Rethink the storage of virtual machine images in clouds

Xiaolin Xu, Hai Jin, Song Wu*, Yihong Wang

Services Computing Technology and System Lab, Huazhong University of Science and Technology, Wuhan, 430074, China
Cluster and Grid Computing Lab, Huazhong University of Science and Technology, Wuhan, 430074, China
School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China

HIGHLIGHTS
- We propose a three-tier zone based VM image storage model.
- Zone-based model balances requirements of performance, resources, and overhead.
- The zones and caches in Zone-based model have high scalability and availability.
- A friendly VM placement strategy is proposed to improve IO performance of VMs.
- We evaluate some popular models and conclude their advantages and disadvantages.

ABSTRACT
As one of the most prevalent cloud services, Infrastructure-as-a-Service provides virtual machines (VMs) with high flexibility. How to effectively manage a huge amount of VM images becomes a big challenge. On one hand, images affect VMs' disk IO performance significantly, which is essential to the quality of services, especially for those having intensive disk IO workloads. On the other hand, they consume many storage resources and cause much management cost, which is cared by cloud managers. Current ways to optimize images usually focus on either improving performance or decreasing image size, which unfortunately cannot satisfy the requirements of high IO performance, low storage consumption, and low management cost simultaneously. Typically, high IO performance requires images storing close to VMs, but this increases redundant data and consumes extra storage at the same time. Besides, a closer image means more data stored in local disks rather than a normal shared storage, which increases management cost as well. In this paper, we analyze these requirements and potential tradeoffs among them, and propose Zone-based model to well balance the requirements. We partition computing nodes into many zones, and construct a shared storage in each zone to cache hot data for high IO performance and low storage consumption. In addition, we improve the normal Copy-on-Write and cache mechanisms, providing new image types and cache functions to enhance the eventual effectiveness. The evaluations show that, our solution improves IO performance by more than 100% in general and even 10 times while adopting a friendly VM placement strategy, and gets close or less storage consumption and management cost than the other models at the same time.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

Depending on the virtualization technology, clouds convert physical resources to virtual resources billed under the “pay-as-you-go” form, which significantly improves resource utilization and reduces users’ costs. One of the basic and widely provided cloud services is Infrastructure-as-a-Service (IaaS) [1], which provides a customizable computing environment saved in a virtual machine (VM) image. Generally, a virtual machine runs on a physical machine, looking as a complete and independent operation environment, while an image acts as a virtual disk, storing the operating system and applications of a VM, just as the disk to a physical machine.

In IaaS clouds, most images are mounted as virtual disks to respond IO requests during VMs’ running, and are stored as ordinary files to persist complete computing environment during VMs’ shutting. With rapid growth of VMs in clouds, how to manage their images effectively becomes a big challenge. On one hand, storing and organizing of images affect VMs’ IO performance. In a
large-scale cloud, computing system and storage system are usually kept separate from each other. The visits to images may cause obvious network loads, which in turn lowers down VMs’ performance. Although the Copy-on-Write (CoW) technology [2] allows VMs visiting local images rather than remote ones for reducing global network loads, it unfortunately increases the chances of hot spots appearing in base images at the same time [3]. On the other hand, management operations probably affect system status. Essentially, managing images is just managing files in the cloud. VM creation and migration equal to the distribution and migration of image files. These management operations occupy tremendous valuable network bandwidth. A larger image would cost more transferring time, and produce more management overhead. Meanwhile, it is unacceptable that lots of redundant image files waste many storage resources.

Therefore, from both users’ and managers’ perspectives, there are obvious requirements to manage images well, which include high IO performance, low management overhead, and high storage utilization. Unfortunately, traditional image storage systems cannot well meet these requirements simultaneously. Almost all the optimizations aim at improving IO performance, such as [4,5], while only a few of them consider the rest requirements, such as [6,7]. The main reason is that there are hard tradeoffs between IO performance and the other two requirements. On one hand, high IO performance and low management overhead are usually hard to meet at the same time. Precisely, a general way of improving IO performance is to cache image data at local disks as much as possible. However, lowering management overhead requires saving data in a remote central storage as much as possible. On the other hand, high IO performance and high storage utilization have conflicts as well. Increasing storage utilization needs reducing redundant data, and a feasible way is to save images in the central storage. However, this approach leads to a high sharing degree, in which many remote requests are produced and thus declines IO performance.

In this paper, we analyze above requirements and tradeoffs, and propose an optimized storage model trying to satisfy the requirements. In detail, first, we describe the background of image storage and format. Second, we analyze the requirements of managing images and the potential tradeoffs among them, and discuss some traditional image storage models. At the end, we propose a new image storage model, which can provide high IO performance without much extra management cost and storage consumption. Roughly speaking, we divide all computing nodes to many zones and construct a shared storage for each zone to cache hot data of images. Zone storage achieves the goal of sharing data among VMs. Not only does it avoid the bottleneck caused by global sharing, it also avoids the high storage consumption caused by pure local caches. The evaluations show that, our solution improves IO performance more obvious than the other optimized solutions with similar management cost and storage consumption. It improves IO performance by more than 100% in general and even 10 times while adopting the VM placement strategy that is friendly to our model. In addition, we also verify the scalability and availability of our model in multiple scenarios.

Our contributions are summarized as below:
- We analyze the requirements of managing VM images, and discuss why it is hard to meet them simultaneously in practice.
- We classify some popular image storage models in industry and academia, and discuss their advantages and disadvantages according to their characteristics and the requirements.
- We propose a zone-based image storage model with considering all requirements and present some key metrics to evaluate models deeply.

The rest of this paper is organized as follows: Section 2 describes the background and related work. We analyze requirements and tradeoffs and discuss traditional models in Section 3, propose our model in Section 4, and evaluate it in Section 5. Section 6 concludes this paper.

2. Background and related work

In this section, we first briefly discuss the background of VM images, including cloud architecture, image format, and image storage, and then we discuss the related works as well.

