Frontiers of Information Technology & Electronic Engineering
www.zju.edu.cn/jzus; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

VirtMan: design and implementation of a fast booting system for homogeneous virtual machines in iVCE*

Zi-yang LI, Yi-ming ZHANG†‡, Dong-sheng LI, Peng-fei ZHANG, Xi-cheng LU

(*National Laboratory for Parallel and Distributed Processing, School of Computer, National University of Defense Technology, Changsha 410073, China*)

†E-mail: ymzhang@nudt.edu.cn

Received July 8, 2015; Revision accepted Dec. 17, 2015; Crosschecked Jan. 6, 2016

Abstract: Internet-based virtual computing environment (iVCE) has been proposed to combine data centers and other kinds of computing resources on the Internet to provide efficient and economical services. Virtual machines (VMs) have been widely used in iVCE to isolate different users/jobs and ensure trustworthiness, but traditionally VMs require a long period of time for booting, which cannot meet the requirement of iVCE's large-scale and highly dynamic applications. To address this problem, in this paper we design and implement VirtMan, a fast booting system for a large number of virtual machines in iVCE. VirtMan uses the Linux Small Computer System Interface (SCSI) target to remotely mount to the source image in a scalable hierarchy, and leverages the homogeneity of a set of VMs to transfer only necessary image data at runtime. We have implemented VirtMan both as a standalone system and for OpenStack. In our 100-server testbed, VirtMan boots up 1000 VMs (with a 15 GB image of Windows Server 2008) on 100 physical servers in less than 120 s, which is three orders of magnitude lower than current public clouds.

Key words: Virtual machine, Fast booting, Homogeneity, Internet-based virtual computing environment (iVCE)
http://dx.doi.org/10.1631/FITEE.1500216 **CLC number:** TP399

1 Introduction

With the proliferation of virtualization and service computing technology, clouds (Armbrust *et al.*, 2010) are becoming increasingly popular as an Internet-based computing paradigm providing IaaS (Infrastructure as a Service), PaaS (Platform as a Service), and SaaS (Software as a Service). However, the newly emerging characteristics of Internet-based applications bring severe challenges to cloud computing. First, the scale of Internet services, including user scale, data scale, and task scale, is increasing exponentially in recent years. More seriously, their demand for resources often varies dramatically over a short period of time. Second, Internet service providers become increasingly dependent on public platforms to provide their services, and meanwhile they expect the users to participate in improving the services (e.g., online videos). Such a 'utility' model brings difficulty in guaranteeing trustworthiness of the services.

Internet-based virtual computing environment (iVCE) (Lu *et al.*, 2006) has been proposed to combine data centers and other kinds of computing resources on the Internet (e.g., edge servers, client PCs, and mobile devices) that are referred to as multi-scale resources in iVCE, in order to provide efficient and economical services. Beyond data centers, iVCE tries to provide users with a transparent, all-in-one solution through virtualization and

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (Nos. 61379055 and 61379053)

ORCID: Yi-ming ZHANG, http://orcid.org/0000-0001-6450-8485

autonomization by aggregating the resources. However, the resources usually have different characteristics and management patterns, which lead to challenges in trustworthiness and efficiency for the computing environment. Virtual machines (VMs) have been widely used to isolate different users/jobs and ensure trustworthiness in the clouds and are promising in addressing these challenges. However, traditionally VMs require a relatively long period of time for booting. For example, to boot up one 4 GB VM, it takes about 400 s in Amazon EC2, almost 800 s in Microsoft Azure, and nearly 300 s in Rackspace. More seriously, it takes much longer time to boot up multiple VMs in these clouds.

Many cloud applications (e.g., animation rendering, material analysis, and fluid simulation) require hundreds or even thousands of VMs to run concurrently for performance purpose (Li *et al.*, 2015). The long boot-up time cannot meet the requirement of such a large-scale and highly dynamic environment where (1) the applications need to run a large number of parallel VMs (Mao and Humphrey, 2012) and (2) there are often applications waiting for resources in the queue. Therefore, in iVCE we focus on this challenge and propose to support fast booting of as many VMs as needed by the applications, which is referred to as the ability of on-demand elastic binding.

To achieve this goal, in this study we design and implement VirtMan, a fast booting system for a large number of homogeneous VMs in iVCE. Different from the state-of-the-art schemes like peer-to-peer (P2P) image dissemination (Chen *et al.*, 2009; Peng *et al.*, 2012), remote memory fork (Lagar-Cavilla *et al.*, 2009), or image stripping on top of distributed file systems (Meyer *et al.*, 2008; Nicolae *et al.*, 2011), VirtMan uses the Linux-IO (LIO) target (http://linux-iscsi.org/wiki/Main_Page) to remotely mount to the source image in a scalable hierarchy, and leverages the homogeneity of a set of VMs to transfer only necessary image data at runtime. VirtMan realizes the cache-group-based snapshot mechanism to ensure that only one copy of data is transferred for one physical server, no matter how many homogeneous VMs are booted up on that server. We have implemented VirtMan both as a standalone system and for OpenStack (https://www.openstack.org/), which is a well-known VM management framework. In our 100-server testbed, VirtMan successfully boots up 1000 VMs (with a 15 GB image of Windows Server 2008) on 100 physical servers in less than 120 s, which is three orders of magnitude lower than current public clouds.

2 Overview

System-level VMs provide an efficient, isolated duplicate of a real machine (Smith and Nair, 2005). The VM performs the execution of a complete operating system (OS) that is hosted on a physical machine (a.k.a. compute node) and controlled by the hypervisor software (such as Xen and Linux KVM). The VM image (VMI for short) is the backend storage of the VM containing operating systems, user applications, and files. An effective VMI must be ready before the VM starts.

