Front. Comput. Sci.
DOI 10.1007/s11704-012-1175-2

RESEARCH ARTICLE

Effectively deploying services on virtualization infrastructure

Wei GAO, Hai JIN, Song WU (✉), Xuanhua SHI, Jinyan YUAN

School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China

Abstract Virtualization technology provides an opportunity to acieve efficient usage of computing resources. However, the management of services on virtualization infrastructure is still in the preliminary stage. Contstructing user service environments quickly and efficiently remains a challenge. This paper presents a service oriented multiple-VM deployment system (SO-MVDS) for creating and configuring virtual appliances running services on-demand. The system provides a template management model where all the virtual machines are created based on the templates with the software environment pre-prepared. To improve the deployment performance, we explore some strategies for incremental mechanisms and deployment. We also design a service deployment mechanism to dynamically and automatically deploy multiple services within virtual appliances. We evaluate both the deployment time and I/O performance using the proposed incremental mechanism. The experimental results show that the incremental mechanism outperforms the clone one.

Keywords multi-VM deployment, template, incremental mechanism, batch deployment

1 Introduction

With the expanding scale of resources and processing capacities of computing systems, administrators face challenges in using and managing these resources flexibly and effectively. Over the last twelve years, the rapid development of virtualization technology has provided a good opportunity for resolving these issues. Virtualization [1–3] is a technology capable of decoupling the underlying physical device, the operating system and the software. Modern virtualization platforms, such as Barham, et al. [4] and Sugerman, et al. [5] are able to divide physical resources into isolated slices with appropriate sizes for different applications. Consequently, different types of job can be run on the same nodes with sufficient resources and without interfering with each other. This encourages virtual machines to be introduced as service containers for improving resource utilization of the service computing system.

In a virtualization infrastructure, resources such as CPU, memory and I/O are organized as virtual machines (VMs). When deploying a service, the system allocates proper resources into a new VM, then the service with its execution environment can be installed and configured into the VM, this VM running the service is managed as a virtual appliance [6].

However, with the increasing number of users and their requirements, deploying and managing services based on a virtualization infrastructure is a remarkable task. Tens or even hundreds of virtual appliances will run at the same time. Since manually creating VMs, installing software onto them and configuring the environment is time-consuming and error-prone, a system that automatically deploys virtual appliance based services according to given user requests is necessary.

In this paper we propose a service oriented multi-VM fast deployment system (SO-MVDS) which allows administrators to design and create virtual machines with specific services running in them. This system works with a template-based mechanism, and employs an automatic deployment strategy to improve the deployment efficiency. Administrators only need to choose the service template and a few optional deployment parameters, and then the system will deploy the corresponding VMs in a few minutes. The sys-

Received March 26, 2012; accepted June 11, 2012

E-mail: wusong@hust.edu.cn

tem also maintains a template repository, and provides approaches for optimizing the preparation of service deployment.

Section 2 presents an overview of SO-MVDS. Sections 3 to 5 give the designation in detail. We present quantitative evaluation in Section 6. Section 7 describes related work and discusses the difference to our work. Finally in Section 8, we conclude the paper.

2 Architecture overview

This section provides an overview of SO-MVDS. Specifics of its implementation are detailed in Section 3. We begin with the consideration of the mechanism, followed by the high-level design issues that have driven our work. The final part of this section presents the design architecture of the system.

2.1 Consideration

Users of our system are service developers or service providers who need to deploy their services onto the virtualization infrastructure. Most service deployment tasks require one or more groups of VMs. Each group has similar content from a template. Figure 1 illustrates a scenario of task deployment. In the case of web application deployment, the task includes several services which can be grouped according to their different requirements. Each group needs the same software environment. To meet this demand, the administrator will choose the proper environment for each service and specify needed resources. Then the system automatically creates VMs on computing nodes with enough resources, deploys services into the VMs that have the right environment, and returns the execution results.

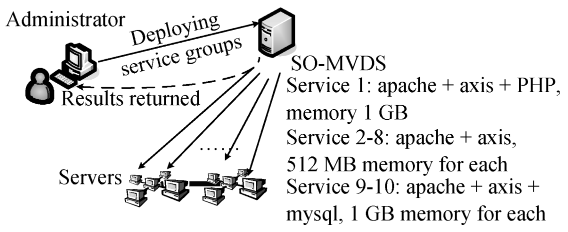
Administrator
Deploying
service groups
SO-MVDS
Results returned
Service 1: apache + axis + PHP, memory 1 GB
Service 2-8: apache + axis, 512 MB memory for each
Service 9-10: apache + axis + mysql, 1 GB memory for each
Servers

Fig. 1 A service oriented VM deployment scenario

SO-MVDS is based on the Xen virtual platform [4], and proposes a template based virtualized service deployment mechanism. The goal of this system is to construct task environments for users in a short time and automatically provide VMs to services on demand. To support fast deployment of multiple service groups, SO-MVDS follows several design requirements:

- **Template-Based** Like other VM management tools, our system also uses a template-based mechanism. There are two kinds of templates in our system: the basic template and the incremental template. A template is an image file, and all templates are created by administrators. Basic templates hold the common operating systems, and incremental templates hold frequently-used software packages based on the operating system. A basic template may have multi-level incremental templates. All templates are located in the template repository. The basic templates are shared by nodes through network file systems (NFS), which provides convenience for administrators to manage the templates. The templates in the repository are static. After deployment they will become running VMs.
- **Flexibility** SO-MVDS maintains a software environment repository, and allows administrators to choose the environment templates they require. It offers a friendly graphical interface. Users can customize virtual appliances' physical resource allocation, such as memory and CPU, and choose the deployment strategy. The system also supports the recall of multiple services that stop and save the active services from the system. Once a service is deployed, the system will create a specific virtual appliance. When this service is recalled, the image of its virtual appliance will be saved as a new template, so next time the service can be easily re-deployed by starting the VM based on this template.
- **Transparency** In this system the entire deployment process is transparent to users. After users submit the services with related parameters, the system can automatically create VMs and deploy services in a few minutes. They need not be concerned with how VMs are created on physical machines and how services are deployed into the VMs. We make the virtualization infrastructure look like a normal service container, users deploy the services onto it even do not feel the service is running in a VM. Some strategies are provided to support automatic deployment of services.