2.1. Typical cloud architecture for storing images

Generally, a typical cloud data center consists of management nodes, computing nodes, sharing storage, multiple switches and routers, and some other types of support equipment. Computing nodes provide core resources, such as CPUs, memory, and disks. Depending on these resources, we can create and run multiple VMs in the cloud. It is possible to store VM images in local disks of computing nodes. However, just a small amount of VMs with large-size images could exhaust the local disk of one node. So, a global sharing storage is provided to store images centrally. These computing nodes and the storage are connected through switches (Ethernet or Fiber). According to different image storage models, VMs have various ways to locate images. If the image of a VM is stored locally, the VM can locate data in the local disk without network transmission. However, if the image is stored in the global storage or in the disks of the other nodes, requests should be redirected and transferred through one or multiple switches.

2.2. Image format and image storage

Image format. Image format is a critical factor that affects images’ management. The most original approach to create a VM is to generate a blank image file in a specified format as a virtual block device, based on which the operating system is installed as same as that on a physical machine. However, this process is unacceptable since it always spends tens of minutes. Fortunately, we can accelerate it by using advanced image formats with new features. This is why the approaches based on the Copy-on-Write mechanism [2] are widely adopted.

Different image formats have distinctive features and drawbacks. Take RAW and QCOW2 as examples, the former one has high compatibility and flexibility, while the latter one has optimized size and CoW ability. In our practice, the images in RAW format always act as base images to support the incremental images in QCOW2 format. In fact, allocating a large-size RAW image to each VM is unbearable in normal clouds. Leveraging the native support of CoW feature in QCOW2 format, multiple incremental images can share the same base image, which reduces redundant data greatly. However, too much sharing may transfer a large amount of requests to base images, occurring hot spots and resource contentions at the same time.

Image storage. The architecture of image storage has significant impacts on images’ organization. Traditionally, there are two choices to store images, local solution and remote solution. The remote solution is usually adopted by large-scale data centers. These data centers are typically composed by multiple computing nodes, and all the nodes are connected to a global storage. In such clouds, VMs run in computing nodes while their images are saved in the shared storage. During VMs’ running, images are remotely mounted as virtual disks through network. Depending on the shared storage, the remote solution supports many advanced features, such as live migration [8,9] and snapshotting. However, the separate placement of VMs and images requires high dependable network hardware to maintain the performance of virtual disks, which unfortunately is incompatible with commercial equipment in general clouds.
In fact, a general cloud is constructed by many commercial servers loosely connected through Ethernet links. In such environment, VMs’ performance may be significantly harmed due to network contentions or hardware failures if we still store all images in a remote storage. Here, the local solution acts in an alternative way, distributing images to the local disks of computing nodes. Thus, VMs can mount their images from the local disks directly, avoiding the network thrashing and thus improving IO performance. However, it brings some additional problems. Such as, redundant image copies occupy many storage resources, management operations consume expensive network resources, and the failures of local disks may completely destroy VMs’ computing environment.

The above problems motivate researchers exploring new image storage models and image formats for better performance and scalability. Nonetheless, there is not a universal approach yet to guarantee all the requirements including IO performance, storage utilization, and management overhead simultaneously.

3. Requirements and tradeoffs

3.1. Requirements

From both users’ and managers’ perspectives, managing images should conform to the following requirements.

**High IO performance**: IO performance is mainly cared by cloud users. There are two aspects of it, throughput and latency. Generally, it is better to maintain throughput higher and latency lower. Higher throughput means VMs have ability to read and write more data during a certain time interval, while lower latency means VMs have ability to obtain the required data in a shorter time. In practice, many factors affect VMs’ IO performance. If images are saved in local disks, disks’ IO speed limits the performance. Advanced storage equipment, such as SSD, can improve throughput and latency greatly in an obvious way. However, if images are stored remotely, network bandwidth and switch ability become new limitations. In either case, network resource contents are difficult to avoid, which degrade the performance as well.

**High storage utilization**: Cloud managers care about whether their physical resources are used in a high efficient way. Since images are stored to support VMs’ running, they occupy many storage resources if multiple VMs run concurrently. Obviously, cloud managers want to explore the potential ability of storage equipment as much as possible, trying to decrease unnecessary using of the storage spaces on computing nodes and central storage. One efficient way is to reduce image size. Although images are usually big and difficult to resize, fortunately, we can use many effective techniques, such as CoW and deduplication mechanisms [15], to eliminate redundant data and thus reduce the image size. If so, the cost would be decreased, and more VMs can be supported and run concurrently.

**Low image management overhead**: Management overhead is concerned by both managers and users, because a high management overhead may cost extra money and potentially affect VMs’ normal running. Although many types of management operations exist in clouds, we only focus on the operations related to images. Concretely, we consider the overhead of distributing and migrating images because they are tightly related to VMs’ creation and migration. Images’ distribution often appears at creating new VMs. Its overhead increases dramatically if multiple VMs are created simultaneously [16]. When many images are copied and transferred, they occupy more expensive network resources than expected. Images’ migration is much similar to distribution, but it only happens when VMs migrate from one node to another. Once a migration happens, the related image should be transferred from the current node to another node. Its overhead would increase if these two nodes are far from each other or if the image is large. Essentially, the overhead is determined by the size and the location of images.

3.2. Tradeoffs

Ideally, we expect the above requirements being optimized for the best at the same time. However, it is difficult to meet this goal in practice. There are two hard tradeoffs among them. The first appears between IO performance and storage utilization, and the second appears between IO performance and management overhead.

