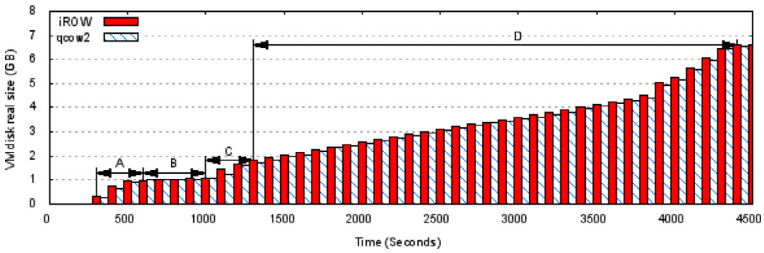


Fig. 34. The size of iROW and qcow2 image. Stage A is the OS installing process; stage B is the VM powering up and tools installing process; stage C is the Kernel source code downloading and decompressing process; stage D is the Kernel compiling process.

the size of a bitmap table, the amount of data required to be read will also increase, which causes the same obvious trend in the merge time. Moreover, the raising of percentage of "1" in each bitmap table implies no need to traverse all the previous bitmap tables, and the irvd file can be found in the near snapshots. Therefore, the merge time decreases.

4.5. Actual disk usage

iROW's disk space allocation function actually relies on the underlying host file system. iROW gradually benefits the VM disk image size with the actual disk usage because the host file system supports the sparse files. In order to prove that the disk space of iROW VM gradually increases with the actual increasing disk usage, we measure the disk usage of both iROW and qcow2. First, we create a 10GB VM disk image with the specified format (iROW or qcow2). Then we install Debian 6 OS with the necessary tool. Then, we download, decompress and compile the Linux Kernel. During such a process, we employ a script to record the real size of the VM disk image every 100 seconds. The results are demonstrated in Fig. 34.

Fig. 34 implies that the real size of iROW image gradually increases with the actual disk usage, and its real size is almost the same as that of qcow2 image.

5. Conclusions and future work

The existing VM snapshot methods suffer from issues such as long snapshot operation duration, long downtime for a live snapshot, and I/O performance degradation. To address such issues, we design an efficient VM snapshot system, iROW, based on the qemu-kvm virtual block device driver. (1) iROW uses bitmap to replace the high-cost two-level index table structure, which enhances the performance of both the VM disk snapshot and the VM I/O. (2) iROW separates the VM state data from the VM disk data, and does not assign any metadata to VM state. This method significantly improves the VM state snapshot performance. (3) iROW identifies free memory pages in the guest OS using *virtual machine introspection* and skips saving these pages when capturing a snapshot, thereby reducing the VM state snapshot creation time and the disk space usage. (4) With the advantages of the spare files in the underlying file system, the VM disk size of iROW gradually increases with the actual increasing disk usage.

The experimental results indicate that iROW possesses obvious performance advantages over qcow2. For the disk snapshot, when the VM disk size is 50GB and cluster size is 64 KB, the time of iROW's disk snapshot creation, rollback and deletion is 17x, 35x, and 47x faster than that of qcow2, respectively. The superiority is particularly evident with the increase in the VM disk size as well. With regard to the VM state snapshot, the iROW's state snapshots time decreases to 60% or less of that of qcow2 due to the data storage mechanism, and the time can be further reduced because of the free page detection, depending on the workloads. For instance, for the VM memory size of 2GB and web service benchmarks, such as SPECjbb, SPECweb2009_Support and TPC-W, the state snapshot creation time can be further reduced by 20%. For the whole live snapshot, the VM downtime of iROW can be reduced to 45% of that of qcow2 in general. When VM disk cluster size is 64 KB, iROW's I/O performance loss is 10% less than that of qcow2. Furthermore, the performance of iROW's first write operation after snapshot is close to 250% of qcow2's. The storage space of iROW is similar to that of qcow2 based on the support provided by the sparse files. We have integrated iROW into our iVIC cloud platform [14].

The new hardware trends appearing in clouds, such as none volatile memory (NVM), and Solid State Disk (SSD), bring new challenges to designing efficient snapshot systems. Therefore, it is exceptionally beneficial to leverage such hardware to further improve iROW performance.

Acknowledgments

This work is supported by China 863 Project (No. 2015AA01A202), NSFC Program (No. 61472022, 61421003), SKLSDE-2016ZX-11 and partly by the Beijing Advanced Innovation Center for Big Data and Brain Computing.

References

[1] M. Ali, S.U. Khan, A.V. Vasilakos, Security in cloud computing: opportunities and challenges, Inf. Sci. 305 (2015) 357–383. http://dx.doi.org/10.1016/j.ins.2015.01.025.

[2] A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr, Basic concepts and taxonomy of dependable and secure computing, IEEE Trans. Dependable Secure Comput 1 (1) (2004) 11–33.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of virtualization, in: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, SOSP '03, ACM, New York, NY, USA, 2003, pp. 164–177, doi:10.1145/945445.945462.

[4] F. Bellard, Qemu,, a fast and portable dynamic translator, in: USENIX Annual Technical Conference, FREENIX Track, 2005, pp. 41–46.

[5] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging {IT} platforms: vision, hype, and reality for delivering computing as the 5th utility, Future Gener. Comput. Syst. 25 (6) (2009) 599–616. http://dx.doi.org/10.1016/j.future.2008.12.001.

[6] L. Cui, Z. Hao, L. Li, H. Fei, Z. Ding, B. Li, P. Liu, Lightweight virtual machine checkpoint and rollback for long-running applications, in: International Conference on Algorithms and Architectures for Parallel Processing, Springer, 2015, pp. 577–596.

[7] L. Cui, J. Li, T. Wo, B. Li, R. Yang, Y. Cao, J. Huai, Hotrestore: a fast restore system for virtual machine cluster, in: 28th Large Installation System Administration Conference (LISA14), 2014, pp. 10–25.