2.2 Architecture

From a hierarchical viewpoint, SO-MVDS can be divided into three layers as shown in Fig. 2. They are user interface layer, deployment strategy service layer, and deployment control service layer. In this layered architecture, each layer has its own services and interfaces invoked by its upper layer. The layers are flexible and independent of each other. This makes the system easy for implementation and maintenance.

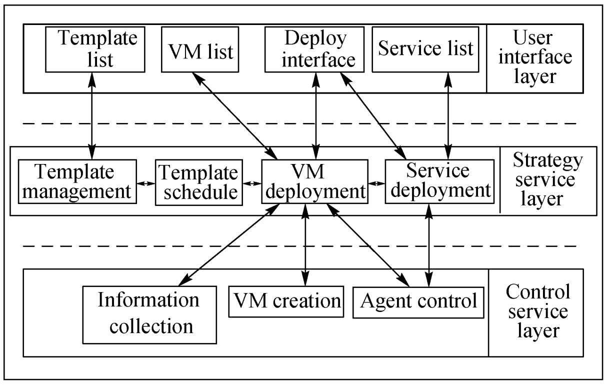

Fig. 2   Architecture of SO-MVDS

2.2.1   User interface

To make this system easy to use, a friendly user interface is necessary. It should be easy for users to customize and create virtual machines. Users can find the organization of the templates, the running status of the virtual machines and the execution process of services from the user interface. Certainly, only authorized users (e.g., administrators) are allowed to use it. It presents five functional options:

- template list: shows detailed information of the templates,
- deploy: edit some parameters and perform service deployment,
- recall: remove some deployed services,
- VM list: list the VM information on each node,
- service list: give the information of all maintained services.

2.2.2   Strategy service

In order to deploy services as fast as possible, it is desirable to manage all the templates in a uniform way. Moreover, a user probably wants to re-deploy a service more than once. So, the deployed services must be properly managed. Deployment strategy is another factor to affect the deployment efficiency, making multiple services batches to be deployed automatically in a short time.

The deployment strategy service layer serves as a middleware between the user interface and control service layer. It is usually located in an independent server and responds to the operations of the user interface layer. The main services provided by this layer include template management, template scheduling, VM deployment, and service deployment services. Template management is responsible for the definition and storage of the templates, as well as the relationship between the templates and the users. Template scheduling refers to template preparation and disposal strategies. VM deployment service is responsible for the strategies and technologies for VM deployment such as incremental and concurrent mechanisms. Service deployment service helps users to install services into the created VM and manage deployed services. The specific functions are provided by the control service.

2.2.3   Control service

The deployment control service layer lies on the computing nodes of the virtualization infrastructure. It receives instructions from the deployment strategy service layer and carries out the actual deployment process. There are three modules presented to control different objects. Firstly, physical resources of each node should be organized to help build a VM. It is performed by the information collection module. Then, the VM creation module controls the VM Monitor to start up a VM on one of the nodes. Finally, through the agent program running in the VM, the service is installed and activated by the agent control module. Here the agent is pre-installed into the VM and exists in the templates.

3   Template management

A template is a disk image with a pre-installed operating system with or without certain application software. Users can choose proper templates according to their requirements. We put all the templates into a repository and maintain them in a template database. As the number of applications, increase the number of templates increases sharply. We will next present some approaches to solve the problems of template management.

3.1   Incremental mechanism

For each raw image or template, it takes several gigabytes to hold the operating system and the applications. For example, Red Hat Enterprise Linux 5 needs 2.5–3.5 GB disk space and Windows Server 2003 may occupy over 2 GB disk space. Moreover, application software generally needs disk space from tens to hundreds of megabytes, and we should preserve another several gigabytes for user data. Therefore, a complete VM image probably has a minimum size of at least 5 GB.

The incremental mechanism uses the copy on write (COW) idea. We choose some most popular templates such as OS images as basic image files, and then backup these files in a COW format as incremental image files. The application software can then be installed into the incremental image file, for sharing data in the basic image. An image file will be set to read-only authority if it is a basic image. All the write operations take effect only on incremental image files. Note that an incremental image file can be the basic file of another image so as to share disk data as much as possible. Each incremental image can be run as a VM. Figure 3 presents the structure of multi-level incremental image tree. Templates are shared as root or branch root. When users pick a template to start a virtual machine, the system creates a new incremental image of the template as the VM's disk image which is the leaf of the template tree. The incremental image file is very small, and it is easy to create and transfer. Consequently, it reducesspace consumption, and the time of VM deployment is much less than cloning the whole raw image file.