**Tradeoff between IO performance and storage utilization** results from the conflict between sharing and not sharing of images. There is always a hard tradeoff between these two requirements [17]. Generally, if more images are shared, less redundant data would exist, and fewer storage resources would be consumed. However, this may cause lots of concurrent visits to the shared images, increase the chance of contentions, and harm the IO performance of VMs. In the opposite, if fewer images are shared, each VM can get its own complete image. Thus, fewer contentions would happen and higher IO performance would be gotten. However, this may cause lots of similar image copies, and wastes a mass of storage resources. Therefore, these two requirements are usually hard to meet at the same time.
a feasible method to reduce the size. With the CoW mechanism, two VMs share the same public image $M$ in the shared storage, and put their private images $I_i$ and $I_j$ separately in local storages. In this way, the cost of image distribution would be approximately zero because it only needs to create empty private images locally, and the cost of image migration just equals the cost of transferring private image $I_i$ or $I_j$. Fig. 3(b) shows an ideal way of shortening the delay. In this case, the private images are stored in the shared storage as well. Thus, the cost of image migration is eliminated because all the nodes can access both public and private images from anywhere. We can see that these approaches require sharing images as much as possible.

By comparing the requirements of IO performance and management cost, it is easy to find that the former one needs a low sharing rate to reduce the global network loads and contentions, while the latter one expects a high sharing rate to reduce the transmission cost. Obviously, it is hard to fulfill these two requirements at the same time.

3.3. Traditional models

There are many existing solutions for storing images in clouds, some of which are optimizations towards the above requirements. Here we discuss some popular models in industry and academia. For simplicity, we classify them into two groups depending on model tiers, the one-tier group and the two-tier group.

Obviously, there is only one tier of storage to save images for the models in the one-tier group. According to the location to store images, we can get two specific models in this group, Central model and Distributed model. The Central model is the simplest and most ordinary model, in which, all images are stored in a central shared storage. When creating a VM, this model needs to copy a work image from a template at first, and then boots the VM based on the image copy. Since all images are stored centrally, bottlenecks are easy to generate. All requests access to the central storage, so the network contentions increase and the performance decreases. It is not difficult to implement the Central model in practice. A typical one is to store all images in a centralized NFS storage server.

The Distributed model works opposite to the Central model. It distributes images at the local storage of computing nodes rather than a central storage. When creating a VM, this model first copies a work image from some template chunks saved in multiple computing nodes, and then boots the VM from the image copy. Some solutions [18–20] distribute image chunks in a parallel way to improve the performance. Owing to many image copies, this model sacrifices storage utilization and management cost for IO performance, which is an extreme tradeoff.

In the two-tier group, images are usually cut into two parts and stored in different storages separately. As same as the one-tier group, there are two typical models as well, which are Global–Local model and Local–Local model. The Global–Local model is an improvement of the Central model. It has a two-tier structure. The first tier stores base images in a central storage and the second tier stores incremental images. Most implementations of this model put the second tier at the local disks of computing nodes, such as [11]. When creating a VM, this model only creates an empty incremental image linked to a base image, and then boots the VM from the empty image locally. However, its IO performance is still not ideal because it also brings the same problem as the Central model.

The Local–Local model is widely adopted in recent works. It also has two tiers including base images and incremental images just like the Global–Local model. However, its two tiers are both stored in computing nodes. Typically, base images are stored in a distributed file system constructed on computing nodes, while incremental images are stored in local disks of computing nodes directly. A VM can be created by generating an empty image locally.

We illustrate three cases to describe the conflict. Fig. 2 shows three possible mappings between VMs and images. In the one–one case, each VM has its own image with no sharing. In particular, $V_1$ and $V_2$ have the same image type, while $V_1$ and $V_2$ have a different type. All images are under the size of $S$. Since each VM gets a complete image, the total consumed storage is $3S$. In the many–one case, $V_1$ and $V_2$ share the same base image $A$. Thus, there are only two different images, and the total consumed storage is $2S$. In the many–many case, we extract the same contents of $A$ and $B$ (assumed as half of $S$) to form a new image $C$, which is shared by all the three VMs. Thus, there are three incomplete images, which consume a total of $1.5S$ storage spaces. We can notice that more sharing causes less storage consumption.

Meanwhile, along with the increase of sharing, each image receives more and more requests. In the one–one case, each image receives the load produced by only one VM (assumed as $L$). In the many–one case, two images support three VMs, and each image should receive the load generated by 1.5 VMs on average ($1.5L$). In the many–many case, three incomplete images support three VMs, and each complete image should receive the load of $2L$. Obviously, the load on each image grows along with the sharing increases, which makes it harder to guarantee the performance.

It is easy to find that, less storage consumption requires more images shared, but meanwhile increases remote requests. When two VMs on separate nodes attempt to access the same base image, at least one of them has to access remotely. As a result, the network loads increase, causing high latency and low throughput.

Tradeoff between IO Performance and Management Overhead also results from the conflict between sharing and not sharing. In fact, there are two ways to improve IO performance, eliminating remote access and using the nearest node if remote accesses are unavoidable. By doing so, the load on the global network is decreased sharply, the latency is reduced greatly, and the throughput is improved as well. Notice that, these methods require reducing shared image files because all nodes accessing the shared storage concurrently would increase the network load significantly, which harms the latency and throughput.

For management overhead, we can reduce the cost by reducing the data size or shortening the transferring delay. Fig. 3(a) shows Fig. 3(a) an abstract architecture of cloud datacenters.

![Fig. 1. An abstract architecture of cloud datacenters.](image)

(a) One–one.  
(b) Many–one.  
(c) Many–many.

Fig. 2. Three typical mappings between VMs and images.
Fig. 3. Sketches of image distribution and migration.

Table 1
Characteristics of traditional image storage models.

<table>
<thead>
<tr>
<th>Models</th>
<th>Tier</th>
<th>IO</th>
<th>Storage</th>
<th>Cost</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Central</td>
<td>1</td>
<td>−−</td>
<td>−−</td>
<td>++</td>
<td>NFS</td>
</tr>
<tr>
<td>Distributed</td>
<td>1</td>
<td>+</td>
<td>−</td>
<td>+</td>
<td>Raw in local</td>
</tr>
<tr>
<td>Local–Local</td>
<td>2</td>
<td>++</td>
<td>+</td>
<td>−</td>
<td>VDN [6]/Blobseer [21]</td>
</tr>
<tr>
<td>Zone-based</td>
<td>3</td>
<td>++</td>
<td>++</td>
<td>+</td>
<td>Our proposal</td>
</tr>
</tbody>
</table>

In the table, the plus sign means better results than the minus.

and linking it to a distributed base image. Since there is no remote request towards the central storage, this model has better IO performance than the above models. In practice, [4] is a typical example of this model.