VirtMan builds a valid path towards the VMI storage (which may be raw, qcow2, etc.), and then boots up the VM OS without waiting for complete preparation of the image data. The VirtMan system contains mainly two modules, i.e., the Volt Coordinator and the VirtMan Compute (Fig. 1).

The Volt Coordinator is a central controller providing necessary configuration information for all the compute nodes, so that they can use the Linux SCSI targets, such as Internet SCSI (iSCSI), to remotely mount to the source image in a scalable hierarchy. Fig. 1 shows an example of a simple binary tree hierarchy, where each inner node has two children.

Each compute node has a single instance of the VirtMan Compute module, which has three main components, namely, cache group, snapshot, and hierarchical attaching. The compute node attaches the image from existing mount points in a hierarchal way (coordinated by Volt), caches the image data locally, and boots up VMs on top of the cache-based snapshots. The lower-level compute nodes in the hierarchy retrieve image data from the cache of their upper-layer nodes respectively (or the original image storage).

Besides the Volt Coordinator and the VirtMan Compute, we implement a test framework for unit testing and integrate the entire project into OpenStack (https://www.openstack.org/). The total numbers of code lines for the components are listed in Table 1.

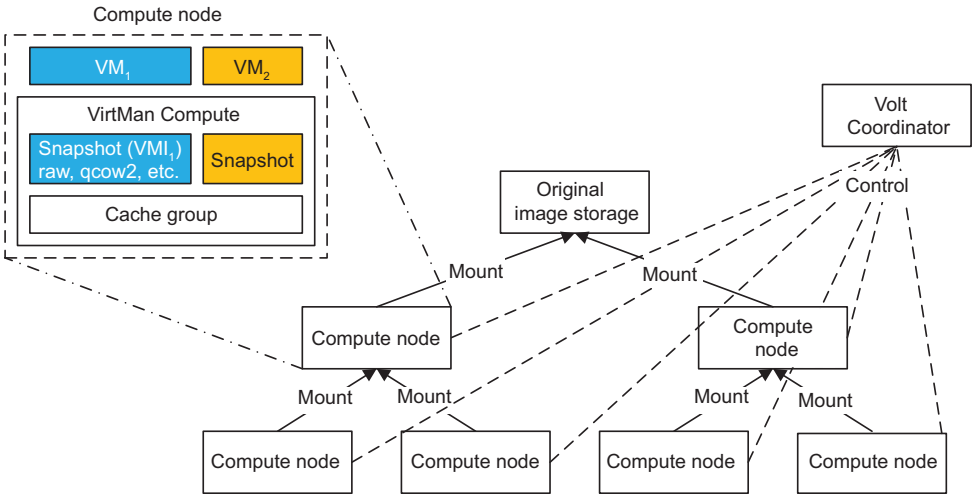

Fig. 1 VirtMan overview

Table 1 The total numbers of code lines for the components

Component	N_{tcl}	Component	N_{tcl}
Volt	16 599	Test	1724
Compute	5259	OpenStack	141

N_{tcl}: total number of code lines

3 Design and implementation

In this section we first introduce the design and implementation of VirtMan's main components (the Coordinator and the VirtMan Compute). Then we summarize the APIs of VirtMan. At last, we present the details of the test framework for VirtMan.

3.1 Volt Coordinator

As shown in Fig. 1, we design and implement Volt to coordinate and track the volume attaching information of all the servers. Volt is a coordinating service deployed on a global master, which stores the attaching hierarchy information, maintains the hierarchy, handles volume requests from compute nodes, and removes dead nodes from the hierarchy. Volt provides the following functions for its clients (i.e., the compute nodes):

1. Search a volume: A client sends a request to search metadata of volumes matching a specific identifier. The client uses the resulting volumes to build the iSCSI connections. Volt always returns a partial matching list in order to improve the overall performance and limit the number of connections.

2. Register a volume: This function is to register a newly added volume and assign an identifier to the volume. When a volume is registered to Volt, it can be attached via the iSCSI protocol.

3. Remove metadata: When a compute node is shut down, its volume should also be removed from the hierarchy. Therefore, before a client quits it will send a removal request to Volt to remove all the metadata of its volume.

4. Send heartbeat: Volt monitors the states of all its clients, each of which periodically sends heartbeats to Volt. If Volt cannot receive heartbeats from a client for a long period of time, it will treat the client as dead and remove all the attached volumes from it.

Volt supports different kinds of tree hierarchies for attaching the compute nodes. Currently we use mainly the (unbalanced) binary tree structure, where the new client is always inserted to the position that has the shortest path to the root.

3.2 VirtMan Compute

3.2.1 Cache group

Cache is desirable when reading/writing data that is stored in the slow storage devices (e.g., disks). There have been many block-level local cache solutions, such as Bcache (http://bcache.evilpiepirate.org/), Dm-cache (https://en.wikipedia.org/wiki/Dm-cache/), Flashcache (https://en.wikipedia.org/wiki/Flashcache), and LVM-cache (http://man7.org/linux/man-pages/man7/lvmcache.7.html), which use the fast

storage (e.g., SSD) as the cache of local disks. Similarly, when multiple VMs on the same compute node read the same image data from remote storage servers, local cache will remarkably reduce the network I/O and improve the I/O performance.

However, currently there is no general cache support for block devices in the state-of-the-art VM management systems (such as OpenStack, CloudStack, and AWS), and the image data is usually stored in the volumes of storage servers which are attached to the compute nodes. This may result in severe bottleneck of I/O performance when reading image data during the boot-up time.

