# V-MCS: A Configuration System for Virtual Machines

Xian-He Sun, Cong Du, Hongbo Zou, Yong Chen, Prerak Shukla
*Computer Science Department*
Illinois Institute of Technology
Chicago, IL, USA
{Sun, ducong, hzou1, chenyon1, pshukla3}@iit.edu

***Abstract*—Vitual Machine (*VM*) technology encapsulates shared computing resources into secure, stable, isolated and customizable private computing environments. While service-oriented computing becomes more and more a norm of computing, VM becomes a must-have common structure. However, creating and customizing a VM system on different hardware/software environments to meet versatile demands is a state-of-the-art task, especially for casual users working in new computing environments. In addition, VM configuration without system support is tedious, time consuming, and error prone. In this study, we propose a Virtual Machine Configuration System (*V-MCS*) for tackling this issue. V-MCS takes a systematic approach to enhance the flexibility and usability of VM. It provides an easy-to-use web interface to users to create their preferred configurations, and to convert the configurations into PAN documents for human-computer interaction and XML documents for machine automation. The underlying definition component parses the configurations and the spawn component generates customized VMs on the fly. V-MCS maintains and deploys these two-level documents when users login in the future. With the help of V-MCS, users can generate their customized VMs easily and swiftly. V-MCS has been implemented and tested. Experimental results match the design goal well.**

## I. INTRODUCTION

Unlike conventional operating systems, *Virtual Machine* (*VM*) allows efficient and dynamic multiplexing of users onto physical resources at the granularity of a single user per operating system session, called *guest operating system*, thereby supporting per-user VM configuration and isolation from other users sharing the same physical resource [12]. Multiple independent guest operating systems can co-exist on the same server hardware [6]. Virtual Machines can be highly customized. It is possible to specify virtual hardware parameters, such as memory and disk size, as well as system software parameters, such as operating system version and kernel configuration. However, the creation and configuration of a VM is a highly challenging task, under current systems.

Current virtualization techniques embed the configuration and management information within the VM image. The configuration information is based on the experience of manual setup and installation and, therefore, cannot be easily extracted, and cannot be managed automatically. It requires system administrator's intervention and is generally time-consuming. Configuring and managing VMs has become a technical huddle preventing VM technology being adopted in actual distributed computing practice.

In addition, the manual hardware and software configuration of a VM is diverse, hard to represent and error prone. The users or applications are not likely to provide every detail of the desired VM. Some software components depend on its lower level software and hardware configuration [5]. For example, if the user asks for OpenSSH 4.0 installed on the VM, the VM configuration has to include the OpenSSL 0.9.6 and Zlib 1.1.4 packages or their later versions as dependency resolution. The configuration system has to complete the VM definition based on its knowledge.

Another challenging issue is how to store and retrieve an existing VM effectively and efficiently. The current approaches are achieved via the huge VM image transfers. This approach is very slow and space consuming. A new virtual environment modeling mechanism is in demand.

Facing these challenges, we present *V-MCS*, a *VM Configuration System*, to balance these imposing requirements using PAN [3] template and XML skeleton to abstract and represent VM at user level and system level respectively. PAN is a high-level configuration language, and we leverage its functionality for the interaction between end users and machines. XML provides a natural support for machine-machine interactions. We provide a web interface to list the existing parameter options for users to configure their system. After users submitting their configuration, V-MCS creates a PAN template automatically to record user's requirements. The administrator can integrate management and knowledge rules into the system to verify the user's configuration depicted in a PAN template. An XML skeleton based on the PAN template is generated and stored with the PAN template on the server along with user's identification information.

Based on the XML skeleton, V-MCS is capable of creating a customized VM swiftly for users. In addition, V-MCS system can regenerate and duplicate customized VM for users at anytime. For the whole process, users just need to choose desired configurations on the web portal and the administrator simply needs to manage configuration rules in form of PAN templates. All of these designs make V-MCS capable of reducing the administrative burden substantially and enhancing the flexibility and usability of VM significantly.

The rest of this paper is organized as follows. The detailed description of V-MCS system architecture and implementation are introduced in the following two sections. The subsequent section presents experimental results to demonstrate how well V-MCS matches our design goals. Finally, we summarize our study and discuss the future work in the last section.

## II. V-MCS System Architecture

This section describes the system and software architecture of V-MCS. Firstly, we introduce the design goals, and then present the system design including a high-level overview of the system, and the purpose and operation of each core component. The interaction among components is covered in the introduction of each component as well.

### *A. Design Goals*

We have studied existing literature and generalized four requirements for designing an efficient system to make VM configuration and incarnation automatically and swiftly. [1, 2, 8, 9, 14] Without imposing strong requirement on technical knowledge, the proposed V-MCS system targets to meet such needs and help users and administrators to configure and manage their VMs efficiently.