In fact, by mixing above models, we can easily implement new hybrid approaches. Take [6] as an example, not only it uses a central storage to save template images, but it also shares some of image file chunks through a P2P network constructed by computing nodes. These approaches aggregate advantages of the above models, which motivate us to adopt this way as well.

A brief discussion about traditional models: Table 1 summarizes all the image storage models mentioned above. Besides the four metrics, we give the tiers and examples for these models as well. To sum up, the models built on a centralized storage usually have lower management overhead but worse IO performance than the others. In particular, the Central model consumes the most storage because each VM has a complete image copy, and the Global–Local model consumes less due to many VMs sharing the same base images. The models built on a distributed storage normally have better IO performance but bigger management overhead. They fully use the storage spaces of computing nodes but ignore the central storage. Same as the Central model, the Distributed model has many redundant copies. Thus, they still consume many storage resources. In the opposite, the Local–Local model eliminates redundant data and distributes unique data on computing nodes. Thus, it consumes less storage than other models. In fact, it is considerable to integrate the advantages of these models, and that is exactly what we take into account in our model. We will give a detailed evaluation in Section 5.

4. Our proposal

In this section, we propose a new model called Zone-based model, to store, distribute, and cache VM images. Our target is to improve IO performance of VMs, and guarantee low resource consumption and management cost. We first take a brief overview of the model, then explain tradeoffs inside of it, and finally discuss how to implement and deploy it in practice.

4.1. Model architecture

Zone-based model adopts multi-tier architecture to store images. All images are stored in a three-tier storage, which consists of global storage, zone storage, and local storage, shown in Fig. 4. All the computing nodes in a cloud are able to access the global storage, which provides core shared storage resources. Its function is to store a huge amount of base images, acting as an image repository. Unlike EC2 [22,23] and some other opensource cloud platforms, the global storage here is only used to store read-only images. Thus, VMs cannot boot from these images directly. To guarantee VMs’ normal running, we store writable work images in the local storage. Finally, the most important improvement of Zone-based model is that we insert a new storage tier between the above two tiers, which is the zone storage.

The zone storage is a regional storage serving a set of specific nodes. It limits the sharing scope unlike the global storage. Thus, only the permitted nodes are able to access this storage. The model partitions all the computing nodes into many zones. The boundaries of zones are dynamically adjusted according to VMs’ distribution and network status, aiming to increase images’ similarity and caches’ utilization in each zone. The zone storage provides smaller but faster and more stable storage capacity. Such features make it much suitable for caching hot data in similar images to accelerate VMs’ booting and running processes. The network links connecting the nodes in each zone multiplex the global network links.

In Zone-based model, images are grouped in three types with different purposes, which are base images, cache images, and work images. Only the proper cooperation of these images can assist VMs’ normal running. The base image stores basic running environment under read-only mode, and the work image stores running data incrementally under read–write mode. These two
image types look like those in the Distributed model mentioned before. However, the difference is that our model adds another image type, the cache image, which locates between the base image and the work image, caching hot data to decrease resource contentions and network latency.

During a VM’s running, its images in three storage tiers form an image chain. When the VM reads a data block, it first finds it in the work image, if the data do not exist, it then finds it in the cache image, and finally in its base image. In the opposite, when the VM writes a data block, it writes in the work image directly. Because many VMs share the same base image, the cache mechanism gradually stores the most frequently accessed data into cache images. We store cache images into the zone storage. They not only cache hot data of base images, but also hot data of work images. It is worth noting that, the data in cache images are completely the copies of the data in base images and work images, which means VMs can still run even if we remove all zone storages and cache images. This feature makes Zone-based model well compatible. Therefore, we can easily migrate images between traditional clouds and Zone-based clouds with only a few modifications.

4.2. Tradeoffs in Zone-based model

We consider the tradeoffs mentioned in Section 3 during our design. Briefly, the three-tier architecture balances IO performance, storage utilization, and management overhead. First, comparing with the one-tier model, we improve IO performance and storage utilization while increasing a little management overhead. In our model, the whole image of a VM is separated to a base image, a work image, and caches, in which, the base image is shared to reduce storage consumption, and the work image is locally stored to improve IO performance.

Second, comparing with the two-tier model, we use caches to improve IO performance and use a zone storage for sharing caches to reduce potential storage consumption and management overhead. Caches save hot data in both base images and work images, which bring better performance than the two-tier model. Although caches increase redundant data, they can be accessed by all the nodes in a zone rather than by only a single node. Thus, the storage space occupied by caches is limited. Due to the cache size is small, the increase of management overhead can be ignored. In fact, since caches are just the data copies of base images and work images, we can safely distribute and migrate images without them, by which, the overhead is eliminated.

There are three simple cases in Fig. 5 for the one-tier model, two-tier model, and Zone-based model respectively. Four VMs are running in each case. In Fig. 5(a), all VMs have their own complete images, and all requests are sent to images directly. In Fig. 5(b), each VM has its own work image and all VMs share the same base image. All write requests are handled in work images while read requests are handled in work images or base images. In Fig. 5(c), every two VMs locate in the same zone and share caches. We can get following observations. First, our model generates the fewest global network loads because all write requests hit in work images and most read requests hit in caches. Thus, we can guarantee IO performance best in clouds’ complicated network environment. Second, our model occupies much fewer storage resources than the one-tier case because we eliminate three redundant copies of the base image. Third, our model causes a bit more overhead than other cases when managing images. In fact, the one-tier case does not need transferring any data when distributing and migrating VMs, and the two-tier model only needs to transfer work images, while ours may need to transfer caches as well. In Section 5, we will evaluate these models deeply to verify the above observations.