In this subsection we design CacheGroup, which caches the image data on local storage devices of the compute nodes so that they can access cached data in their local caches instead of from the remote storage servers. Several challenges have to be addressed to achieve this goal: (1) Since compute nodes dynamically attach and release volumes from storage servers, the cache scheme must support dynamically changing configurations to support addition and removal of volumes at any time; (2) The cache should be transparent so that users can use the cache in the same way as they directly use the volume.

To address these challenges, we implement CacheGroup by grouping both the remote storages and local cache devices (such as local SSD and disks), as shown in Fig. 2. CacheGroup uses DM-linear (https://www.kernel.org/doc/Documentation/device-mapper/linear.txt) to create a logical group for remote volumes (e.g., sda and sdb) and combine the local storages (e.g., ssd0 and ssd1). We call these two linear mapping groups the 'HDD group' and 'SSD group', respectively (The concepts of HDD and SSD are borrowed from flashcache, and we use them to indicate the original device and the cache media here). The HDD group is being cached and the SSD group is used as cache media.

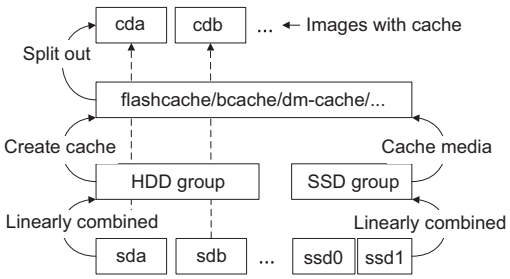

Fig. 2 Cache group principle

It makes cache of the logical volume group using the combined local storage, which is called a cached group. When adding a new remote volume to the cache group, we add them in the HDD group first, and CacheGroup creates a corresponding cached volume out of the cached group by using DM-Linear mapping. When removing a volume from the logical volume group, FlashCacheGroup (FCG) removes the cached volume accordingly. Once the cached volume is removed, its space in the group will be marked as an error.

We set up a mount point for each cached device, so that other compute nodes can mount to it and read data from the device. Therefore, the cached devices serve not only local VMs, but also the remote ones.

3.2.2 Cache-based snapshot

VirtMan builds the snapshot (https://www.kernel.org/doc/Documentation/device-mapper/snapshot.txt) to boot up VMs from the cached image. However, since the cache might be shared by multiple homogeneous VMs on the same compute node, each VM must have its own snapshot to avoid inconsistency. A VM writes its private data to its snapshot without modifying the read-only image cache. The structure of the cache-based snapshot is depicted in Fig. 3. The cache provides the read-only data from the original volume (referred to as VolumeO), and for each VM there is a private writable volume (referred to as VolumeU) for recording the differences written by the VM during its runtime. VolumeO and VolumeU together form the snapshot.

A compute node creates and uses the cache-based snapshot in the following steps: (1) Attach the remote original volume (through iSCSI) as a

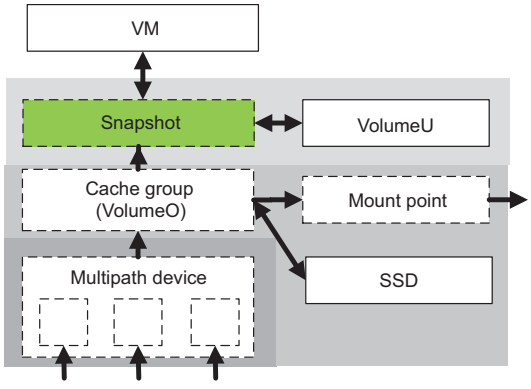

Fig. 3 Cache-based snapshot

read-only volume (VolumeO); (2) Use local storage of the compute node as a (shared) cache of VolumeO; (3) Create a writable difference volume (VolumeU) for each VM to store the VM's private data; (4) Create the snapshot (using the Linux device mapper module) upon the cache and the writable VolumeU; (5) Boot up a VM on top of the snapshot.

Fig. 3 depicts the I/O flow of VirtMan Compute. The VM interacts with its snapshot, writing private data to VolumeU and reading shared image data from the CacheGroup. The CacheGroup can also be mounted to other compute nodes which will become the children of this node in the hierarchy depicted in Fig. 1. The cached image data is stored in the local storage such as an SSD. When a cache miss happens, the read request will be forwarded to one path of the multipath device (which will be discussed in the next subsection).

3.2.3 Hierarchical attaching

Using the cache-based snapshot, VirtMan transfers only one piece of boot-up data for all the homogeneous VMs on the same physical server. However, if there are a large number of physical servers reading the boot-up data from the original volume, there might be a bottleneck both on the original server and in the network.

VirtMan uses hierarchical attaching to avoid the possibility of bottleneck. As shown in Fig. 1, the original volume is the root of a template volume hierarchy, and each VM fetches only the necessary data from its parent over the iSCSI protocol. As introduced previously, VirtMan makes use of a compute node's local storage as a cache to provide the read-only VolumeO, and prepares each compute node with a writable VolumeU for writing difference to VolumeO. The compute nodes at lower levels of the hierarchy attach to only the nodes at their direct upper level, which avoids the bottleneck at the original volume and accelerates the image transferring process.

Each intermediate node in the hierarchy (Fig. 1) provides the image mount points for its child nodes. The VirtMan Compute module validates the mount points and then mounts them locally as the virtual block devices, each of which is a valid path towards the original image storage. We view these virtual block devices as a multipath device. The (missed) I/O requests are forwarded to one of these paths that is chosen by specific algorithms like round-robin, least service time, and minimum queue length. When one path fails, the requests will be forwarded to other available paths. Clearly, the depth of the hierarchy is $O(\log N)$, where N is the total number of the compute nodes.