*1) Not just simple interface:* The user interface of V-MCS is helpful to users. Similar to other web interfaces of virtual environment configuration, an efficient VM configuration system should provide a set of options to its users too. In addition, it should provide even more customized services specifically for users with different technical background. For instance, a user might know the software they want to use, but does not know what software or what specific hardware support it requires. As a consequence, the system should provide an aid interfaces to help users configuration. It is not sufficient to just provide simple options. A default configuration should support to resolve these issues. This default configuration should be provided by some basic platform configuration templates for users. Default templates can be derived from user's previous configuration logs, historical favourites trace and general configuration knowledge. If this goal is achieved, users and administrators should be able to create their platform easily and swiftly.

*2) Rectifying conflicting options:* The causal user is not a professional, who has no enough experience and knowledge to avoid various problems, such as the compatibility issues among different software, systems, and platforms. For example, users cannot install a 64-bit OS on the 32-bit hardware architecture, and it will be a time-consuming and tedious task for administrators to check this problem for every user. V-MCS should design some rules to help users to detect and rectify conflicting options automatically.

*3) Elimination of software dependency:* Software dependency is related to the complex dependent relationship among software packages, which almost cannot be found until packages being called. So, it is different to be solved by conflicts rectification, and can not be exposed when the user is filling in his configurations. Using dependency knowledge rules, system can extract hidden-software packages should be installed after completing configuration process. Dependency knowledge rules can be learned at the time of dependency emergence.

*4) Automatic creation of VM:* Configuration system should provide a function that creates a VM entirely from scratch without intervention of system administrator. To achieve this goal, we should provide a efficient VM modeling to represent and save VM configurations in V-MCS. Based on the modeling, VM will be generated and deployed on the target physical machine automatically. Furthermore, V-MCS will use this saved modeling information to re-construct users' working environment when users login the system again.

### *B. Design of V-MCS system*

Effective design plays an important role in every successful project. Taking all of these factors into consideration and to achieve goals mentioned above, we design V-MCS with four components as shown in Figure 1. We discuss the system overview and the design of each component in detail respectively.

*1) V-MCS overview:* As shown in Figure 1, the proposed V-MCS system contains four major components, web portal, definition component, spawn component and knowledge center. Web portal provides a user-friendly interface and authentication functionality for a user to access their figuration. On this process, V-MCS will provides some default configurations on web portal based on suggestion and rule temples saved in KC. With the help information, the user can configure their desired VM swiftly on web portal. Figure 2 shows a rule template sample.

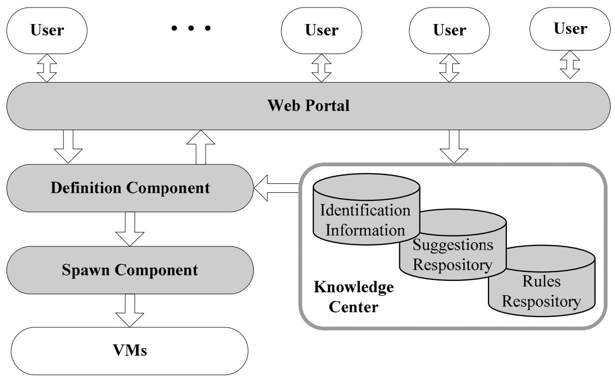


Figure 1. V-MCS components

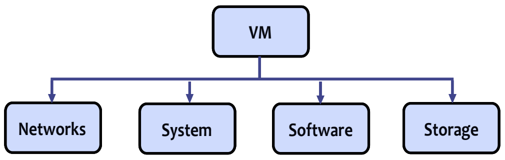
Once user configurations are submitted, definition component will generate a PAN template and XML skeleton following the user's submitted configuration information. The definition component will also verify the user input based on the knowledge rules in the knowledge center.

```
structure template python;
 “this” = create( “package” );
 “this/PkgID” = “77” ;
#package information
 “this/name” = “python” ;
 “this/version” = “2.2.2” ;
 “this/release” = “26” ;
 “this/location” = undef;
 “this/image/” = “python.img” ;
valid “/packages/tlk/version” >= “8.4” ;
valid “/packages/gcc/version” >= “3.3” ;
```

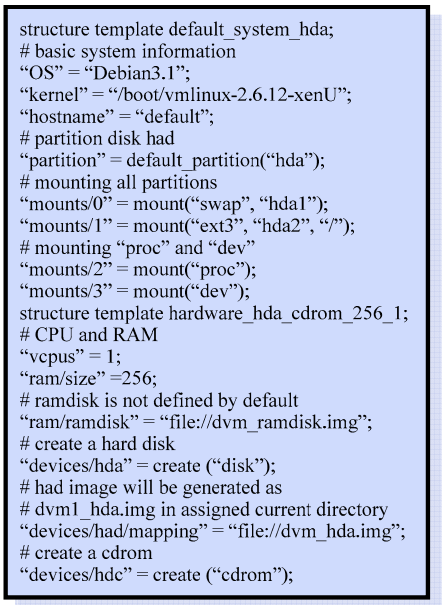
Figure 2. An example of software dependence

After that, the spawn component (or incarnation component) will create a VM instance based on the XML skeleton on the assigned machine. Compared with the definition component that takes care of all the configuration phases involved in the system, the spawn component takes care of the creation of VM environment based on the final configuration generated by the definition component. Following the steps described above, V-MCS can help users to create their preferred virtual computing environment efficiently.