4.3. Prototype implementation

In this section, we briefly introduce the prototype implementation of our model, including the implementations of storage, image, zone, and a recommended VM placement strategy.

4.3.1. Storage

Ideally, it is feasible to leverage the local storages (disks) of computing nodes to construct the zone storage. In the prototype, we construct the three tiers of storage completely with existing storage equipment in the cloud without extra hardware. For the global storage, we adopt a part of the global storage server in the cloud as they already connect to all computing nodes. If there is no such shared storage server, we can alternatively choose one of the computing nodes those have large disks to act as the global storage.

The local storage is easy to implement just depending on local disks of computing nodes. We separate the local disk of each computing node into two parts. One of them acts as the private part to store work images, and the other acts as the public part waiting to be aggregated. We then integrate the public parts of local disks to form a distributed storage through a distributed file system. The distributed file system takes in charge of managing cache images and routing IO requests into corresponding zones. This form has the advantages of sufficiently consuming local disks (usually in a low utilization), concurrently accelerating data transmission, and flexibly adding or removing nodes.

4.3.2. Image

As mentioned before, we extend the image of a VM to three types. In practice, we use RAW format for base images, QCOW2 format for work images, and data chunks for cache images. RAW is the most ordinary format in the virtualization environment. It is very stable and suitable for saving the operating system that has a large size of data. QCOW2 is a different image format depending on the CoW technology. We adopt it for work images because it is designed to store data incrementally with a much smaller size than RAW. Thus, it is suitable for local disks that have limited storage resources.

For cache images, there is no mature format to directly adopt. We just split them to many data chunks (8 kB for each) and store them in the zone storage. In other words, in a zone’s distributed file system, we can only see numerous small files rather than a complete large image file as in the global storage. The distributed
file system maintains these small files and transfers them through corresponding APIs. Different from the other two image types, caches are not bound to a unique VM but shared by multiple VMs, which depends on the function of the distributed file system as well. Typically, cache images are blank during zones’ initialization.

Depending on the feature of CoR in the newest version of QCOW2 format, data blocks are pulled from base images into cache images when hit misses appear on both of work images and cache images. In order to ensure the uniqueness of data blocks in the zone storage, we split caches into small chunks and check their hash codes to guarantee storing each of them only once in a single zone. Meanwhile, we integrate some cache replacement policies, including LFU and LRU, to limit the cache size.

4.3.3. Zone

In our initial design, zones’ ranges are constant, which cannot be changed after the initial setup of the cloud, similar to [4]. However, we find a potential drawback in practice. Owing to the dynamic nature and management policies of the cloud, the VMs with similar images are frequently migrated across different zones, leading to low utilization of caches and degrading availability of zone storage. Thus, it is better to enable adjusting zones’ ranges and redistributing hot data across zones.

Considering the network topology in Fig. 5(b), the performance of a zone reaches best only when two conditions are fulfilled. First, nodes that contain the VMs with high similar images are partitioned into the same zone. Otherwise, the similarity of VMs in the zone is probably too low to maintain a high cache-hit ratio. Second, nodes with close topological distance are partitioned into the same zone. Otherwise, if the nodes in the same zone are physically far away from each other, the cost will be huge while VMs visiting caches.

In the prototype, we adopt a simple but effective method to partition zones. The partition process is that, we first calculate the image similarity of all VMs, and then determine the dominant image type of each node. For each node, we find another node with the similar dominant image type in the scope of a predefined topological distance, and aggregate them as a temporary zone if they do not belong to any zones yet. We repeat this process until there is no proper node in the scope any more, and finally we mark this temporary zone as a fixed zone. The rest zones can be obtained from the rest nodes by repeating the above process as well. Meanwhile, we update zone partition by continuously monitoring the dominant image type of each node, because the type may be changed by VMs’ migration or termination. If the dominant image type of node changes, we move the node from the current zone to another zone, in which the nodes have similar dominant image types to it.

Furthermore, update of zones will cause update of zone storage and redistribution of cache data. We use a simple method to accelerate this process and reduce costs. When one node is removed from a zone, part of zone storage contributed by this node would be shifted from the current zone to the target zone. At the same time, unique cache data only stored in this part of storage needs to be transferred to the other nodes in the current zone, and the cache data related to the VMs only on this node would be shifted to the target zone. In this way, we guarantee that all caches are available after the update, and all VMs can access them inside zones.

4.3.4. Zone-friendly placement

Traditionally, image management is rarely related to VMs’ placement. The only request is that VMs should be placed to the computing nodes where they can access images. However, a specific VM placement strategy to Zone-based model can improve application performance greatly. In fact, we can see from [13] that, VMs in the same application usually have very similar images, which is just the necessary condition to improve the cache-hit ratio in our work. Meanwhile, [24] indicates that it would be better to put VMs in the same application together since they communicate with each other frequently. Thus, we have an opportunity to improve performance by adopting this placement strategy, i.e. scheduling VMs in the same application or similar applications into the same zone, which we call as Zone-friendly placement strategy. By adopting this strategy, we can improve VMs’ IO performance and reduce global network contentions obviously.

5. Evaluation

In this section, we first conclude some key metrics to evaluate image storage models, and then use them to evaluate our work by simulation.

5.1. Assumptions and evaluation metrics

Before analyzing in following sections, we make some assumptions about cloud architecture here for simplicity. In Section 2, we abstract a typical cloud architecture in Fig. 1. It has a tree-based network topology, and we assume each link has a bandwidth of B. Meanwhile, we assume there are N same VMs running on computing nodes, and they send IO requests at a large rate simultaneously. There, G(P, E) represents the network topology of the cloud, in which, P indicates the set of computing nodes and switches, and E indicates the links connecting the nodes in P. For arbitrary couple nodes in P, such as P1 and P2, we define Eij as the network path between them.