3.3 VirtMan interfaces

VirtMan provides a set of application program interfaces (APIs) (Table 2) that support to fast boot up many homogeneous VMs simultaneously. The VirtMan APIs can be used by IaaS platforms including OpenStack (as introduced in the next section) and CloudStack. These platforms can support the basic VM operations like creation, clone, migration, and billing. The VirtMan APIs accelerate the process of creating, cloning, and migrating VMs.

1. When creating a new VM on a compute node, the VirtMan Compute instance on the node first queries the Volt Coordinator for available parent mount points, and then validates and mounts them. After that VirtMan creates the multipath and cache group, and then the compute node also becomes a mount point for others and registers to the Volt Coordinator. Finally, it creates a private snapshot and returns the snapshot path.

2. The destroy function is the reverse processing of creating a VM.

3. For clone and migration functions, VirtMan Compute copies the configuration file and the private snapshot data of the VM to the destination, and then boots up a new VM there.

4. For the snapshot function, VirtMan Compute creates a new snapshot on the top of storage stack. Then the VM private data will be writen to the new snapshot device.

5. The list function is used to list the VirtMan configurations on the node.

3.4 Unit test

We have done automatic unit tests for guaranteeing the correctness of the VirtMan source code. The unit tests of VirtMan are based on testtools (Lange, 2015), along with a command line tool named tox (Krekel, 2015) for test management. Testtools is a testing framework of OpenStack that extends the Python standard unit test library (https://docs.python.org/2/library/unittest.html).

Table 2 VirtMan APIs

API name	Parameters	Return value	Description
create	vm_name, image_name, image_location	snapshot_path	Create a valid path towards the VMI
destroy	vm_name, instace_only*	True if success, false otherwise	Destroy the VirtMan configurations of a VM
snapshot	vm_name, snapshot_name, snapshot_location	True if success, false otherwise	Create a snapshot for the VM
clone	vm_name, destination	True if success, false otherwise	Clone VirtMan configurations and private snapshot data of the VM to the destination
migrate	vm_name, destination	True if success, false otherwise	Migrate VirtMan configurations and private snapshot data of the VM to the destination
list		List of VMs	List VirtMan configurations on the node

* The parameter is optional

With abundant assertion functions, testtools is used to check the corresponding output in a general routine by scripting test cases for given test sets. A test suite is composed of several test cases with the same environment. Configurations are initialized by the setup() function and cleared by the teardown() function. We implement a test suite for each component. In our test the functions are delivered to and executed by different objects, reducing the dependency between different components. We use control inversion (https://en.wikipedia.org/wiki/Inversion_of_control) to ensure isolation, and use dependence injection (which replaces the initial dependencies with specified external ones) to decouple the components in VirtMan. We deploy the fake and mock tests (https://code.google.com/p/mock/) for VirtMan. Fake extends the same abstract class as its entity while the logic of fake only processes tests. For example, the base image can be injected using dependence injection and then we can use fake to carry out all the image tests. In the test of the snapshot component of VirtMan, we use mock to isolate file operations and obtain the expected input. Overall the test coverage for VirtMan has reached 93%.

4 Integration with OpenStack

4.1 Lightweight integration

Currently OpenStack (https://www.openstack.org/) is the most powerful and widely used VM management system. It has several components including mainly: OpenStack Compute (Nova), which is the brain and allows the user to create and manage virtual servers using the machine images; OpenStack Block Storage (Cinder), which provides persistent block storage; OpenStack Object Storage, which stores and retrieves unstructured data objects; OpenStack Networking (Neutron), which is responsible for managing networks (e.g., IP address, VLAN, and firewall); OpenStack Image Service (Glance), which provides the discovery, registration, and delivery of disk and server images; and OpenStack Dashboard (Horizon), which provides a web-based portal to interact with other OpenStack services.

In this section we briefly introduce the integration of VirtMan with OpenStack. Currently, OpenStack provides two categories of methods for booting up a VM: booting from a local image and booting from a remote Cinder volume. The first category needs to copy the entire image to a compute node, making it suffer from a long transfer delay for large images. The second category remotely attaches a volume to a VM and transfers only the necessary data from the volume, thus having better performance. However, this approach allows only booting up a single VM from a volume at a time. Moreover, preparing a volume for each VM requires a long time. So, it is inevitable to take a long time for booting up a large number of VMs in OpenStack.

To overcome this shortcoming, we integrate VirtMan as a third-party library into OpenStack. Since the coordinator and the hierarchical attaching functionality do not interact with OpenStack directly, we need only to incorporate the cache group and snapshot functionalities, both of which are for the compute side (i.e., modifications are needed mainly to the Nova component). For the cache group we have the following modifications: (1) The cache

modules (flashcache, bcache, etc.) are added to the drivers; (2) We add a parameter to attach_volume(), which indicates whether to use cache or not, and if the parameter is true then add the volume to the cache group after it is attached; (3) Nova sets up a database record to indicate which volume is cached; (4) When detaching a volume, we will remove the cache first; (5) To support dynamic addition/removal of volumes, we organize the cached volumes as a group and backing devices can be attached and detached at runtime. The users can configure nova.conf and set 'use_cachegroup = true' to enable and use the cache group. The integration relies on some other packages specific to the cache schemes: for FlashCacheGroup, Facebook's FlashCache must have already been installed. For BcacheGroup, the Linux kernel has to be greater than or equal to 3.10.