*2) VM Modeling:* To facilitate describing VM in various formats, we employ four abstractions to represent any VM environment. We call the script containing the modeling information as the Virtual Machine Description Script (VMD). Based on our prior studies and analyses [4], four types of information should be included in virtual machine description script: networks, system, software, and storage, as illustrated in Figure 3 (a). Figure 3 (b) shows an example of a PAN template, which includes basic system configuration information and part of hardware configuration information. At the beginning of the template, the system information is declared, including OS type, system kernel's location, hostname, and disk information etc. The consequent part is hardware related information, including CPU, RAM, and device information.



(a)



```
structure template default_system_hda;
# basic system information
"OS" = "Debian3.1";
"kernel" = "/boot/vmlinux-2.6.12-xenU";
"hostname" = "default";
# partition disk had
"partition" = default_partition("hda");
# mounting all partitions
"mounts/0" = mount("swap", "hda1");
"mounts/1" = mount("ext3", "hda2", "/");
# mounting "proc" and "dev"
"mounts/2" = mount("proc");
"mounts/3" = mount("dev");
structure template hardware_hda_cdrom_256_1;
# CPU and RAM
"vcpus" = 1;
"ram/size" =256;
# ramdisk is not defined by default
"ram/ramdisk" = "file://dvm_ramdisk.img";
# create a hard disk
"devices/hda" = create ("disk");
# had image will be generated as
# dvm1_hda.img in assigned current directory
"devices/had/mapping" = "file://dvm_hda.img";
# create a cdrom
"devices/hdc" = create ("cdrom");
```

(b)

Figure 3. (a) Virtual machine description (b). System template example

*3) Definition Component:* The task of the definition component is to directly interact with the user interface, manipulate the configuration and create a configuration that is fed into the spawn component. Our design employs a two-level approach for the definition component, and its task is divided into two sub-components, user-level definition component and system-level definition component.

The user-level component uses a high-level language, whereas the system-level component uses a low-level language. We adopt PAN for a user-level representation of configuration options and XML for low-level representation. PAN [3] was initially designed to represent the node configuration of Data Grid. It is descriptive and thus, a user can easily understand and manually create a script. It has the validation mechanism that checks the configuration and ensures the script is correct. However directly using PAN templates to create virtual machine raises some issues as it is not structural in nature and hard to interpret automatically. In addition, PAN provides features like conversion to XML and validation. We can write validation scripts (to check the user input) using PAN efficiently. The user-level definition component interacts with user input and generates PAN templates based on the information provided by the user.

*The Suggestion Repository* (*SR*) contains the suggestions for the default option of a particular set of configuration. For example, let us consider a situation where a user selects software that has better compatibility with GCC compiler, but the user is unaware of this fact and selects the default Intel C compiler. At this time, users can get a suggestion rule that V-MCS takes out from suggestion repository. For the situation explained above, our system can help users through providing the best suitable option rather than listing the whole fixed options for users. The suggestion database may contain additional performance enhancement suggestions or other parameters. It provides the flexibility to user in the sense that V-MCS "dynamically" sets the optimal configuration for users. Our current design requires system administrator's intervention in order to generate these suggestions. The form of suggestion is also kept in PAN template format to keep design simple and uniform and its validation feature used to suggest the right value based on default parameters provided by the user.

As the term indicates, the *Rule Repository* (*RR*) contains the rules that must be followed when configuring a VM. Security policies, restriction or violation of software dependency related rules will be stored in it. These rules are utilized along with the suggestion database to create the optimal configuration. The form of rules is similar to suggestions using PAN templates.