During VMs’ running, we assume their requests can be fairly handled by physical equipment. For example, VMs on the same node can fairly use the local disk. Thus, they can both read and write data on the equal rate. For another example, the requests of different VMs can fairly occupy the bandwidth of network links. Moreover, since the image is possible to be split into separate places, we assume the requests are uniformly located at these places no matter where the image is stored. Based on the CoW mechanism, the image of a VM consists of a public part and a private part. If we assume m VMs sharing a base image, and then each of them occupies the storage space of 1/m times the image size plus the private part. It is worth noting that we make assumptions just for evaluation simplicity, so our work is not restricted by these assumptions in practice.

According to the requirements mentioned before, we consider four metrics to evaluate different image storage models here. They get most attentions from industry and academia, including IO latency, IO throughput, relative storage consumption, and operation costs, presented in Table 2.

IO latency: We consider the one-way latency of requests traveling from VMs to corresponding images, which may pass single or multiple switches. Any switches with long queues between VMs and their images may increase the total latency of VMs. Due to that the time of one packet going through Ethernet mainly spends on queuing, we mainly focus on queuing delay and ignore transmission delay and propagation delay. According to the queue theory, the latency of a switch w is related to its handling ability a and package arriving speed λ, i.e. L(w) = f(a, λ). Thus, from a VM sending a request to the request hitting target image, the total latency can be described as

\[ L = \sum_{w \text{ in path}} L(w). \quad (1) \]

In which, path is the link path between a VM and its image. For end users, when L goes down, IO performance goes up. We can see the latency is larger if the path is longer or if the congestion is more significant. Thus, we need to reduce the path length and resource contentions.
IO throughput: Throughput is the other metric to express IO performance. It represents the amount of IO requests towards one VM handled by the cloud during a certain time interval. According to the assumption that all switches adopt fair scheduling policy, the bandwidth of links would be fairly shared by VMs. Thus, the throughput of a VM is either the sending rate of requests if the link is idle, or the actual bandwidth assigned to the VM if the link is congested. Then, the throughput of a VM on the link \( E_t \) can be described as \( T(E_t) = \min(R_t, B_t/\text{img} \times \gamma) \) in which \( R_t \) is the requests’ sending rate, \( B_t \) is the link bandwidth, and \( \text{img} \times \gamma \) is the amount of VMs sharing the link. In practice, the throughput of a VM is limited by the throughput of the bottleneck link in the path between the VM and its image. Therefore, the eventual throughput of a VM can be calculated by

\[
T = \min_{E_t \in \text{path}} T(E_t).
\] (2)

For end users, when \( T \) goes up, IO performance goes up as well. Throughput gets more chances to be low if the path is longer and if the sharing is more. Thus, we need to reduce the path length and sharing scope to guarantee high throughput.

Relative storage consumption (RSC): Relative storage consumption refers to the ratio of the actual storage consumption by images from a baseline. The baseline is the case of the original virtualized environment where each VM has a complete image copying from the template when it is created. Thus, the baseline occupies more storage than many methods theoretically. Current methods try to increase storage utilization by eliminating duplicate data in images, such as [13,15]. In this way, one VM may have less than one complete image. Thus, they can reduce storage consumption to much less than the baseline. Here, we use RSC to compare storage consumption degrees of different methods. Let \( \gamma \) represent RSC, we have

\[
\gamma = \frac{\sum_{i=1}^{N} \text{space}(V_i)}{\sum_{i=1}^{N} \text{space} \text{ (template)}}.
\] (3)

In which, \( N \) indicates the amount of VMs, \( \text{space}(V_i) \) represents the actual image size of a VM, and \( \text{space} \text{ (template)} \) represents the template image size in the baseline case. A smaller \( \gamma \) means more free storage resources. If one method gets a smaller \( \gamma \) than the other methods, it occupies fewer storage resources to support the same amount VMs.

Operation cost: As mentioned before, we mainly consider two types of management overhead here, migration cost and distribution cost. Essentially, all distribution and migration operations can be viewed as transferring images. An intuitive thought is that, an operation would cause more influence on normal applications and cost more money if it takes longer time and occupies more resources to finish the same job. Thus, we describe the cost of transferring an image \( \text{img} \) from \( P_i \) to \( P_j \) as

\[
\text{Cost}(P_i, P_j, \text{img}) = L(P_i, P_j) \times \text{space(img)}.
\] (4)

In which, \( L(P_i, P_j) \) is the network latency between the computing nodes \( P_i \) and \( P_j \), and \( \text{space(img)} \) is the network traffic caused by the operations. The cost grows up when the average latency between the nodes gets bigger as it costs more time to complete the operation. At the same time, the cost grows down when the image size gets smaller because there is less data to transfer.

5.2. Evaluation setup

In our experiments, we simulate a cloud data center built on the CloudSim toolkit to simulate virtual machines’ creation, placement, and scheduling processes. In addition, we write programs to simulate the data accessing process of different image storage models, and integrate them into the simulator. Our simulated cloud contains a three-tier network topology illustrated in Fig. 6. In this topology, the root switch uses 1 Gbps Ethernet to connect 4 aggregation switches; each aggregation switch uses 10 Gbps Ethernet to connect 8 access switches; and each access switch connects 5 computing nodes or storage nodes through 10 Gbps Ethernet. There are 250 computing nodes and 6 storage nodes in total. All VMs are distributed to computing nodes, and an image repository is distributed to storage nodes. In the repository, we make 20 template images with size of 8 GB, and every four template images form a series. Based on the conclusion of [13] and our practice, we set image similarity between 60% and 80% inside each series while between 5% and 30% among different series, and less than 10% between work images and template images. During evaluation, we control zones’ partition based on similarity threshold and partition boundary. For convenience of implementation, the similarity threshold is set between 0 and 1, and the partition boundary is set to one of three forms, which allow computing nodes under the same host, the same access switch, or the same aggregation switch to form a zone, respectively. Meanwhile, cache size ranges from 0.1 times to 4 times of the template image. Cache chunks (8 kB per chunk) are randomly distributed to the nodes in a zone.

