Automated deployment of virtualization-based research models of distributed computer systems

Andrey Zenzinov
Mechanics and mathematics department, Moscow State University
Institute of mechanics, Moscow State University
Moscow, Russia
andrey.zenzinov@gmail.com

***Abstract*—In the research and development of distributed computer systems and related technologies it is appropriate to use research models of such systems based on the virtual infrastructure. These models can simulate a large number of different configurations of distributed systems. The paper presents an approach to automate the creation of virtual models for one of the classes of distributed systems used for scientific computing. Also there considers the existing ways to automate some maintaining processes and provides practical results obtained by the author in the development and testing of prototype software tools to create virtual models.**

***Index Terms*—Distributed computer systems, Virtualization, Automation, Grid computing**

I. Introduction

Research and development of distributed computer systems usually involves performing a lot of testing and development activities. It seems useful to carry out experiments and tests not on a production system, but on its research model created specifically for the purposes of performing the experiments. Virtualization-based research models of computer systems may be used to accurately model software components of such systems. This kind of models is widely used in the study of distributed systems [1], [2].

Another possible use case for virtual research models is development of parallel and distributed programs, e.g. client-server applications, parallel computing programs. It is also applicable to information security sphere, particularly in the development of different monitoring and auditing systems.

With virtualization technologies it is possible to simulate distributed systems of various architecture. Also, the virtualization-based approach significantly simplifies the process of deploying the model and preparing the experiments.

The main idea of this approach is to use one or more computers (virtualization hosts) with a set of deployed virtual machines which run the software identical or similar to the software in the production system. Similar approaches are widely used, e.g. in cloud computing, to deploy multiple computing nodes on a single physical host. The overhead of running a set of virtual machines is relatively low on hosts with hardware virtualization support.

In this research we consider grid computing systems designed for parallel task execution as the object of modelling. Typical examples of these systems are the distributed systems based on the Globus Toolkit—a set of open source software used for building grid computing systems. Distributed systems of this kind usually do not require the use of virtualization technologies to function, contrary to other types of distributed systems, e.g. in cloud computing. This property of typical grid computing systems simplifies the virtualization-based modelling as nested virtualization is not required in this case.

In our research we consider evaluation of information security properties of distributed systems as the goal of modelling. The following attacks are particularly relevant to grid computing systems: denial of service (DoS) and distributed denial of service (DDoS) attacks; exploitation of software vulnerabilities; attacks on the system's infrastructure allowing the attacker to eavesdrop and to substitute trusted components of the system.

Different kinds of modelling parameters should be taken into account, such as the system's architecture, attacker location, configuration and composition of security mechanisms, and attack scenarios. Usually it is necessary to perform a series of experiments for each configuration of parameters to show the adequacy of the experiments' coverage. Modelling different variants of system's architecture requires to iteratively rebuild and redeploy the model, which may be performed at a high degree of automation when using the virtualization-based models.

II. Workflow of distributed system deployment

The building process of the distributed system contains the following steps:

- a set of nodes creation;
- OS installation and setting up on each node;
- additional software installation on each node;
- setting up distributed network.

All these processes takes a lot of time and it is a very monotonous work, which requires carefulness and attention, because mistakes can lead to system failure.

Suppose the operator performing the deployment processes have got given system configuration, which describes nodes of the distributed system, its network architecture, a set of software tools installed on the nodes and other necessary options. The distributed system and its virtual model are being constructed following this configuration.

The operator should perform actions based on the algorithm, which was described above. Some of these actions require their completion, which can take a long time, e.g. disk image copying, software installation, etc. The operator can make mistakes, which may result in system performance loss.

It seems appropriate to reduce the amount of non-automated actions to increase the reliability of deployment.

III. Goals of the research

The aim of the study is to automate the deployment and setting up of virtual research model with given configuration file. Let us require following options from the deployment system:

- support for different types of nodes;
- nodes have a configured required software for remote access to nodes, e.g. via SSH;
- the deployment system should work only with open source software.

The idea of last requirement is that we may need to modify the code of some programs for further development. Different tasks in a distributed system determines different types of nodes. For example, we can divide grid nodes into several types: compute nodes, gateway nodes, certificate distribution nodes, task distribution nodes. These types have different software and configurations. On the other side usually there are not much kinds of nodes.

IV. Ways to automation

Let us consider the process of deployment. It is divided into the steps, as described above, and we're using VMs to emulate nodes. Network infrastructure is also virtualized.

It is convenient to use libvirt [3] for virtual system management. Libvirt is an open source cross-platform API, daemon and management tool for managing platform virtualization. It provides unified controlling for most hypervisors like KVM, Xen, VMware and others. There are API for some popular programming languages like Python. The idea of automation lies in using libvirt, well-known shell scripting automation and programs written in Python.