For cache-based snapshot, the modification to Nova is light-weighted: (1) For snapshot creation, we add a driver class which extends the original class 'DriverVolumeBlockDevice' in file nova/virt/block_device.py (+ about 50 lines) to prepare the snapshot; (2) For snapshot deletion, we call the delete method of snapshot in file nova/compute/manager.py (+ about 20 lines). We also modify Horizon to add an option 'boot from snapshot' in the drop-down list, which can be selected in the dashboard allowing users to specify the number of homogeneous VMs to boot simultaneously. The users need to configure nova.conf and set 'use_vmtsnapshot = true' to enable the option in the dashboard and use the snapshot. Besides, we add validation for the POST request messages (from Nova-client) at the Nova-API module. All modifications are depicted in Fig. 4 with red-lined rectangles.

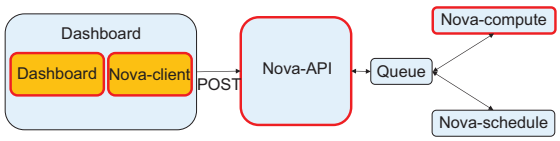

Fig. 4 Integration with OpenStack (references to color refer to the online version of this figure)

4.2 Experiences

OpenStack does not perform well in supporting a large number of VMs since its controllers cannot maintain too much VM information (https://www.openstack.org/). As shown in our evaluation in Section 5.3, a large fraction of the booting VMs may fail if we directly use VirtMan in OpenStack. To address this problem, we analyze the message queue of OpenStack Nova and find that many messages are unacknowledged at Nova-conductors.

Our basic idea for alleviating the bottleneck is to add more 'workers'. In OpenStack, workers are the daemon processes for the OpenStack services, which respond to the requests in the queue and execute a series of practical operation commands. The scheduler allocates the requests to different service nodes and a suitable worker is started to conduct corresponding operations, so more workers in use can increase the concurrency of the whole service under certain conditions. To support high concurrency in practical deployment, clearly we should have more Nova Compute API workers, Nova conductor workers, glance API workers, and Cinder API workers. However, simply increasing the number of the workers cannot achieve better effects, and we further make the following improvements for a practical configuration of message queues and databases:

4.2.1 Modifying remote procedure call (RPC) parameters

Under high load conditions, some requests will not obtain a timely response. There are certain timeout mechanisms for OpenStack services and if these services do not return the expected results within a certain time, the execution will be considered as failures. Meanwhile, the repeat mechanism will wait for some time and retry the operation under high load. Thus, it is prone to be judged as a timeout for some operations when they are in execution or waiting. Such problems can be solved to some extent by extending the response time of remote procedure call (RPC) and adjusting the numbers. We have adjusted the following RPC parameters, namely, response time for network configuration, the number of network configuration retries, response time for the block device configuration, and the number of device configuration block retries.

4.2.2 Optimizing MySQL

Status of VMs will be frequently updated during the deployment, creating a large number of database connection accesses. In the MySQL master node, the default connection limit can be eliminated by setting the maximum number of connections larger.

In the compute nodes, using the database connection pool can alleviate excessive database connection. The minimum pool size, maximum pool size, and maximum overflow should be added in the pool configuration. At the same time, MySQL Galera is helpful in improving the performance and availability of the MySQL cluster.

4.2.3 Configuring HA RabbitMQ

The command distribution of the entire OpenStack system is based on the advanced message queuing protocol (AMQP). System stability depends upon the proper functioning of the message queue. The default OpenStack AMQP is implemented by RabbitMQ. In practical deployment, we use the active-active mode high-availability RabbitMQ cluster and use HAProxy to guarantee the availability of services. In the configuration of the RabbitMQ, we configure a greater VM memory with a high watermark value to allow RabbitMQ occupying more memory in response to a connection and a larger maximum number of open files to the socket limit to carry higher loads.

5 Evaluation

5.1 Testbed

We have built a testbed with 100 PowerLeader servers and five Pronto 3290 48-port 1 GbE switches (located in 4 racks). Each server has 12 Intel Xeon E5-2640 2.5 GHz cores and 64 GB RAM, and installs two Hitachi 7200 r/min, 1 TB disks, and one 1 GbE 2-port NIC. We used five switches to build a 100-node tree: each of the first four switches connecting to 25 servers, and the fifth switch acting as the aggregate switch obtaining a relatively high oversubscription ratio of 1:25.

We used different numbers of physical servers to boot up different numbers of VMs. Our evaluation answers the following questions: How fast can VirtMan boot up a large number of homogeneous VMs on a different number of physical servers (Section 5.2)? How well does VirtMan perform when integrated into OpenStack (Section 5.3)? And what is the advantage of VirtMan compared with schemes that are based on distributed file systems (DFSs) (Section 5.4)?

5.2 Standalone VirtMan

We used one additional server to run the coordinator which is responsible for constructing the attaching hierarchy, i.e., designating each physical server with an upper-level parent that can be attached via the iSCSI protocol. We prepared a 15 GB image of Windows Server 2008 at the original volume server, and let the original volume server and each physical server have no more than two children in the hierarchy. The number of physical servers varied from 10 to 100, each of which booting up a fixed number (10) of VMs. Therefore, in our experiments VirtMan booted up 100–1000 VMs. In each experiment we evaluated the maximum delay of all the boot-up VMs.

The average boot-up time for each experiment is depicted in Fig. 5, where each point is an average of five runs. From this figure we conclude that as the numbers of physical servers and VMs increase, the maximum boot-up time ($O(\log N)$ as discussed in Section 3.2.3) increases very slowly (from 60 s to 116 s). The boot-up time has a jump when the number of physical servers increases from 10 to 20, from 30 to 40, and from 60 to 70, which is due to the increase of the number of attaching levels in the hierarchy (when the number of servers reaches $16 = 2^4$, $32 = 2^5$, and $64 = 2^6$, respectively): the iSCSI protocol requires 10 s for each attachment. In the worst case of 100 physical servers, VirtMan successfully boots up 1000 VMs in less than 120 s, which is three orders of magnitude lower than current public clouds.