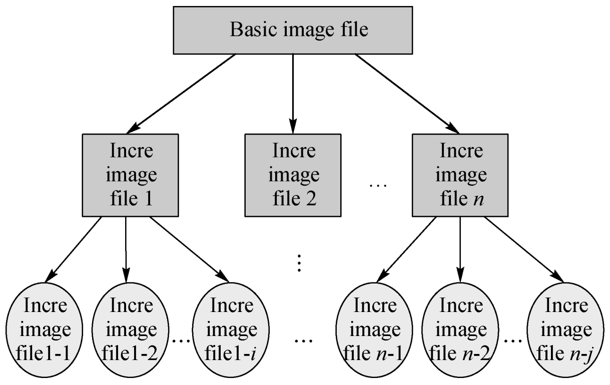

Fig. 3 Multi-level incremental image file structure

3.2 Template sharing model

Because of users' different requirements and the variety of software, it is important to organize templates. The problem is how to describe the templates exactly so that a user can easily find them according to his requirements. We introduce a user specifying mechanism and propose a template sharing model (TSM) which allows VM users to maintain the index of templates. There are three entities in this model: template, label, and user. Template is a template image files whose metadata includes the identity and other properties of the template. Label is the character string defined by user and it provides a semantic description of the template; in other words, a label is a personal classification, which represents a user's interest to a group of templates. User is the operation subject which fetches templates belonging to his/her interested Labels, browses information of those templates, and picks a favorite one.

The relationships between user, template and label are shown in Fig. 4. Users create the templates and labels, and then assign the labels to templates. Each user has different interests, so may pay attention to different labels. Users with similar interests may exchange shared templates. In Fig. 4, user means administrator or other authorized user.

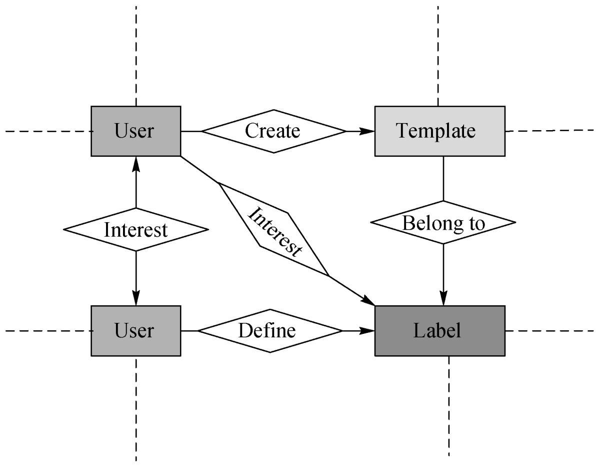

Fig. 4 Relationships of entities in TSM

TSM provides a flexible template indexing and fetching mechanism that helps users to match their needs to virtual computing environments. The labels assigned to a template can be resource capacities (CPU speed, memory capacity, etc.), software environments (OS version, network setup, program configuration, etc.) and/or other relative specifications of the template. Each template is connected to many labels, and accessible by each label assigned to it. A user can browse templates in the repository and filter out irrelevant ones by labels. In this way, users can easily find a group of templates that satisfy his requirements, and pick one from a limited number of candidates of their own interest.

3.3 Version and lifecycle control

As more and more new versions of software are released, users often want to use updated versions or work in a steady environment. So, we should provide not only different applications' templates, but also different versions of them. For users who want a specific version of application, each version of that application is an individual template. They can choose what they want. For those who are not sensitive to software versions, we consider all templates with different versions as a series and choose one of them according to a strategy. The strategy we implement is to choose the most recently used template. Otherwise, the one with the newest version will be chosen.

To reduce deployment preparation time and avoid the over stored templates, two data lifecycle mechanisms should be imposed by the TSM model.

Template pre-copying means that the TSM creates many copies of hot templates and sends them to the proper server nodes in order to enhance efficiency. The number of copies of each template is determined by its practical usage frequency. The higher a template's popularity or importance degree, the more copies are distributed across the physical network.

Template disposal will be executed when there are low-value templates (or their versions) which have not been activated by any user after a specific cool down time. These cold templates, including their copies, should be discarded for clearing garbage templates and reclaiming storage space

4 VM deployment

The process of deploying a service onto virtualization infrastructure has two main steps: VM deployment and service deployment. When a user submits a service deployment task, VMs designed to run the services are created, and then each service is installed into its host VM. This section presents the VM deployment details in SO-MVDS and the strategy we employ to spread VMs across physical nodes. Service deployment is discussed in Section 5.

4.1 VM creation

Figure 5 illustrates how VMs are created on physical machines. Though users can appoint certain nodes to deploy virtual appliances on, most users do not care which hosts are running their services. According to the physical resources parameters given from user, the system should find a computing node with sufficient resources to support the VM running. SO-MVDS collects resource usage details of every computing node using the information collection module, and then matches VMs to nodes for deployment based on the VM distribution strategy. The result of the VM matching is a VM deployment sequence, which records a list of VMs and their host node pairs. Following the deployment sequence, the VM deployment module sends every VM's parameters to its host node, and informs the template schedule module to pre-copy the templates chosen by user. Then the VM creation module located on each node will respectively launch the assigned VM.

Each VM maps to an image and a configuration file, they need to be prepared well before the VM is created. The image file contains the VM's disk image which is incrementally generated from a specified template. A VM will boot from the operating system installed in the image file. To speed up the VM boot time, the VM's memory data can also be saved into the image file, or a run-time checkpoint, and then the VM can load the memory and directly come into a running state. The configuration file is automatically made in XML format by the VM creation module. It records necessary information such as VM name, OS kernel, memory, CPU, and network setup. Some parameters are open to the user. Users can edit these configurations as the service's requirement, or leave it to the default setup. The other parameters are maintained by the system, including the address of the image file. Finally the configuration file is sent to the VM Monitor to start a VM.