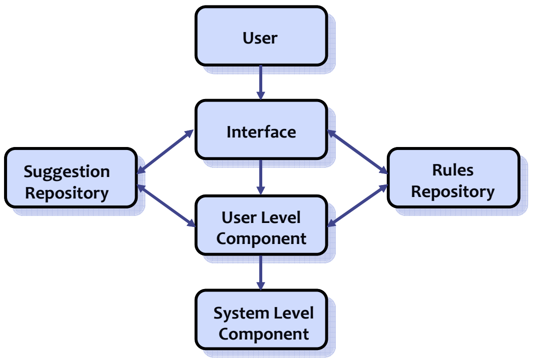


Figure 4. Overview of definition component

Figure 4 depicts the interaction workflow between the definition component and repositories. Firstly, user-level definition component generates PAN template based on users' input from interface. Then, definition component will recall rule knowledge coming from SR and RR to verify the user's input and eliminate software dependencies. After that, a corresponding XML skeleton is generated based on the validation PAN skeleton using definition components.

The system-level component deals with the low-level XML skeletons generated from the user-level component. It handles the storage of XML files for particular users in the database. The following situation will give insights of the low-level component.

Let us consider a situation where user requires a VM for their job. They visit the website for user configuration. They log in using the username/password pair. The user goes through various options quickly with the help from web portal and definition component. If their configuration conflicts with the rules set in the RR, definition component will provide the default options to the user via the SR. After a successful configuration, a template in PAN language is generated. The resulting PAN template is compiled to generate the XML skeleton by using the PAN compiler. The XML output is stored on storage server's database along with the authentication information of that particular user. After the XML skeleton being generated, the spawn component does the rest of the creation task and delivers user their desired VM. In the above process, the XML creation and VM instantiation are included in the task of system-level component.

4) *Spawn Component:* As the name suggests, the task of the spawn component is to create VM on the target physical machine. The spawn component can be viewed as software running on the target physical machine. As explained in the VM modeling, the information required to describe any virtual environment can be divided into four abstractions. We have created a parser program, within spawn component, to fetch the information from the XML skeleton and to call spawn scripts.

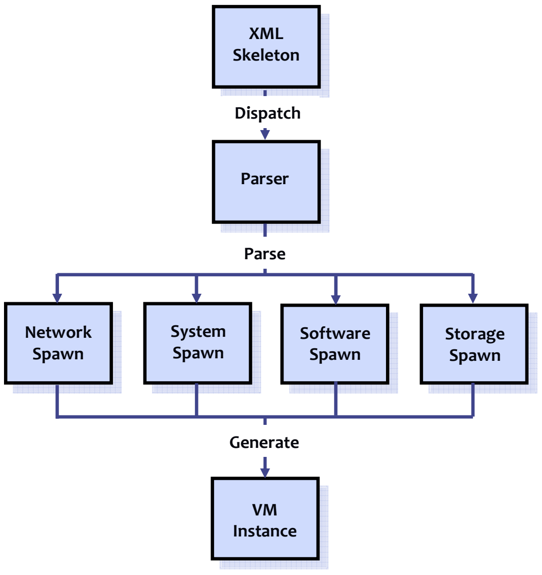


Figure 5. Workflow of the spawn component

In order to process the information and to create the VM, there are four spawn scripts implemented (Network, System, Storage and Software). These scripts get the information fetched from the XML. The scripts process the relevant information and create configurations in turn. After the entire configuration is completed, the VM is instantiated. Figure 5 illustrates the complete design and work flow of the spawn component. This component is mainly composed by one parser and four spawn scripts. The parser scans the XML VMD script and generates four spawned files that are used by the Virtual Machine Monitor (VMM) as the configuration files. Finally, A VM instance matching the configuration requirements is generated by a pre-defined Python script.

## III. Implementation

We chose Xen as the virtualization platform in our implementation. We selected Xen because it is open source, and has plenty of useful tools. We used a NFS partition as shared data storage, and created a dedicated daemon process to deal with users' requirement. The parser program that fetches information from XML skeleton was written using Flex and Bison. The spawn scripts were written in Perl. The OS platform for current implementation is Debian with kernel version 2.6.x.

We used debootstrap to create Debian system from scratch. We also used apt, a package handling tool of Debian, to configure, install and resolve dependency of software packages. Our current implementation also supports the configuration, installation, and dependency resolution features.

Figure 6 shows the web portal provided to users for inputting their requirement. Every user can operate his VMs using PuTTY which has been embedded in our web portal. Moreover, every user can monitor all their VMs in web portal directly.