5.3 OpenStack-based VirtMan

We also evaluated the boot-up process of OpenStack-based VirtMan. As discussed in the OpenStack community (https://www.openstack.org/), the current implementation (version Kilo) of OpenStack performs not well in supporting a large number of VMs since its controllers have to maintain too much VM information. Therefore, our first evaluation tested how many VMs can be booted up in an original OpenStack cluster, where each physical server runs a fixed number (10) of VMs.

The results are depicted in Fig. 6, where each point is an average of five runs. We conclude that the original OpenStack performs well when the number of physical servers is less than 20 (corresponding to

<200 VMs). However, when the number of physical servers exceeds 30, the failure rate of OpenStack will sharply increase. For example, the failure rate is over 80% when booting up 400 or 500 VMs (on 40 or 50 physical servers, respectively).

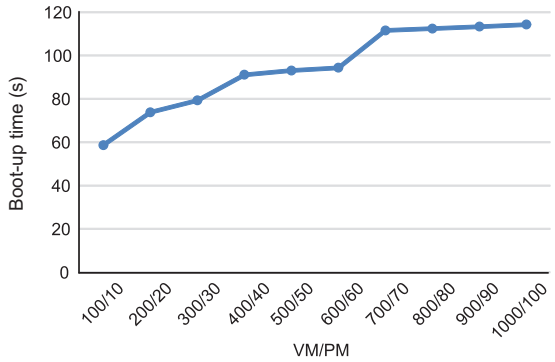

Fig. 5 Standalone VirtMan boot-up time (VM: number of virtual machines; PM: number of physical machines)

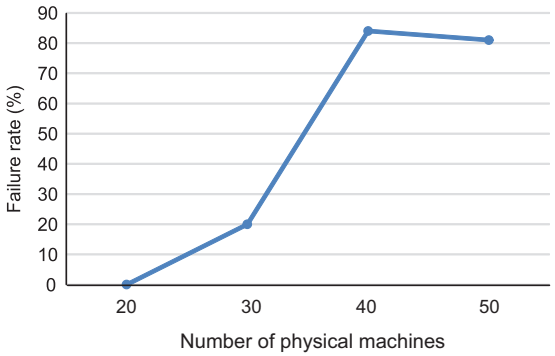

Fig. 6 Original OpenStack failure rate when booting up a large number of VMs

As discussed in Section 4.2, to alleviate the bottleneck problem, we (1) added more workers for OpenStack services, (2) modified the RPC parameters, (3) optimized the MySQL database, and (4) configured the HA RabbitMQ. By this means we currently can support at most 500 VMs in one OpenStack cluster.

We used the improved configuration of OpenStack to evaluate the boot-up process with/without VirtMan. Since currently OpenStack cannot support more than 500 VMs booting concurrently, we evaluated the process for 200 and 500 VMs, respectively. All other configurations were the same as in previous tests. The results are depicted in Fig. 7, where each point is an average of five runs. Our conclusion for this figure is two-fold. First, the boot-up time of VirtMan-enabled OpenStack is more than one order of magnitude lower than the original OpenStack, which proves that VirtMan effectively reduces the boot-up time when integrated with OpenStack. Second, the boot-up time of OpenStack-based VirtMan is longer than that of the standalone VirtMan. This is because here OpenStack has to handle many events (that are orthogonal to VirtMan), which may result in long waiting time in the message queue.

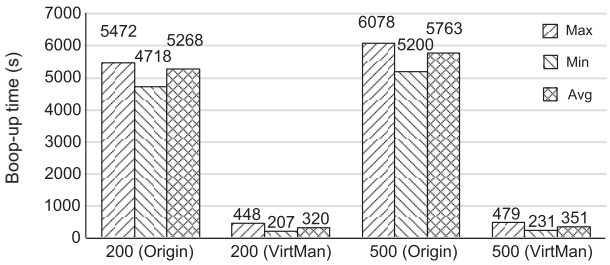

Fig. 7 OpenStack-based VirtMan boot-up time. 'Origin' represents the original OpenStack and 'VirtMan' represents the VirtMan-enabled OpenStack

Several approaches might alleviate the bottleneck problem. First, we may use HA (high availability) in the OpenStack cluster to enable multiple controllers, e.g., to use rabbit HA for RabbitMQ, to use galara (http://galeracluster.com/products/) for MySQL DB, and to use HAProxy for load balancing of DB and Cinder-API. Second, we may use hierarchical deployment to support multiple sub-clusters in one OpenStack cluster and limit the number of VMs of each sub-cluster.

5.4 VirtMan vs. DFS-based schemes

Many public cloud providers support users to boot up VMs on top of distributed file systems (DFSs), which can balance the load of I/O among many physical servers underlying DFS. In this subsection we compared VirtMan with Ceph, the most popular and widely used DFS. We evaluated the VM boot-up process on Ceph's RADOS Block Devices (RBD) system built on 30 physical servers. We copied the 15 GB Windows 7 image to RBD, and used snapshot to boot up 300 VMs. This process took about 40 s, which was comparable with the standalone VirtMan. However, several drawbacks cannot be resolved in this solution. First, the write throughput of current Ceph could achieve only 15 MB/s (Weil *et al.*, 2006), much slower than that of a normal disk (about 100 MB per second). Second,

it took about 3 h to install the 15 GB Windows 7 system on Ceph, and more than 10 min to copy an installed image to Ceph, both of which were much longer than the boot-up time. Therefore, VirtMan outperforms DFS-based schemes both in the overall boot-up time and in normal write operations after the systems have been booted up.