It is reasonable that the system concurrently deploys VMs to reduce the total deployment time. Yet the efficiency of it meets the bottleneck for Xen, the virtualization platform we build our system on must create VMs one by one for manageable and security considerations. So SO-MVDS just simultaneously executes the deployments on different computing

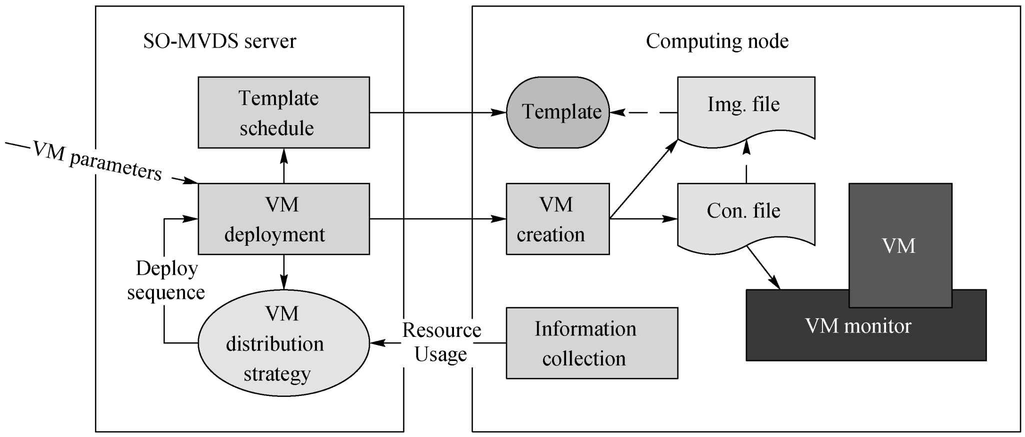

Fig. 5 Process of VM creation

nodes. Suppose a VM deployment task needs to launch VMs on N hosts and n_j is the number of VMs needed to start on the jth node. Then the total time spent on deploying VMs can be described by the following equation:

$$T_{\mathrm{VM}} = T_{\mathrm{c}} + T_{\mathrm{s}} + \max\left(t \middle| t = \sum_{i=1}^{n_j} t_{ij}, j \in N \right) \tag{1}$$

where T_{c} is the time for calculating resource usage, T_{s} is the time for process VM distribution strategy, and t_{ij} is the time for deploying the ith VM on node j. Since the total time is relative to the final node that finishes VM deployments, spreading VMs across more nodes, thus reducing n_j, is a considerable way to improve efficiency.

4.2 VM distribution strategy

We design a strategy to choose a proper destination host for the virtual appliance. The principle is to meet each user's requirements as well as to balance workloads across the whole host network. The strategy is described as the algorithms as follows.

Before describing the algorithms, we illustrate some symbols and variables. P is a set of VM requirements, and each element of P contains the attributes of a VM such as name, CPU and memory. H is a set of host resource information. The information of each host includes the CPU used, memory resources, and so on. $D$is the result set containing all the VM deployment sequences. N is the required deployment number. T_{CPU} and T_{Mem} are the marginal values of the host which ranges between 0.7 and 0.8. Variable d is the result returned from the ChooseDestination algorithm and it represents the host id.

Algorithm 1 presents the basic GetDeploySequence algorithm which consists of two phases. The first phase (lines 1–7) computes the free CPU and memory of each host. In the second phase (lines 8–16), we invoke the ChooseDestination algorithm which returns the destination host id for each VM deployment. Then we add the VM and host id to set D. Finally we return the ultimate collection Dwhich contains the entire deployment sequence at line 17.

Algorithm 2 describes the procedure of finding the proper host for each VM. It is designed to be called by Algorithm 1. This algorithm considers both memory size and CPU usage as resources that hosts should have to be selected for running the VM. First, this algorithm descends the set H by order of the freeMem and freeCPU. By choosing the host with the most free resources first, the VM will take priority of being deployed onto a host with a light load. Then it compares repeatedly the VM's required CPU usage (V.CPU) and memory size (V.Mem) with every host's freeCPU and freeMem until a host with enough resources is found. Finally the host id (or -1 if no host fulfills the requirement) is returned.

Algorithm 1: GetDeploySecquenc (P, N, H)

```
1:    D= {}, i = 0
2:    while i < number of hosts ∈ H do
3:       get used CPU and memory of H(i)
4:    H(i). freeCPU = total CPU*Tcpu –usedCPU
5:    H(i). freeMem = total Mem*Tmem–usedMem
6:    i = i + 1
7: end while
8:    n = 0
9: while n < N do
10:    d = ChooseDestination(P(n), H)
11: if d > −1 then
12:    add P(n).name, H(d) to D
13: else return D
14: end if
15:    n = n + 1
16: end while
17:    return D
```

Algorithm 2: ChooseDestination (V, H)

```
1: Sort H by order of freeMem and freeCPU
2: i = 0
3: while H(i).freeMem > V.Mem do
4: if H(i).freeCPU > V.CPU then
5: H(i).freeCPU = H(i).freeCPU − V.CPU
6: H(i).freeMem = H(i).freeMem − V.mem
7:    return i
8:    else
9:       i = i + 1
10:    end if
11: end while
12:    return−1
```

5 Service deployment

Though the service can be manually installed into a dedicated VM once it is created, an automatic approach is preferred for enhancing the efficiency of service deployment. In this section we propose the designation of a service deployment mechanism over VMs.