OS installing is usually interactive. It contains a set of specific questions about disk partitions, packages, users, time zone, etc. These questions are obstacles for automated OS installing. However, there are various solutions such as network installation and using specified file with the answers. These solutions have been successfully used in many modern systems, e.g. compatible with the Debian GNU/Linux and Red Hat Enterprise Linux.

Automation is also applicable to editing configuration files. On the one hand, this implements with text processing tools and specific actions using regular expressions. On the other hand, there are special systems designed to automate the configuration of OS and software – Chef and Puppet [4], [5].

V. Related works

Automation of VM deployment is studied in IBM developerWorks paper "Automate VM deployment" [6]. In this paper the author propose to create separate virtual machines on a given configuration. The described system consists of two parts: Virtual Machine Deployment Manager (VDM) and Virtual Machine Configuration Manager (VCM). Configuration for each VM stored on special disk image. Then it boots on the VM via virtual CD-ROM and configure the system. The VDM handles user requests to deploy VM such as cloning virtual images, configuring VM hardware settings, and registering the VM to the VM hypervisor.
The VCM is installed in the VM template. After a system starts, it will start automatically and launch the configuration applications on the CD with configuration data.

The deployment process is illustrated on figure 1.

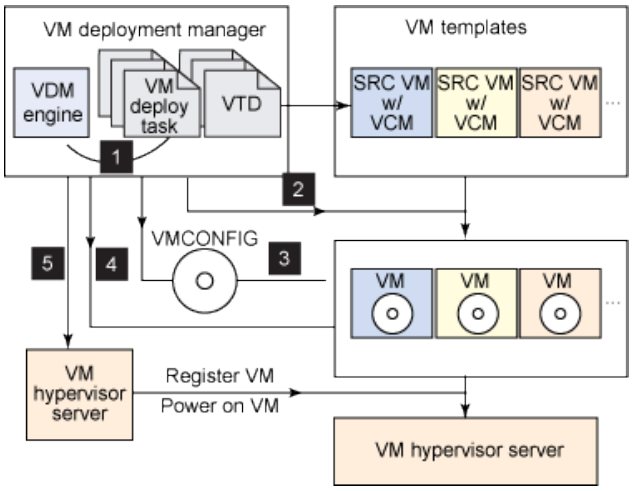

Fig. 1. Architecture of the automatic VM deployment framework

Using pre-configured VM templates and configuration files is the main advantage in this paper. This system is based on VMware virtualization and shell scripts. It supports Red Hat Enterprise Linux, SuSE Linux and Windows.
There are also some disadvantages. Unfortunately, described system does not support creating multiple copies of VM template, which is essential for our research. Use of special configuration CD seems redundant.

Another approach is presented in Vagrant system [7]. Vagrant is an open source tool for building development environment using virtual machines. The idea of this system is to use already prepared VM images, called "boxes". You need only three commands:

```
vagrant box add <name> <box url>
vagrant init <name>
vagrant up
```

These commands launches pre-configured VM with specific configuration. You should create another box to

create a machine with different configuration. Configuration parameters stores in ”vagrantfile“ which describes machine settings such as hostname, network settings, SSH settings and provider (hypervisor) settings. VirtualBox is a default provider for Vagrant. But you can use other hypervisors like VMware via special plugins. Additional software can be installed using the Chef and Puppet.

This system is simple to use, there are Chef, Puppet, SSH, Network File System (NFS) support – it is a major advantage. There are multi-machine support: each machine describes separately in ”vagrantfile“. But concept using in Vagrant suppose that there is a ”master“ machine and limited number of ”slave“ machines. It is not convenient for large-scale distributed system.

VMware presents ”Auto Deploy“ technology in their products [8]. Auto Deploy is based on the Preboot eXecution Environment (PXE) – environment to boot computers using a network interface. Another important part is vSphere PowerCLI – a PowerShell based command line interface for managing vSphere.

Unfortunately, Auto Deploy deals only with VMware ESXi hypervisor and it only available in VMware vSphere Enterprise Plus version, which is non-free.

We should consider CFEngine. It is an open source configuration management system widely used for managing large numbers of hosts that run heterogeneous operating systems. There is support for a wide range of architectures: Windows (cygwin), Linux, Mac OS X, Solaris, AIX, HPUX, and other UNIX-systems. This system is not directly related to virtualization, but it is a proven tool for large systems. The main idea is to use unified configurations that describe required state of the system.

CFEngine automates file system operations, service management and system settings cloning.

VI. Requirements for the deployment system

After the review of existing approaches to automation, let us formulate the requirements for the deployment system:

- using of general configuration;
- VM templates using;
- the ability to create a set of clones of the template VM;
- automated initialization;
- ability to making manual setting up.