6 Related work

6.1 Storage and cache

Backend storage optimization is an important approach to accelerating the boot-up process of VMs. The DFSs like GlusterFS and MooseFS can decrease the size of transferred volumes. However, the I/O pressure on the storage servers increases dramatically when powering on a large number of homogeneous VMs, since there may not be enough replicas on the storage servers for offloading the I/O demands (Zhang *et al.*, 2015).

Some storage systems specifically designed for VM image (Meyer *et al.*, 2008; Wartel *et al.*, 2010; Shamma *et al.*, 2011) make great effort in enabling VM specific functions (e.g., create, snapshot, clone, and migrate). They improve VM provisioning performance by snapshot and cache in local storage. The block-level storage service for VMs has been provided (Flouris and Bilas, 2005; Flouris *et al.*, 2008).

Cache is desirable when the same piece of data is read multiple times. DM-Cache uses I/O scheduling and cache management techniques optimized for flash-based SSDs. The device mapper target (DM-Cache) reuses the metadata library used in the thin-provisioning library. Both write-back and write-through are supported by DM-Cache. The problem of DM-Cache is that its metadata device is not easy to handle. LVM-Cache is built on top of DM-Cache so that logical volumes can be turned into cache devices. Thus, LVM-Cache splits the cache pool LV into two devices, namely, the cache data LV and cache metadata LV. LVM-Cache will face the same problem as DM-Cache.

6.2 OpenStack

Nova's image-caching facility reduces the start-up time for creating homogeneous VMs on one Nova-compute node. However, it helps neither the first-time provisioning nor the Cinder-based booting process. Direct image access uses the direct_url of the Glance v2 API, such that the number of hops to transfer an image to a Nova-compute node is decreased. When images are stored at multiple backend locations, the Nova-compute servers can select a proper image storage to speed up the downloading process. This approach cannot reduce the boot-up data transfer for multiple VMs on one host.

Multi-attach volume allows a volume to be attached to more than one instance simultaneously. As a result, volumes can be shared among multiple guests when the instances are already available. Besides, these volumes can be used to boot up a number of VMs by enforcing the multi-attach volumes as read-only image disks. Unfortunately, this approach does not scale well because of the star-structured topology in the data dissemination process.

6.3 P2P-based image transferring

The P2P protocol (Zhang *et al.*, 2010; Zhang and Liu, 2012; Zhao *et al.*, 2014) can accelerate the transfer of the image files. For example, in OpenStack the glance-bittorrent-delivery proposal (https://blueprints.launchpad.net/glance/+spec/glance-bittorrent-delivery) transfers the image templates from the glance storage to Nova-compute servers. This approach, however, needs to transfer the entire image to all peers. Squirrel (Razavi *et al.*, 2014) proposes to scatter VM image contents on IaaS compute nodes.

VMThunder has been proposed in our previous work (Zhang *et al.*, 2014). It downloads image data using iSCSI and speeds up VM image transfer by strategically integrating P2P streaming techniques. The idea is inherited by VirtMan. However, VMThunder implements only a bundle of bash scripts to manually configure an emulation environment for VM booting. In contrast, VirtMan is a complete working system that already runs in production environments, e.g., CNCERT/CC (www.cert.org.cn). All the components we designed and implemented in VirtMan, including the Volt Coordinator, the CacheGroup, the cache-based snapshot, and the hierarchical attaching procedure, are brand-new contributions. Moreover, we have integrated VirtMan into OpenStack, which proves VirtMan's flexibility and compatibility and supports common VM managements like snapshot, migration, and clone. All the source

codes of VirtMan components can be publicly available at https://github.com/vmthunder/virtman and https://github.com/vmthunder/volt.

7 Conclusions

Traditionally it takes a long period of time to boot up a large number of VMs (with large image sizes), which cannot meet the requirement of large-scale and highly dynamic applications. To address this problem, in iVCE we design and implement VirtMan, a fast booting system for a large number of homogeneous VMs in iVCE. VirtMan uses the iSCSI protocol to remotely mount to the source image in a scalable hierarchy, and leverages the homogeneity of a set of VMs to transfer only necessary image data at runtime. We have implemented VirtMan both as a standalone system and for OpenStack, which dramatically accelerates the VM boot-up process and realizes the elastic binding property of iVCE.

References

Armbrust, M., Fox, A., Griffith, R., *et al.*, 2010. A view of cloud computing. *Commun. ACM*, **53**(4):50-58. http://dx.doi.org/10.1145/1721654.1721672

Chen, Z., Zhao, Y., Miao, X., *et al.*, 2009. Rapid provisioning of cloud infrastructure leveraging peer-to-peer networks. Proc. 29th IEEE Int. Conf. on Distributed Computing Systems Workshops, p.324-329. http://dx.doi.org/10.1109/ICDCSW.2009.35

Flouris, M.D., Bilas, A., 2005. Violin: a framework for extensible block-level storage. Proc. 13th NASA Goddard Conf. on Mass Storage Systems and Technologies, p.128-142. http://dx.doi.org/10.1109/MSST.2005.41

Flouris, M.D., Lachaize, R., Bilas, A., 2008. Orchestra: extensible block-level support for resource and data sharing in networked storage systems. Proc. 14th IEEE Int. Conf. on Parallel and Distributed Systems, p.237-244. http://dx.doi.org/10.1109/ICPADS.2008.110

Krekel, H., 2015. Python Tox 2.3.1. Available from https://pypi.python.org/pypi/tox [Accessed on June 28, 2015].