The main goal of the service deployment mechanism is to encapsulate the operations toward VMs and physical machines, for users deploying their services onto a virtualization infrastructure and thus make it as easy as using a physical machine. Users need only submit service packages with optional parameters, and then get the addresses of deployed services as results. We define a service deployment request as a set D

= (S, C, R, P) where S is the package of service execution files, C is the VM template with the container that the service is deploying in, R is the service's requirements of physical resources, and P records other needed parameters. We also define a service group deployment task as a collection of D: TASK = $\{D_1, D_2, \cdots, D_n\}$. One of the benefits that the virtualization technology brings to the service computing system is, because the VMs isolate the execution environments of services from each other, many different service platforms or containers can be supported on a virtualization infrastructure. SO-MVDS is designed to have the ability to deploy services onto different types of containers.

As Fig. 6 shows, when a task is submitted to SO-MVDS, there are three sub-modules working in the service deployment module.

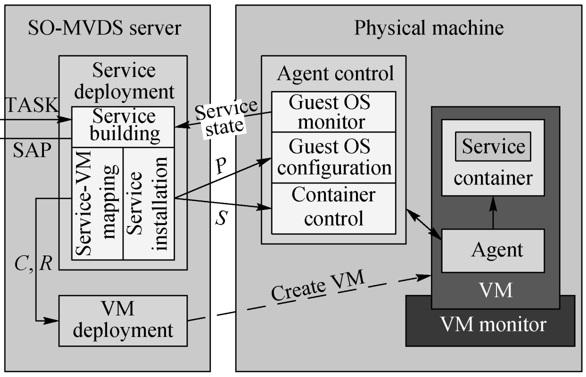

Fig. 6 Process of service deployment

- **Service Building** This operation analyzes every request set D in the deployment task, and controls the service deploying process based on the information gathered from agents. Towards different types of services, each type needs a particular builder to generate the access point of the deploying service (SAP). Normally, a service can be invoked from a URL that is composed of three parts: IP address of host server, service container name and service name. The service's IP address is the VM's IP assigned by the user or chosen from the IP pool that is maintained by the system. The container name can be extracted from the template's labels, and some containers may need to appoint the network port number and IP address. The service name exists in its description file that needs to be parsed. Users also can put the service name in the parameters for convenience.
- **Service-VM Mapping** Because a service is running in a dedicated VM, it is important to record the bounding information of the service and its host VM. Here the system assigns a UUID to every VM, and records these along with the service information into a service-VM mapping table. At the same time, this sub-module picks the container template C and resource requirements R of every service into a list and sends these parameters to the VM Deployment Module to deploy VMs. If the VMs have been created and are running, the deployment module can find a service's mapping VM by comparing the VM's UUID through via agents.
- **Service Installation** Once VMs are launched, this submodule communicates all agent control modules and locates the VM mapping to every service. The agent senses and examines the running state of the service along with its environment and reports to the service deployment module. For arranging the deployment operations, we classify the service state as (1) VM unavailable (no agent response); (2) VM ready; (3) container ready; and (4) service available. The guest OS configuration will not start until after it receives the VM ready state from the agent. Before the service is installed into the VM, its relative parameters in P are sent to the Agent to configure the guest OS of the host VM, for example, network setup and registration of environment variables. When the container is ready, the service package S is transferred to the VM and deployed by the container. When the service deployment task is finished, a user can invokes services through the SAPs returned by the system.

6 Evaluation

In this section we describe experiments for quantitative evaluation of the system. We perform two sets of experiments. First, we evaluate the VM deployment time with different strategies. Then we evaluate the I/O performance of the VM with incremental image file in different environments. Deployment time is the total time from the point when the deployment is started by the request, to the point when the VM is running on the server.

6.1 Experimental setup

All experiments are performed on a cluster of 9 computers each equipped with an Intel Xeon 4 core 2.33 GHz CPU. The front-end node in the cluster is has 4 GB of memory and 160 GB storage. The remaining 8 nodes have 8 GB of memory and 30 GB storage each. The cluster system is in the VM-based environment and Xen-3.1.0 with the 2.6.16 kernel is used on all computing nodes. The Xen domain0 hosts Red

Hat Enterprise 5 with 2 GB memory. All user domains are run a single virtual CPU with 512 MB memory. The guest OS in DomU which is used to evaluate the deployment time uses Windows XP and RedHat9 templates. The application environments include Windows 2003 Server and Fedora Core 7. All basic templates are shared via NFS.

6.2 Deployment time evaluation

In this section, we compare the VM deployment time of our incremental method with that of the traditional clone method. Tests are conducted on eight different nodes. We use a Windows XP template with a size of 1.5 GB and a RedHat9 template with a size of 2.3 GB, and all VMs are created using these as a basis. The clone strategy clones the template each time the system deploys a new VM. In contrast, under the incremental strategy, we create a new incremental image file based on the template we choose each time. Each incremental image file has an index pointing to the basic file and it only records the updated data, so it is small and can be created very fast.

Figure 7 compares the deployment time with varying VM numbers under different strategies. We find there is a large difference in deployment time between the clone and the incremental strategies. As the number of VMs grows, the deployment time with clone strategy increases rapidly, but the deployment time with incremental strategy increases much more slowly. Figure 7 shows a reduction in deployment time of up to 80% or 90% using our incremental strategy when ten VMs are deployed, which is an encouraging improvement. The reason is that our basic templates located in the cluster NFS can be accessed by each node and the incremental method only needs one basic template each time. Unlike the clone method, it need not copy the basic template, which is the main reason for the reduction in deployment time.