Using of the general configuration assumes unified method of describing various systems with general parameters. It means that there is a unified set of parameters for all modelling systems. Requirements for configuration with these parameters are described below. If the simulated system contains of a large number of repetitive nodes, it seems appropriate to use a special VM templates. In this case you should make requested number of copies and possibly add some changes to their configuration. There are two ways of cloning VMs: complete cloning – copying the template disk image; incremental cloning – the base image used in the ”read-only“ mode, and changes of clone’s settings saved in separate files. The second way can significantly reduce disk space usage and deployment time. This economy is particularly noticeable in the large series of experiments. We should note that there is an analogue of this approach applied to memory. It is KSM (Kernel SamePage Merging) technology. There is also KSM modification – UKSM (Ultra KSM).
As to the requirement of ability to manual setting up, it may be necessary when the experiment requires operator interaction.

A. Requirements for configuration file

We should request following for general configurations. General configuration should describe:

- all the types of VMs and number of creating copies;
- the parameters for each VM type (e.g., allocated resources, path to disk image);
- virtualization parameters (type of hypervisor);
- network settings;
- post-install scripts.

This set of parameters is enough to creating wide class of research models.

VII. Author’s approach

At present time, we have created an automatic system to deploy a VM using the libvirt library. It supports the deployment of the model of a distributed system based on a set of VM types. Figure 2 schematically shows the general scheme of polygon.

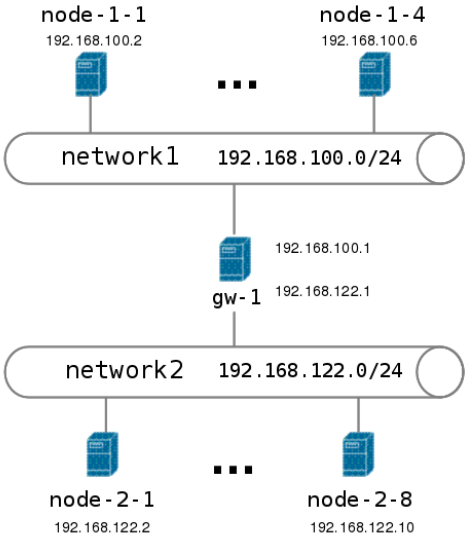

Fig. 2. System diagram

An algorithm includes following steps:

- creating a universal configuration in JSON (figure 3);
- preparing VM template disk images;
- incremental cloning of template images, network settings customization;
- creating XML-descriptions for each VM instance;
- creating a set of VMs based on these XML-descriptions via libvirt methods.

The first two steps are making manually by the operator, and the rest is automated.

```
{
"type" : "kvm",
"machines" : {
  "gw" : {
    "number" : 1,
    "memory" : "524288",
    "disk" : "/images/1.img",
    "network" : [network1, network2]
  },
  "node-1" : {
    "number" : 4,
    "memory" : "262144",
    "disk" : "/images/2.img",
    "network" : [network1]
  },
  "node-2" : {
    "number" : 8,
    "memory" : "262144",
    "disk" : "/images/3.img",
    "network" : [network2]
  }
},
"netconfig" : "Network.cfg"
}
```

Fig. 3. Example configuration of the test model

Figure 3 shows the configuration of the system consisting of three types of nodes: ”gw“ (1 item), ”node-1“ (4 items) and ”node-2“ (8 items). There specified the size of allocated memory and the path to disk image for each type of nodes. ”Network“ option contains a list of used virtual networks, which are described in the configuration file ”Network.cfg“ (figure 4).

”Network1“ in this example – is the network connecting nodes of type ”node-1“, and ”network2“ connecting ”node-2“ nodes. ”Gw-1“ node plays the role of the router and has connected to both networks.

It should be noted that the operator sets the configuration manually, but the rest is automated. Deployed system have automatically configured remote access via SSH and the keys stores on the host machine.

A. Deployment on a distributed host system

There are also a possibility to use distributed host system for deployment. It means that you have a number of physical machines and the operator can deploy a large-scale distributed system based on several host machines. For example, if we have four hosts with 100 VMs, summary there are 400 VMs.

```
{
"networks" : {
  "network1" : {
    "range_start" : "192.168.100.2",
    "range_end" : "192.168.100.255",
    "gateway" : "192.168.100.1",
    "netmask" : "255.255.255.0",
    "nat" : "yes"
  },
  "network2" : {
    "range_start" : "192.168.122.2",
    "range_end" : "192.168.122.255",
    "gateway" : "192.168.122.1",
    "netmask" : "255.255.255.0",
    "nat" : "yes"
  }
}
}
```

Fig. 4. Example configuration of the network (Network.cfg)

Distributed deployment system requires following:

- all hosts are connected to the VPN-network;
- there are one controlling host (Deployment Server);
- NFS-server is installed on the Deployment Server;
- configuration file ”hosts.json“ (figure 5) is stored on the DS contains IP-addresses of all hosts;
- DS have remote access to other hosts via SSH.

```
{
"hosts" : {
  "Deployment Server" : {
    "address" : "10.8.0.1",
    "config" : "server_conf.json",
    "netconfig" : "server_netw.json"
  },
  "host1" : {
    "address" : "10.8.0.4",
    "config" : "host1_conf.json",
    "netconfig" : "host1_netw.json"
  },
  "host2" : {
    "address" : "10.8.0.8",
    "config" : "host2_conf.json",
    "netconfig" : "host2_netw.json"
  }
}
```

Fig. 5. Example configuration of the distributed deployment system

To start deployment process the operator should launch server application on DS, and then launch client applications on hosts.

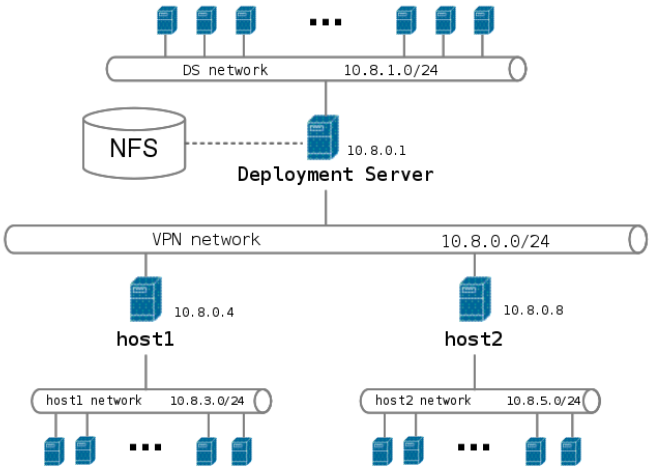

Fig. 6. Distributed system diagram

Figure 6 shows a distributed system with general NFS-store. Additionally, there are possibility to allow access to host's virtual networks to other hosts using VPN-tunnel.

VIII. Experiments

Using the developed software was made a series of experiments on the parallel tasks execution. Experiments showed that deployed virtual model of the distributed computer system satisfies the requirements for the system functions. Particularly, the created nodes can execute remotely received tasks.

The test were conducted using remote access via SSH. There were created program scripts which launches DDoS-attack against one chosen node automatically, while the other nodes were attacking. Simulated attacks were successful. A network access to the victim VM was blocked.

The scenario of the experiment is parameterized, i.e. you can change parameters of the experiment such as number of nodes, addresses, launching tasks, but the used script is universal.

The experiments were performed with Intel i5-3450 based system with 16 GB RAM. There was deployed a model of distributed system consisting of 200 nodes on this host. Elapsed time of deployment was 24 minutes. Deduplication technologies such as UKSM led to this result. At the launch time the memory use was 14 GB, and one hour later it was reduced to 7 GB.

IX. Conclusion

As a result of this research we have created a software prototype for deploying a virtual model of a distributed computer system. Virtualization-based models of grid computing systems, produced with the help of the developed software, were used to simulate various processes in such systems including their regular functioning and the reaction to denial-of-service attacks.

Further work is planned to add support for automated deployment of the software for distributed and parallel computing, such as the Globus Toolkit and MPI implementations. Beside that, we plan to add support for nested virtualization in order to create virtualization-based models for the systems which use virtualization technologies by themselves—cloud computing systems being a notable example.

References

[1] Grossman, R., et al. The open cloud testbed: A wide area testbed for cloud computing utilizing high performance network services. Preprint arXiv:0907.4810. 2009.
[2] Krestis A., et al. Implementing and evaluating scheduling policies in gLite middleware // Concurrency and Computations: Practice and Experience. Wiley, 2012. URL: http://www.ceid.upatras.gr/faculty/manos/files/papers/cpe_2832_Rev_EV.pdf
[3] Libivrt. The virtualization API. URL: http://libvirt.org
[4] Chef. // Opscode. URL: http://www.opscode.com/chef/
[5] Puppet Labs: IT Automation Software for System Administrators. // Puppet Labs. 2013. URL: https://puppetlabs.com/
[6] Yong Kui Wang, Jie Li. Automate VM Deployment // IBM.COM. 2009. URL: http://www.ibm.com/developerworks/linux/library/l-auto-deploy-vm/
[7] Vagrant // HashiCorp. 2013. URL: http://www.vagrantup.com/
[8] VMware vSphere Auto Deploy: Server Provisioning // VMware. 2013. URL: http://www.vmware.com/products/datacenter-virtualization/vsphere/auto-deploy.html