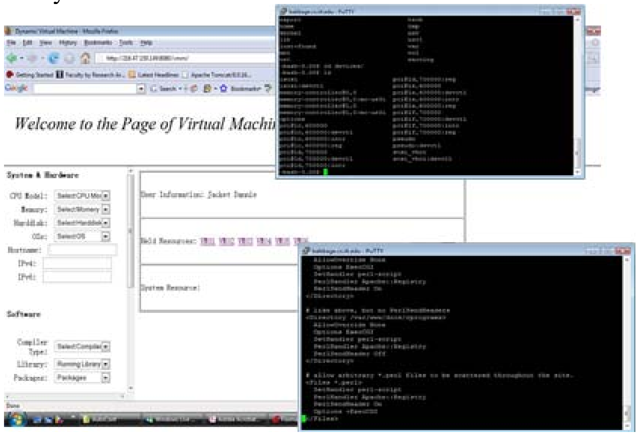
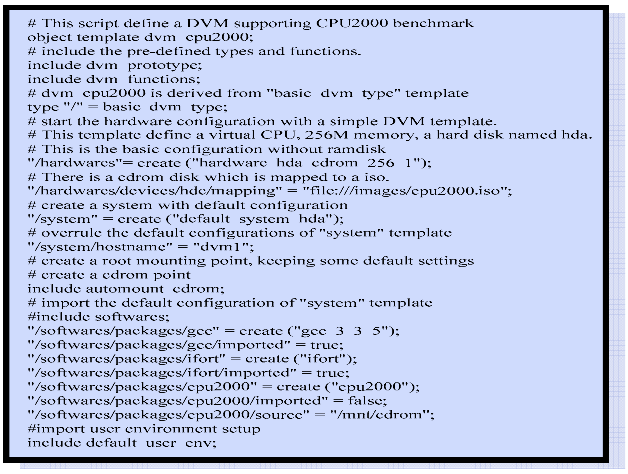


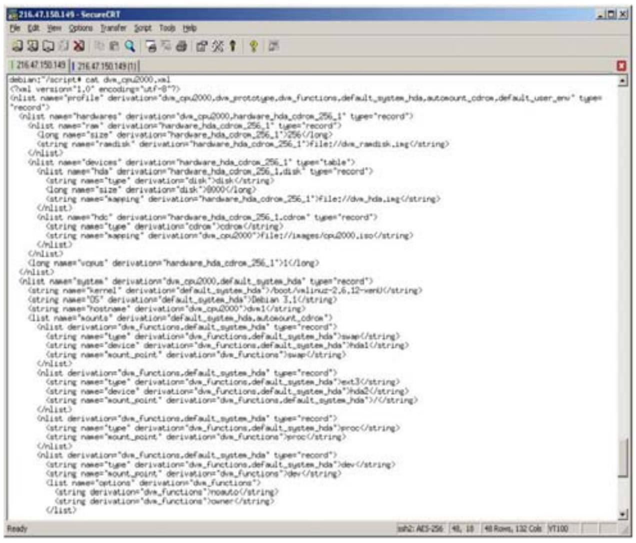
Figure 6. Web Portal Embedded PuTTY

Figure 7 (a) shows a PAN template example, which was generated by definition component based on the user's requirements. At the beginning of the template, the dvm_cpu2000 is defined as an object template that can be instantiated into a VM. The dvm_prototype and dvm_functions are two templates declaring the VM basic type structure and functions to compute or validate resource configurations. Figure 7 (b) shows the compiled and validated XML skeleton yielded by PAN template.



```
# This script define a DVM supporting CPU2000 benchmark
object template dvm_cpu2000;
# include the pre-defined types and functions.
include dvm_prototype;
include dvm_functions;
# dvm_cpu2000 is derived from "basic_dvm_type" template
type "/" = basic_dvm_type;
# start the hardware configuration with a simple DVM template.
# This template define a virtual CPU, 256M memory, a hard disk named hda.
# This is the basic configuration without ramdisk
"/hardwares"= create ("hardware_hda_cdrom_256_1");
# There is a cdrom disk which is mapped to a iso.
"/hardwares/devices/hdc/mapping" = "file:///images/cpu2000.iso";
# create a system with default configuration
"/system" = create ("default_system_hda");
# overrule the default configurations of "system" template
"/system/hostname" = "dvm1";
# create a root mounting point, keeping some default settings
# create a cdrom point
include automount_cdrom;
# import the default configuration of "system" template
#include softwares;
"/softwares/packages/gcc" = create ("gcc_3_3_5");
"/softwares/packages/gcc/imported" = true;
"/softwares/packages/ifort" = create ("ifort");
"/softwares/packages/ifort/imported" = true;
"/softwares/packages/cpu2000" = create ("cpu2000");
"/softwares/packages/cpu2000/imported" = false;
"/softwares/packages/cpu2000/source" = "/mnt/cdrom";
#import user environment setup
include default_user_env;
```