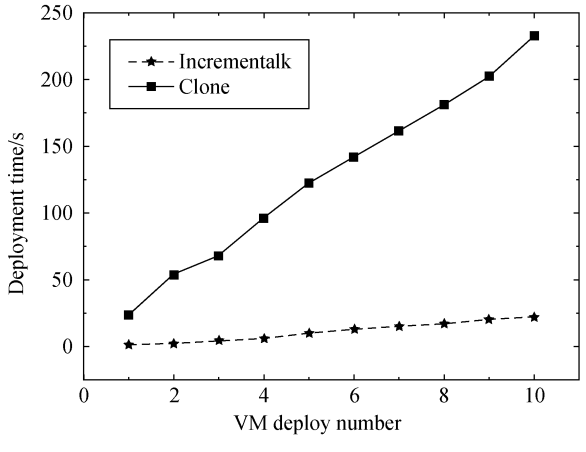

Fig. 7 Deployment time of different strategies

Figure 7 convinces us that the deployment time of the clone strategy is proportional to image size. It will increase obviously as the image size grows. Testing different templates of Windows XP and RedHat9 we find that the deployment time is very different between these two templates. The growth of the deployment time based on RedHat9 is about 40%. Therefore, the image size is the decisive factor of the deployment time with the clone one.

Intuitively, deployment time will increase with a larger template even using the incremental strategy. However, Fig. 8 tells us that there is no direct relationship between deployment time and image size by using the incremental strategy in our system. We can see that the deployment time based on the RedHat9 template is slightly less than the Windows XP template, while the size of RedHat9 template is larger than the Windows XP template. Therefore, our system can deploy various applications appropriately with nearly the same time no matter how large the basic template is.

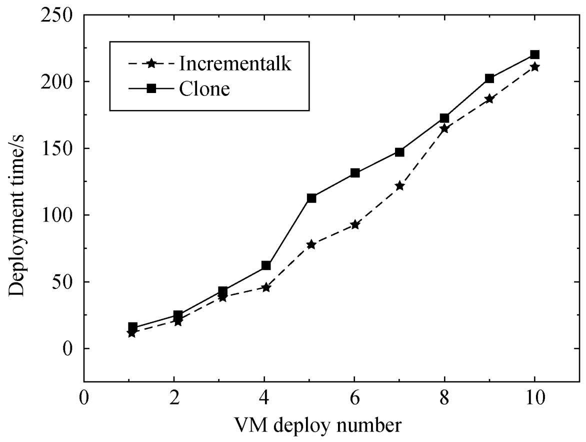

Fig. 8 Deployment time of different templates with incremental mechanism

In order to show the performance of batch deployment, we conduct a set of experiments on the Windows XP template. Figure 9 presents the deployment time for increasing numbers of VMs. We see that the deployment time increases in a *n* approximately linear fashion, about 11 seconds for five VMs, rising to about 88 seconds for 40 VMs. The average deployment time of each VM is about 2.5 seconds which is deemed acceptable for users.

Sometimes administrators deploy many VMs for the tasks with a short lifespan. After they finish the tasks, they should destroy the VMs. To destroy the VMs manually one by one is an arduous job. Our system provides the recall function and a batch operation is offered, so it is convenient for the administrators to destroy the VMs according to their requests.

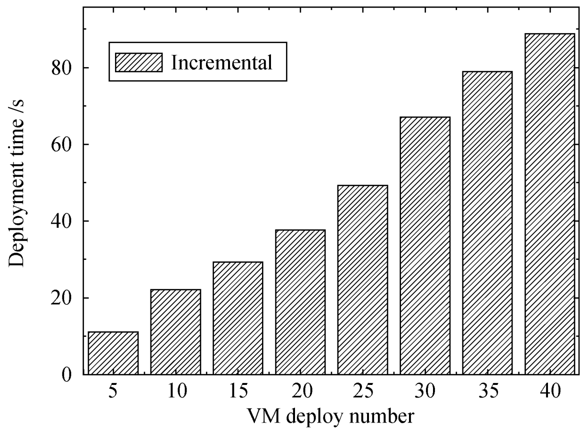

Fig. 9 Batch deployment time with our incremental mechanism

We also measure the deployment influence on the node's performance. The deployment process consumes limited resources. The average CPU usage is about 5% and the memory usage is scarcely. The deployment performs well as a whole.

The concurrent deployment with incremental mechanism promises good scalability for our system, as well as low cost. Equation (**??**) suggests that the deployments should faster on more physical nodes in the data center. Though the costs of template management and deployment strategy may increase slightly as the number of hosts grows, the overall performance can be ensured by organizing those hosts into many second level groups for VM distribution strategy and template replication.

6.3 I/O evaluation

We have created a few application templates stored in our template repository. All applications are installed as incremental images from the basic templates of Windows 2003 Server, Windows XP, and Fedora Core 7. These incremental image files are stored on NFS or native disk. In this section we compare I/O performance of the incremental image file with NFS and local file store schemes, using the basic image file RedHat 9 stored in NFS system. We deploy the same application respectively in the two incremental image files.

Figure 10 shows the file copy speed with different mechanisms. This experiment copies a file existing in the basic image written to an incremental image. We can see that the file copy speed of local scheme is much faster than NFS. The average copy speed of local vs NFS scheme is about 480 Mbps vs 199 Mbps, respectively. That is because of the communication speed limit of NFS. The VM reads the file from the basic image file stored on NFS, and then the data is sent back to the incremental image file also located on NFS. The situation forms an infinite access loop, which reduces the I/O performance. So, the I/O performance of local scheme is much better than the NFS scheme. That is the reason why we take the native scheme.