Lagar-Cavilla, H.A., Whitney, J.A., Scannell, A.M., *et al.*, 2009. SnowFlock: rapid virtual machine cloning for cloud computing. Proc. 4th ACM European Conf. on Computer systems, p.1-12. http://dx.doi.org/10.1145/1519065.1519067

Lange, J.M., 2015. Python Testtools 1.8.1. Available from https://pypi.python.org/pypi/testtools [Accessed on June 28, 2015].

Li, J., Li, D., Ye, Y., *et al.*, 2015. Efficient multi-tenant virtual machine allocation in cloud data centers. *Tsinghua Sci. Technol.*, **20**(1):81-89. http://dx.doi.org/10.1109/TST.2015.7040517

Lu, X., Wang, H., Wang, J., 2006. Internet-based virtual computing environment (iVCE): concepts and architecture. *Sci. China Ser. F*, **49**(6):681-701. http://dx.doi.org/10.1007/s11432-006-2030-6

Mao, M., Humphrey, M., 2012. A performance study on the VM startup time in the cloud. Proc. 5th Int. Conf. on Cloud Computing, p.423-430. http://dx.doi.org/10.1109/CLOUD.2012.103

Meyer, D.T., Aggarwal, G., Cully, B., *et al.*, 2008. Parallax: virtual disks for virtual machines. *ACM SIGOPS Oper. Syst. Rev.*, **42**(4):41-54. http://dx.doi.org/10.1145/1357010.1352598

Nicolae, B., Bresnahan, J., Keahey, K., *et al.*, 2011. Going back and forth: efficient multideployment and multi-snapshotting on clouds. Proc. 20th Int. Symp. on High Performance Distributed Computing, p.147-158. http://dx.doi.org/10.1145/1996130.1996152

Peng, C., Kim, M., Zhang, Z., *et al.*, 2012. VDN: virtual machine image distribution network for cloud data centers. Proc. IEEE INFOCOM, p.181-189. http://dx.doi.org/10.1109/INFCOM.2012.6195556

Razavi, K., Ion, A., Kielmann, T., 2014. Squirrel: scatter hoarding VM image contents on IaaS compute nodes. Proc. 23rd Int. Symp. on High-Performance Parallel and Distributed Computing, p.265-278. http://dx.doi.org/10.1145/2600212.2600221

Shamma, M., Meyer, D.T., Wires, J., *et al.*, 2011. Capo: recapitulating storage for virtual desktops. FAST, p.31-45.

Smith, J.E., Nair, R., 2005. The architecture of virtual machines. *Computer*, **38**(5):32-38. http://dx.doi.org/10.1109/MC.2005.173

Wartel, R., Cass, T., Moreira, B., *et al.*, 2010. Image distribution mechanisms in large scale cloud providers. Proc. 2nd Int. Conf. on Cloud Computing Technology and Science, p.112-117. http://dx.doi.org/10.1109/CloudCom.2010.73

Weil, S.A., Brandt, S.A., Miller, E.L., *et al.*, 2006. Ceph: a scalable, high-performance distributed file system. Proc. 7th Symp. on Operating Systems Design and Implementation, p.307-320.

Zhang, Y., Liu, L., 2012. Distributed line graphs: a universal technique for designing DHTs based on arbitrary regular graphs. *IEEE Trans. Knowl. Data Eng.*, **24**(9):1556-1569. http://dx.doi.org/10.1109/TKDE.2011.258

Zhang, Y., Chen, L., Lu, X., *et al.*, 2010. Enabling routing control in a DHT. *IEEE J. Sel. Areas Commun.*, **28**(1):28-38. http://dx.doi.org/10.1109/JSAC.2010.100104

Zhang, Y., Guo, C., Li, D., *et al.*, 2015. CubicRing: enabling one-hop failure detection and recovery for distributed in-memory storage systems. Proc. 12th USENIX Symp. on Networked Systems Design and Implementation, p.529-542.

Zhang, Z., Li, Z., Wu, K., *et al.*, 2014. VMThunder: fast provisioning of large-scale virtual machine clusters. *IEEE Trans. Parall. Distr. Syst.*, **25**(12):3328-3338. http://dx.doi.org/10.1109/TPDS.2014.7

Zhao, Y., Wu, J., Liu, C., 2014. On peer-assisted data dissemination in data center networks: analysis and implementation. *Tsinghua Sci. Technol.*, **19**(1):51-64. http://dx.doi.org/10.1109/TST.2014.6733208

Yi-ming ZHANG, corresponding author of this paper, received the China Computer Federation (CCF) Distinguished PhD Dissertation Award in 2011. He is currently an associate professor in the National Laboratory for Parallel and Distributed Processing, School of Computer, National University of Defense Technology (NUDT), China. His current research interests include cloud computing and operating systems.

Xi-cheng LU, Editor-in-Chief of *Frontiers of Information Technology & Electronic Engineering*, received the B.Sc. degree in computer science from Harbin Military Engineering Institute, Harbin, China, in 1970. He was a visiting scholar at the University of Massachusetts between 1982 and 1984. He is currently a professor in the School of Computer, National University of Defense Technology, China. His research interests include distributed computing, computer networks, and parallel computing. He has served as a member of editorial boards of several journals and has cochaired many professional conferences. He is a joint recipient of more than a dozen academic awards, including four First-Class National Scientific and Technological Progress Prize of China. He is a member of the Chinese Academy of Engineering.