(a)



```
debian:~/script# cat dvm_cpu2000.xml
<?xml version="1.0" encoding="utf-8"?>
<nlist name="profile" derivation="dvm_cpu2000,dvm_prototype,dvm_functions,default_system_hda,automount_cdrom,default_user_env" type=
"record">
  <nlist name="hardwares" derivation="dvm_cpu2000,hardware_hda_cdrom_256_1" type="record">
    <nlist name="ram" derivation="hardware_hda_cdrom_256_1" type="record">
      <long name="size" derivation="hardware_hda_cdrom_256_1">256</long>
      <string name="ramdisk" derivation="hardware_hda_cdrom_256_1">file://dvm_ramdisk.img</string>
    </nlist>
    <nlist name="devices" derivation="hardware_hda_cdrom_256_1" type="table">
      <nlist name="hda" derivation="hardware_hda_cdrom_256_1,disk" type="record">
        <string name="type" derivation="disk">disk</string>
        <long name="size" derivation="disk">8000</long>
        <string name="mapping" derivation="hardware_hda_cdrom_256_1">file://dvm_hda.img</string>
      </nlist>
      <nlist name="hdc" derivation="hardware_hda_cdrom_256_1,cdrom" type="record">
        <string name="type" derivation="cdrom">cdrom</string>
        <string name="mapping" derivation="dvm_cpu2000">file://images/cpu2000.iso</string>
      </nlist>
    </nlist>
    <long name="vcpus" derivation="hardware_hda_cdrom_256_1">1</long>
  </nlist>
  <nlist name="system" derivation="dvm_cpu2000,default_system_hda" type="record">
    <string name="kernel" derivation="default_system_hda">/boot/vmlinuz-2.6.12-xenU</string>
    <string name="OS" derivation="default_system_hda">Debian 3.1</string>
    <string name="hostname" derivation="dvm_cpu2000">dvm1</string>
    <list name="mounts" derivation="default_system_hda,automount_cdrom">
      <nlist derivation="dvm_functions,default_system_hda" type="record">
        <string name="type" derivation="dvm_functions,default_system_hda">swap</string>
        <string name="device" derivation="dvm_functions,default_system_hda">hda1</string>
        <string name="mount_point" derivation="dvm_functions">swap</string>
      </nlist>
      <nlist derivation="dvm_functions,default_system_hda" type="record">
        <string name="type" derivation="dvm_functions,default_system_hda">ext3</string>
        <string name="device" derivation="dvm_functions,default_system_hda">hda2</string>
        <string name="mount_point" derivation="dvm_functions,default_system_hda">/</string>
      </nlist>
      <nlist derivation="dvm_functions,default_system_hda" type="record">
        <string name="type" derivation="dvm_functions,default_system_hda">proc</string>
        <string name="mount_point" derivation="dvm_functions">proc</string>
      </nlist>
      <nlist derivation="dvm_functions,default_system_hda" type="record">
        <string name="type" derivation="dvm_functions,default_system_hda">dev</string>
        <string name="mount_point" derivation="dvm_functions">dev</string>
        <list name="options" derivation="dvm_functions">
          <string derivation="dvm_functions">noauto</string>
          <string derivation="dvm_functions">owner</string>
        </list>
```

(b)

Figure 7. (a) VMD in PAN format (b). VMD in XML format

## IV. Experimental Results

Experimental testing was conducted to verify the feasibility of the design and to measure the effectiveness of the implementation. The performance test of the VM instantiation, initiation and configuration was performed on a Dell Dimension 3000 machine with one P4 2.6GHz CPU, 1GB Memory, and 160GB hard disk. We used Xen 3.0.1 as the virtualization platform. Configuration information was inputted through Web portal, and then the system generated relevant XML skeleton to initiate VM.

Experimental testing has compared the average instantiation time of users' compute nodes under three setup situations – physical machines (PM), VMM without V-MCS (w/o V-MCS), and VMM with V-MCS (V-MCS). Table 1 lists the system configuration information of the compute node. SPECCPU 2000 benchmarks need to be installed on every compute node to simulate users' compute application.

TABLE I. Virtual Compute Node Configurations

| Configurations | Value |
|---|---|
| Initial Memory | 512 MB |
| Maximum Memory | 1024MB |
| Virtual CPUs | 1 |
| Operating System | Generic 2.6.24 kernel |
| Storage Disk Size | 2000MB |
| Networks Type | Shared physical device |

Table 2 shows the comparison of performance results of one node installation time under three conditions. As listed in this table, the installation time on a physical machine takes 854 seconds, which is much longer than the time taken with the other two methods. The main reason is that system installation on VMM just needs to bootstrap Linux kernel, which is same with image clone. The time of the compute node instantiation on VMM, thus, should be shorter than instantiation on a physical machine. In addition, because V-MCS needs more time to check and rectify configuration confliction at the first time, V-MCS increases virtual machine domain creation time on VMM. This deficiency, however, will be complemented, when users create more compute nodes.

TABLE II. VM Incarnation Time with SPECCPU2000 Application (Seconds)

| | DomU Creation | Guest OS Installation | Application Installation |
|---|---|---|---|
| On PM | No | 837 | 17 |
| w/o V-MCS | 67 | 521 | 24 |
| V-MCS | 176 | 562 | 27 |

Figure 8 shows different performance result of multi-node instantiation under three conditions. In our experiment, when the nodes reach and exceed four nodes, VMCS is able to reduce the average time of system installation.