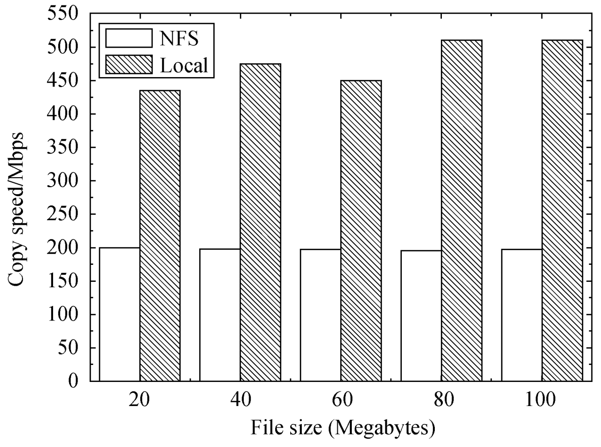

Fig. 10 File copy speed

We also conduct experiments on VMs based on raw image files, which are used in other VM deployment systems such as Xenfire [7] and virtual workspace [8]. The average file copying speed is 476.2 Mbps, very close to the incremental images in native disk.

7 Related work

Many tools have been developed to perform VM deployment. Xenfire [7] is a lightweight Xen-based software which can install VMs on a single physical server. Begnum, et al. [9,10] use a configuration language to describe the design of a group of VMs, which means that users have to specify each VM's detailed resource configuration and its host node. Collective [11,12] project uses copy-on-write (COW) disks as template organized in a hierarchy tree [13], in order to reduce storage space and data to transfer. But this project has no mechanism to automatically deploy VMs on nodes with sufficient resources. Similar work has been applied to virtual cluster systems, Krsul, et al. [14], creates a VM by cloning a predefined template and then applying other setups to the cloned VM. Another proposal for virtual clusters [15] proposes a mechanism to allocate the VM physical computing resources requested by the user. One shortcoming of virtual cluster deployment is that after the creation of a VM, they need cluster installers [16,17] to help finish software installation and configuration [18]. Similar to our incremental mechanism, Lagar-Cavilla, et al. [19] proposes a VM clone technique to deploy VMs by reusing data from an already deployed VM, and only new data of the new VM are transferred. In

[20], the data of the image file is sent to the VM on demand, which reduces the VM deployment time as well as incremental mechanism. Other recent works [21,22] use P2P networks to distribute and transfer the pieces of an image file. Such approaches improve the efficiency of VM deployment on very large datacenters, though the network cost of image indexing will be increased.

The benefits of using VM to deploy service are discussed in [6]. Globus virtual workspace services [8] provide functions and protocols to dynamically deploy and manage execution environments. Globus can design a group of services with their physical resources and software configurations and implement them in VMs. Kecskemeti et al. [23] extend the workspace service with two new services: AVS creates virtual appliances for grid services and stores them in a repository; SAS helps to define and select a service instalation site, and deploy virtual appliances from the repository. The main difference between the above work and our system is that, the workspace services and AVS/SAS expose the VM to the user, the user must create the VM before submitting the service, while SO-MVDS attempts to hide VM operations from the service provider. We also introduce mechanisms such as incremental templates and our VM distribution strategy to improve the efficiency of service deployment. As the popularity of cloud computing grows, deploying VM based services in a cloud environment has been studied. Jump-start cloud [24] proposed a VM cloning based image distribution and sharing scheme for saving the deployment time of cloud services. Csorba, et al. [25] maintains replicas of VM images on different nodes in the cluster, and uses a scalable algorithm to optimize service-node mapping. Sampaio and Mendonca [26] employs open standards to perform VM and service deployment over multiple heterogeneous cloud platforms. SO-MVDS distinguishes itself from these cloud service deployment approaches by its label based incremental template sharing model. Our TSM maintains the templates and their indexes according to user interest, while other solutions assume that users already know the identities of templates required by their services.

8 Conclusions

This paper presents SO-MVDS, a service oriented Multi-VM deployment architecture for deploying users' service environment on-demand. In order to improve the deployment performance, we design a template management model which helps the administrator to manage the templates conveniently, as well as reduce space consumption. The incremental mechanism significantly reduces deployment time, as demonstrated by our experiments. The system can automatically and transparently deploy services into VMs in a short time.

In our future work we will increase the transparency and efficiency of SO-MVDS. A virtualized service container will be developed as an access interface level that transforms the service deployment task across VMs into a scenario of deploying services into a typical service container. A service replication mechanism that quickly clones many replicas of a service based on virtualization infrastructure is also planned.

Acknowledgements This work is supported by National Natural Science Foundation of China (Grant Nos. 61073024, 60973037, and 61133008), and by the Outstanding Youth Foundation of Hubei Province (2011CDA086), and by the research fund for the Doctoral Program of MOE (20110142130005).