[8] L. Cui, T. Wo, B. Li, J. Li, B. Shi, J. Huai, Pars: A page-aware replication system for efficiently storing virtual machine snapshots, in: Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE '15, ACM, New York, NY, USA, 2015, pp. 215–228, doi:10.1145/2731186.2731190.

[9] N. Garimella, Understanding and exploiting snapshot technology for data protection, http://www.ibm.com/developerworks/tivoli/library/t-snaptsm1/index.html, accessed: 08.10.11 (2006).

[10] R. Ghosh, F. Longo, F. Frattini, S. Russo, K.S. Trivedi, Scalable analytics for iaas cloud availability, IEEE Trans. Cloud Comput. 2 (1) (2014) 57–70, doi:10.1109/TCC.2014.2310737.

[11] Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2, Intel Corporation, 2011.

[12] iROW patch for qemu, http://patchwork.ozlabs.org/patch/215912/, (accessed: 17.07.16) (2013).

[13] K. Kourai, S. Chiba, Fast software rejuvenation of virtual machine monitors, IEEE Trans. Dependable Secure Comput. 8 (6) (2011) 839–851, doi:10.1109/TDSC.2010.20.

[14] J. Li, B. Li, T. Wo, C. Hu, J. Huai, L. Liu, K. Lam, CyberGuarder: a virtualization security assurance architecture for green cloud computing, Future Gener. Comput. Syst. 28 (2) (2012) 379–390.

[15] J. Li, H. Liu, L. Cui, B. Li, T. Wo, iROW: An efficient live snapshot system for virtual machine disk, in: Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on, 2012, pp. 376–383, doi:10.1109/ICPADS.2012.59.

[16] X. Lu, H. Wang, J. Wang, J. Xu, D. Li, Internet-based virtual computing environment: beyond the data center as a computer, Future Gener. Comput. Syst. 29 (1) (2013) 309–322 including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures, doi: http://dx.doi.org/10.1016/j.future.2011.08.005.

[17] D.T. Meyer, G. Aggarwal, B. Cully, G. Lefebvre, M.J. Feeley, N.C. Hutchinson, A. Warfield, Parallax: virtual disks for virtual machines, in: Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems, Eurosys '08, 2008, ACM, New York, NY, USA, 2008, pp. 41–54, doi:10.1145/1352592.1352598.

[18] E. Park, B. Egger, J. Lee, Fast and space-efficient virtual machine checkpointing, in: Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE '11, ACM, New York, NY, USA, 2011, pp. 75–86, doi:10.1145/1952682.1952694.

[19] Z. Shan, X. Wang, T.-C. Chiueh, Malware clearance for secure commitment of os-level virtual machines, IEEE Trans. Dependable Secure Comput. 10 (2) (2013) 70–83.

[20] C. Tang, Fvd: a high-performance virtual machine image format for cloud, in: USENIX Annual Technical Conference, 2011, p. 2.

[21] Virtualbox, https://www.virtualbox.org/, (accessed: 17.07.16).

[22] A. Warfield, R. Ross, K. Fraser, C. Limpach, S. Hand, HotOS, 2005.

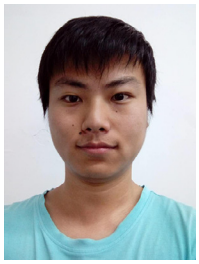
[23] W. Xiao, Q. Yang, J. Ren, C. Xie, H. Li, Design and analysis of block-level snapshots for data protection and recovery, IEEE Trans. Comput. 58 (12) (2009) 1615–1625.

[24] L. Yu, C. Weng, M. Li, Y. Luo, Snpdisk: an efficient para-virtualization snapshot mechanism for virtual disks in private clouds, IEEE Netw. 25 (4) (2011) 20–26.

[25] I. Zhang, A. Garthwaite, Y. Baskakov, K.C. Barr, Fast restore of checkpointed memory using working set estimation, in: Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE '11, ACM, New York, NY, USA, 2011, pp. 87–98, doi:10.1145/1952682.1952695.

[26] Z. Zhang, Z. Li, K. Wu, D. Li, H. Li, Y. Peng, X. Lu, Vmthunder: fast provisioning of large-scale virtual machine clusters, IEEE Trans. Parallel Distrib. Syst. 25 (12) (2014) 3328–3338.

Jianxin Li is a professor at the School of Computer Science and Engineering, Beihang University, and a member of IEEE and ACM. He received the Ph.D. degree in Jan. 2008. He was a visiting scholar at machine learning department of CMU in 2015, and a visiting researchers of MSRA in 2011. His current research interests include virtualization and cloud computing, data analysis and processing.



Yangyang Zhang is currently a Ph.D. student at the School of Computer Science and Engineering, Beihang University, China. His research interests include virtualization and distributed systems.

Jingsheng Zheng received the M.S. degree in CS from Beihang University, China in 2015. His research interests include virtualization and cloud computing.

Hanqing Liu received the M.S. degree in CS from Beihang University, China in 2013. His research interests include virtualization and cloud computing.

Bo Li is an assistant professor at the School of Computer Science and Engineering, Beihang University. He received the Ph.D. degree in Jan. 2012. He was a visiting scholar in computer science department of University of Edinburgh in 2014. His current research interests include virtualization, system reliability and data mining etc.

Jinpeng Huai is a professor at the School of Computer Science and Engineering, Beihang University. He received his Ph.D. degree in computer science from Beihang University, China, in 1993. Prof. Huai is an academician of Chinese Academy of Sciences and the vice honorary chairman of China Computer Federation (CCF). His research interests include big data computing, distributed systems, virtual computing, service-oriented computing, trustworthiness and security.