Based on the above setup, we evaluate from three aspects. First, we compare different models horizontally. Second, we change some key parameters in Zone-based model to evaluate it vertically. Third, we change some external parameters to measure scalability and availability of our model.

5.3. Models comparison

We compare the key metrics of Zone-based model with other models at first, including IO latency,.IO throughput, RSC, and operation cost. We choose Global–Local (an abstraction of [11]) and Local–Local (an abstraction of [4]) models as the contrasts because they are more general and practical and have better performance than pure Central and Distributed models. Taking the fact that there is a cache tier in our model, we modify the normal Global–Local model by adding a cache tier too, forming a Global–Local (cache) model. It is an abstraction of [7] and improves

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO latency</td>
<td>One-way latency when VMs access images</td>
</tr>
<tr>
<td>IO throughput</td>
<td>Throughput when VMs access images</td>
</tr>
<tr>
<td>Relative storage consumption</td>
<td>Ratio of actual storage consumption from the baseline</td>
</tr>
<tr>
<td>Operation cost</td>
<td>Cost caused by image distribution and migration</td>
</tr>
</tbody>
</table>
Fig. 7. Evaluation results under different models. In each subfigure, the result of zone-based model acts as the baseline.

the Global–Local model. In each group of experiments for different models, we organize and place images according to characters of the model, send IO requests from VMs to images, and record metric values continuously. Meanwhile, we use the random placement strategy to randomly schedule VMs to computing nodes, and use the same image size and IO requests trace for each model.

Fig. 7 presents the evaluation results. In order to describe the comparison clearer, we normalize them based upon the result of Zone-based model. There are four phenomena observed from the results. First, in Fig. 7(a), our model lowers down IO latency by more than 50% and 40% comparing with Global–Local model and Local–Local model respectively. This indicates that the caches in our model play a role in reducing remote requests when VMs require the data of base images. Second, in Fig. 7(b), we can see that our model provides a remarkably higher throughput than Global–Local models, but little lower than Local–Local model. We consider the reason is that Local–Local model distributes images into many computing nodes and reduces hot network links as well. Thus, it improves throughput just like ours. In some ideal cases, it has opportunities to get an even higher throughput due to data’s concurrent transmission. Third, in Fig. 7(c), we can see that our model has a little bigger RSC than Local–Local model and Global–Local (normal) model, but smaller than Global–Local (cache). This is because, on one hand, the former two models have no cache tier while the caches in our model indeed add storage consumption to some degree. On the other hand, our cache is shared in a zone while the cache in Global–Local (cache) model is limited to a single node, which generates more redundant data than ours. Thus, our model consumes fewer storage resources than Global–Local (cache) does. Last, in Fig. 7(d) and (e), our model lowers down distribution cost by 17% comparing with Global–Local models, and is similar as Local–Local model. Meanwhile, they get almost the same migration costs except Global–Local (cache) model because it transfers cache and work images during migration while the others transfer only work images.

5.4. Parameters of Zone-based model

In this section, we explore the potential influence of inside parameters of our model. We focus on three inside parameters, which are similarity threshold, partition boundary, and cache size.

5.4.1. Similarity threshold

Similarity threshold is the threshold for image similarities of computing nodes. If any nodes have similarity values larger than

Fig. 8. Evaluation results under different similarity thresholds. In each subfigure, the result when the similarity threshold is at 0% acts as the baseline.

the threshold, they should be aggregated into the same zone. In this experiment, we test our model in the same environment and parameter settings except similarity threshold, which is manually set from 0% to 95%. In each experiment group, we send the same IO requests from VMs to images and record metric values continuously.

Fig. 8(a) displays the variation trends of IO latency and throughput as long as similarity threshold changes. In order to show the results clearer, we normalize them based upon the result when the similarity threshold is at 0%. During the similarity threshold increasing from 0% to 60%, the IO latency curve goes down obviously. After the threshold reaches over 60%, IO latency becomes a horizontal line. In the opposite, the throughput curve goes up before the similarity reaches 60% and maintains smooth after that. In fact, as the similarity grows when it is small, the amount of
zones increases too. The cache-hit ratio would be improved when zones’ partition becomes more and more accurate. Once arriving at a certain point that almost every single VM forms a unique zone, increasing similarity would not add partition accuracy any more. Then the cache-hit ratio becomes stable, and IO performance reaches the best level. Meanwhile, Fig. 8(b) shows the results of RSC and operation costs. Both distribution cost and migration cost are nearly the same in different similarity thresholds. This means different zone partitions affect management cost a little. Therefore, we can choose a similarity threshold flexibly for a specific cloud environment without worrying about management overhead increasing too much. However, RSC grows continuously as long as the threshold grows before 60%. This is because when the amount of zones increases, there are more zone storages which should be generated to store caches. Once the amount maintains stable, RSC becomes stable as well.

5.4.2. Partition boundary

Alike as similarity threshold, partition boundary is also a factor that affects zones’ partition, but more coarse-grained. Different partition boundaries result in different zone partitions; they decide how large a single zone could be and how many zones should exist in the whole data center. Thus, the partition boundary affects IO performance and RSC significantly. In this experiment, we test our model in different partition boundaries by changing the boundary from the host level to the global level. In each test group, all the cloud environment and parameters are the same except zones’ partition. Especially, we ensure that the mapping of VMs and computing nodes is the same. We create the same amount VMs in different boundaries, send the same IO requests to images, and record metric values continuously.

We normalize results based upon the result of the partition boundary at the host level. As shown in Fig. 9, along with the scale of a zone increasing, IO latency increases and throughput decreases. When the zone scale is small, most IO requests hit in a small scope, which avoids many global traffics and long transferring paths. Thus, the latency is lower and the throughput is higher in host level than in the other levels. When the scale expands, the cache-hit ratio goes down and long transferring paths appear. Thus, IO performance becomes dropping. For RSC, a smaller zone scale means more zones and more cache data copies. Thus, it is large in the host level partition, and it decreases when the scale expands. As the same result of changing similarity threshold, the operation costs stay stable for the same reason. Therefore, we should trade off IO performance and RSC when partitioning computing nodes into zones. If IO performance is more important, we should choose a smaller zone scale, while if resource utilization is more important, a larger zone scale should be better.