References

1. Smith J, Nair R. Virtual machines: versatile platforms for systems and processes. San Francisco: Morgan Kaufmann Publishers, 2005
2. Wolf C, Halter E M. Virtualization from the desktop to the enterprise. APress, 2005
3. Goldberg R P. Architecture of virtual machines. Massachusetts: Honeywell Information Systems, Massachusetts and Harvard University Cambridge, 1973, 32–38
4. Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho R N A, Pratt I, Warfield A. Xen and the art of virtualization. In: Proceedings of ACM Symposium on Operating Systems Principles. 2003, 164–177
5. Sugerman J, Venkitachalam G, Lim B. Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor. In: Proceedings of the 2001 USENIX Annual Technical Conference. 2001
6. Sun C, He L, Wang Q, Willenborg R. Simplifying service deployment with virtual appliances. In: Proceedings of 2008 IEEE International Conference on Services Computing. 2008, 265–272
7. Xenfire project. http://developer.novell.com/wiki/index.php/Xenfire
8. Keahey K, Foster I, Freeman T, Zhang X. Virtual workspaces: achieving quality of service and quality of life in the grid. Scientific Programming, 2005, 13(4): 265–275
9. Begnum K M, Sechrest J. The MLN manual mln version 0.80. http://mln.sourceforge.net/doc/mln-manual.html
10. Begnum K M. Managing large networks of virtual machines. In: Proceedings of the 20th USENIX Large Installation System Administration Conference. 2006, 205–214
11. Sapuntzakis C, Brumley D, Chandra R, Zeldovich N, Chow J, Lam M S, Rosenblum M. Virtual appliances for deploying and maintaining software. In: Proceedings of the Seventh USENIX Large Installation System Administration Conference. 2003, 181–194
12. Chandra R, Zeldovich N, Sapuntzakis C, Lam M S. The collective: a cache-based system management architecture. In: Proceedings of the Second Symposium on Networked Systems Design and Implementation. 2005, 259–272
13. Sapuntzakis C P, Chandra R, Pfaff B, Chow J, Lam M S, Rosenblum M. Optimizing the migration of virtual computers. In: Proceedings of the Fifth Symposium on Operating Systems Design and Implementation. 2002, 337–390

14. Krsul I, Ganguly A, Zhang J, Fortes J A B, Figueiredo R J. Vmplants: providing and managing virtual machine execution environments for grid computing. In: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing. 2004, 7–18
15. Foster I, Freeman T, Keahy K, Scheftner D, Sotomayer B, Zhang X. Virtual clusters for grid communities. In: Proceeding of the 2006 International Conference on Cluster Computing and Grid. 2006, 513–520
16. Papadopoulos P M, Katz M J, Bruno G. Npacirocks: Tools and techniques for easily deploying manageable linux clusters. In: Proceedings of the International Conference on Cluster Computing. 2001
17. Takamiya Y. Large-scale configuration management and installation of commodity clusters. PhD thesis, Tokyo: Tokyo Institute of Technology, 2006
18. Nishimura H, Maruyama N, Matsuoka S. Virtual clusters on the fly-fast, scalable, and flexible installation. In: Proceeding of the 2007 International Conference on Cluster Computing and Grid. 2007: 549-556
19. Lagar-Cavilla H A, Whitney J A, Scannell A M, Patchin P, Rumble S M, de Lara E, Brudno M, Satyanarayanan M. SnowFlock: rapid virtual machine cloning for cloud computing. In: Proceedings of the 4th ACM European conference on Computer systems. 2009, 1–12
20. Nicolae B, Bresnahan J, Keahey K, Antoniu G. Going back and forth: efficient multideployment and multisnapshotting on clouds. In: Proceedings of the 20th International Symposium on High Performance Distributed Computing. 2011, 147–158
21. Schmidt M, Fallenbeck N, Smith M, Freisleben B. Efficient distribution of virtual machines for cloud computing. In: Proceedings of the 18th Euromicro Conference on Parallel, Distributed, and Network-Based Processing. 2010, 567–574
22. Wartel R, Cass T, Moreira B, Roche E, Guijarro M, Goasguen S, Schwickerath U. Image distribution mechanisms in large scale cloud providers. In: Proceedings of the 2nd IEEE International Conference on Cloud Computing Technology and Science. 2010, 112–117
23. Kecskemeti G, Kacsuk P, Terstyanszky G, Kiss T, Delaitre T. Automatic service deployment using virtualization. In: Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing. 2008, 628–635
24. Wu X, Shen Z, Wu R, Lin Y. Jump-start cloud: efficient deployment framework for large-scale cloud applications. In: Proceedings of the 7th international conference on Distributed computing and internet technology. 2011, 112–125
25. Csorba M J, Meling H, Heegaard P E. Ant system for service deployment in private and public clouds. In: Proceeding of the 2nd Workshop on Bio-inspired Algorithms for Distributed Systems. 2010, 19–28
26. Sampaio A, Mendonca N. Uni4cloud: an approach based on open standards for deployment and management of multi-cloud applications. In: Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing. 2011, 15–21

Wei Gao is a PhD candidate in the School of Computer Science and Technology at Huazhong University of Science and Technology (HUST). His research interests include computing system virtualization, distributed parallel computing, and cloud computing.

Hai Jin received his PhD in computer engineering from HUST in 1994. Currently, he is a Cheung Kung Scholars Chair Professor of Computer Science and Engineering at HUST. His research interests include cluster computing, grid computing, P2P computing, and computing system virtualization. He is now the dean of the School of Computer Science and Technology at HUST and a senior member of IEEE and ACM.

Song Wu received his PhD in computer science from HUST, China in 2003. He is a professor at SCTS/CGCL of HUST. His research interests include cloud computing, grid computing, and system virtualization. He is the director of the Parallel and Distributed Computing Institute at HUST and a senior member of the China Computer Federation.

Xuanhua Shi received his PhD in computer architecture from HUST in 2005. He is an associate professor at HUST. His research interests include cluster and grid computing, trusted computing, computing system virtualization, and cloud computing. He is a member of ACM and the China Computer Federation.

Jinyan Yuan is a Master candidate in the School of Computer Science and Technology at HUST. Her research interests include computing system virtualization and cloud computing.