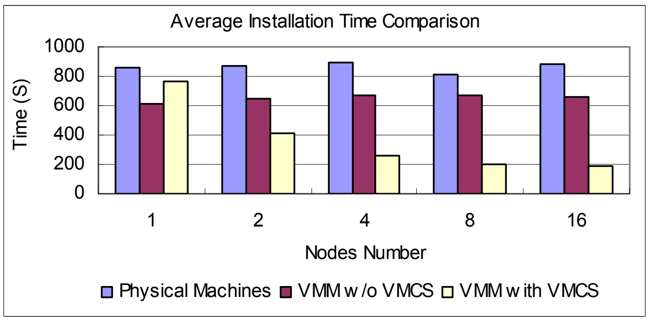


Figure 8. Multi-node Average Installation Time Comparison

To further analyze the results, we break down the installation steps under three conditions and report the time consumed on each step as in Figure 9. As shown in Figure 9, the average time taken by every installation step is consistent on the Physical Machine (PM) with the same hardware configuration. There are subtle differences based on the hardware performance. However, the system installation time on every node has large variation under VMM situations.

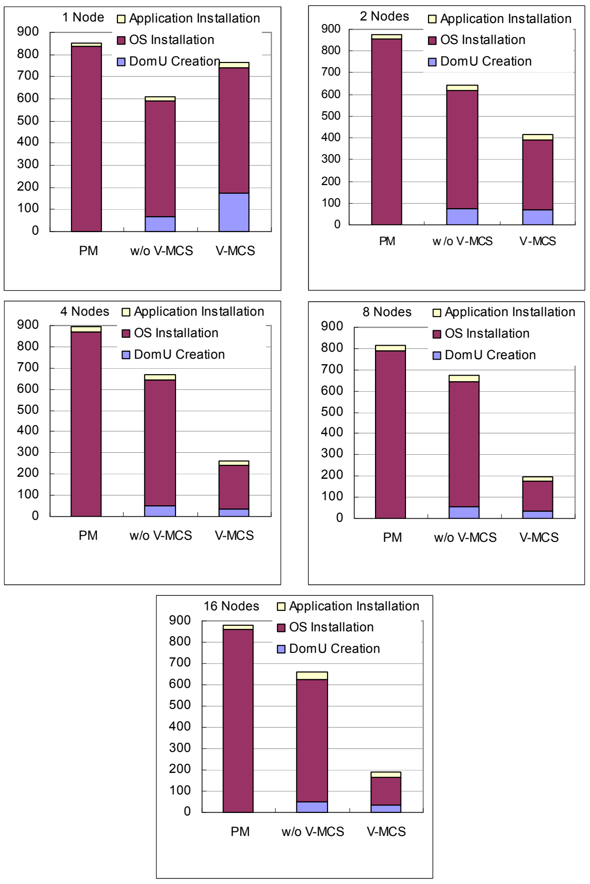


Figure 9. Average Time Comparison of Every Installation Step Under Three conditions

Under the VM situation, the average installation time with and without V-MCS has large variations. This is because that every system node installation on VMM needs to download OS image from Internet separately. So, network bandwidth and repository response time both affect the system installation average time. With the assistance of V-MCS, however, the system needs not to be downloaded and configured on every virtual node. V-MCS can record system configuration and save the core OS image at the first system installation. The installation of rest nodes simply needs to copy the previous image through local network. That is why V-MCS achieved better performance, when the number of installation nodes was increased to 4, 8, and 16.

Our experimental testing has verified the complete process of VM modeling and incarnation, including establishing VM basic type structure and templates, modeling a VM with user configuration, compiling and validating VMD scripts, generating platform independent XML script, and instantiating and initiating a VM. Experimental results show that V-MCS is efficient, and significantly better than existing methods of VM configuration and creation. Especially, V-MCS will be more effective for virtual nodes configuration and creation with large-scale size.

## V. Related Work

V-MCS is a part of our *Dynamic Virtual Machine (DVM)* project [4], which provides support for dynamic VM instantiation and migration. We have successfully applied the PAN template and XML skeleton into the VM migration and virtual group management. A detailed introduction about DVM middleware can be found in [5].

Prior work of VM configuration can be roughly classified into three categories: (a) manually configuring according to the configuration documents; (b) semi-automatically configuring with the help of sample templates; (c) Other configuration functionalities.

### *A. Manual configuration*

This is a basic configuration method. Most systems provide user manual and tell users how to modify various parameters to obtain a basic system. Nevertheless, manual configuration requires users must directly access system file to modify configuration. In some sense, this method imposes strong technical knowledge requirement on users. For example, at the early stage of Xen [13], users have to figure out how to configure VM before installing it.