5.4.3. Cache size

Generally, cache size determines how much image data can be cached in a zone. Thus, it makes a huge impact on the cache-hit ratio. In this experiment, we test our model with different cache size limits. Similar as above experiments, we perform the tests in the same environment and settings. However, we limit cache images with different sizes. Since IO requests are the same ones, IO performance is only affected by the size of cache images.

The variation trends of metrics while changing the cache size are given in Fig. 10. Note that, all the results are normalized based upon the result when the relative cache size is at 0.1×. An interesting phenomenon is that the trends are much alike as those while we test the similarity threshold. Clearly, IO latency decreases and throughput increases when the cache size grows, and when the size reaches 3.75 times of the original size, the performance is nearly stable. This is because almost all the requested data are cached in zone storage when the cache size reaches to a certain point. Under an extreme condition, each individual host forms a distinct zone, and the cache stores all data of base images. Thus, latency and throughput just look like those in the local access case. Meanwhile, RSC keeps increasing as long as the cache size growing because a bigger size makes the cache tolerating more redundant data, which increases the storage consumption. Therefore, we should choose a proper cache size just as considering partition boundaries.

5.5. Scalability of Zone-based model

In this section, we evaluate the scalability of Zone-based model by changing some external parameters, including VM placement strategy and application access pattern. In the evaluation, we mainly focus on IO performance, examining how these parameters affect the performance of different models.

5.5.1. VM placement strategy

We simulate three VM placement strategies, which are Random placement, Balance placement and Zone-friendly placement. Among them, Random strategy places VMs to all computing nodes randomly, Balance strategy places VMs evenly on all nodes, and Zone-friendly strategy uses the method described in Section 4 to place VMs. Theoretically, different strategies affect models in various degrees. Take Zone-friendly strategy as an example, it places VMs with high similar images close to each other, resulting in low communication costs among VMs. In this experiment, we test our model with these VM placement strategies. After VMs’
initial scheduling, we send IO requests to their images and record the performance values. Under different strategies, we ensure that the cloud environment, parameter settings, and IO requests are all the same. We create the equivalent amount VMS and schedule them to the same amount computing nodes.

The results are presented in Fig. 11, in which they are normalized based upon the result of the random scheduling in Zone-based model. After using the friendly strategy, the IO latency of Zone-based model decreases greatly, reaching to only 16% of that in Random strategy. The throughput increases to about 14 times of that in Zone-based strategy and to about 17.2 times of that in Random strategy. Meanwhile, the other models do not appear obvious changes under different placement strategies, and all of them express worse performance than our model while adopting the friendly strategy. Therefore, we recommend combining Zone-friendly strategy with Zone-based model in practice.

5.5.2. Application access pattern

Application access pattern affects cache-hit ratio, and thus affects IO performance. Zone-based model is based on the CoW mechanism, which means IO requests may hit the data in different images (work image, cache image, or base image). Therefore, applications with different access patterns may express various performance in our model. Here, we simulate some cases to evaluate the influence of access patterns, in which the probability of IO requests missing in work images (miss rate) changes from 0% to 90%. In other words, the requests are redirected to base images or cache images with the probability ranging from 0% to 90%.

Fig. 12 shows the results, which are normalized based upon the result when the miss rate is at 0% in Zone-based model. For each model, a higher miss rate causes larger IO latency and lower throughput. This is reasonable because that more hit misses cause more global traffics and longer transferring paths. It is remarkable that our model gets much less performance degradation than the other models when the miss rate increases. Therefore, Zone-based model is much more scalable than the other models when there are many applications with very different access patterns.

6. Conclusion

Popularity of the cloud computing makes a huge amount of virtual machine images storing in data centers more than ever before.
Xiaolin Xu, received his B.S. (2009) degree from Huazhong University of Science and Technology (HUST) in China. Currently he is a Ph.D. student at HUST. His research interests are in the area of cloud computing and virtualization, focusing on resource management in the cloud.

Hai Jin, is a Cheung Kung Scholars Chair Professor of Computer Science and Engineering at Huazhong University of Science and Technology (HUST) in China. He is now the Dean of the School of Computer Science and Technology at HUST. He received his Ph.D. in Computer Engineering from HUST in 1994. In 1996, he was awarded a German Academic Exchange Service fellowship to visit the Technical University of Chemnitz in Germany. He worked at The University of Hong Kong between 1998 and 2000, and as a visiting scholar at the University of Southern California between 1999 and 2000. He was awarded Excellent Youth Award from the National Science Foundation of China in 2001. He is the chief scientist of ChinaGrid, the largest grid computing project in China, and the chief scientist of National 973 Basic Research Program Project of Virtualization Technology of Computing System. He is a senior member of the IEEE and a member of the ACM. He is the member of Grid Forum Steering Group (GFG). He has co-authored 15 books and published over 400 research papers. His research interests include computer architecture, virtualization technology, cloud computing and grid computing, peer-to-peer computing, network storage, and network security.

Song Wu, is a professor of Computer Science and Engineering at Huazhong University of Science and Technology (HUST) in China. He received his Ph.D. from HUST in 2003. He is now the director of Parallel and Distributed Computing Institute at HUST. He has also served as the vice director of Service Computing Technology and System Lab (SCTS) and Cluster and Grid Computing Lab (CGCL) of HUST. His current research interests include grid/cloud computing and virtualization technology.

Yihong Wang, received his B.S. (2012) degree from Huazhong University of Science and Technology (HUST) in China. Currently he is a master student at HUST. His research interests are in the area of cloud computing and virtualization, focusing on resource management in the cloud.