### *B. Configuring with sample template*

Most systems offer sample templates coming with default configurations. Users can create VM instance with the default configuration directly. Users can also modify the sample template catering for their requirement. However, same as the manual configuration, this method does not provide automatic configuration support. Compared with the manual configuration, this approach provides sample template and reduces the required technical knowledge for users. The latest Xen user's manual describes this method well [13].

### *C. Other configuration methods*

In recent years, several projects endeavor to enable virtualization in distributed computing environments. Figueiredo et. al. explored the feasibility of VMs on Grid computing and proposed a VM based architecture for Grid computing [6]. They aimed to isolate the user and administrator's views, and to provide system security and ease the administration. Virtual private workspace [7] focuses on the authentication, authorization, and resource management of a virtual environment in Grid. However, these projects are designed to use current virtualization technologies in a distributed environment. They do not support the modeling, and incarnation of a virtual environment. Current virtualization techniques [10] embed the configuration and management information within the VM image. The configuration information, therefore, cannot be easily extracted from experiential process of manual setup and installation, and cannot be managed automatically. Some VM management systems [1, 2, 8, 9, 14] integrate certain VM configuration functions in it. These solutions have their special scenarios, so they are separate solutions comparing our system. In addition, a toolkit named Virtual Machine Manager (Virt-manager) also helps configure VM on single machine, but does not provide support for large scale configuration, such as for two or more nodes. [15]

## VI. Conclusion and Future Work

While Virtual Machine (VM) technology gains rapidly increasing attention and popularity for its security, back-ward compatibility and many other novel features, the fast and efficient configuration and deployment of VM remains elusive. We have presented the design and implementation of a Virtual Machine Configuration System, or V-MCS in short, in this study, to help users and administrators to configure and manage VM flexibly and make VM more usable. We have defined and modeled the VM using VM descriptions, and introduced the PAN template and XML skeleton to represent VM at the user and system level respectively. The proposed modeling mechanism and two-level representation system maintain all the configuration information, and make configuration verification and VM incarnation easier. Experimental results have demonstrated that V-MCS is effective and efficient in the VM creation and incarnation.

Our current experiments are preliminary, but the results are promising. V-MCS is a part of our Dynamic Virtual Machine project, and we are extending VM monitoring and management functionalities into V-MCS to serve DVM better. Furthermore, we plan to perform more experiments on various environments such as parallel computing environments for large-scale scientific applications, which will put more attention on VM migration and communication. V-MCS is moving forward with these desires.

## References

[1] Albrecht, J., R. Braud, D. Dao, N. Topilski, C. Tuttle, A. C. Snoeren, A. Vahdat, "Remote Control: Distributed Application Configuration, Management, and Vistualization with Plush," *Proceedings of the twenty-first USENIX Large Installation System Administration Conference (LISA '07)*, 2007.

[2] Begnum, K. M., "Managing Large Networks of Virtual Machines", Proceedings of the twentieth USENIX Large Installation System Administration Conference (LISA '06), 2006.

[3] Cons, L., P. Poznanski, "Pan: A High-Level Configuration Language," Proceedings of the Sixteenth USENIX Large Installation System Administration Conference (LISA '02), 2002.

[4] Du, C., "Dynamic Virtual Private Environment in A Shared Cyberspace," *Illinois Institute of Technology Ph.D dissertation*, 2008.

[5] Du, C., X-H. Sun, "Model and Incarnate Virtual Environments," IIT-CS Technical Report (No. IIT-CS-06-01), *Illinois Institute of Technology*, 2006.

[6] Figueiredo, R. J., P. A. Dinda, J. A. B. Fortes, "A Case For Grid Computing On Virtual Machines," *Technical Report TR-ACIS-02-001*, 2002.

[7] Keahey, K., M., Ripeanu, K., Doering "Dynamic Creation and Management of Runtime Environments in the Grid," *Workshop on Designing and Building Web Services (GGF9)*, Chicago, October, 2003.

[8] Krsul, I., A. Ganguly, J. Zhang, J. A. B. Fortes, R. J. Figueiredo, "VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing," *Supercomputing 2004*, Pittsburg, PA, Nov. 2004.

[9] McNett, M., D. Gupta, A. Vahdat,. G. M. Voelker, "Useher: An Extensible Framework for Managing Clusters of Virtual Machines," *Proceedings of the twenty-first USENIX Large Installation System Administration Conference (LISA '07)*, 2007.

[10] Nagarajan, A. B., F. Mueller, "Proactive Fault Tolerance for HPC with Xen Virtualization," *Proceedings of 21st ACM International Conference on Supercomputing,* Seattle, WA, June 2007.

[11] Naik, V. K., P. Garbacki, A. Mohindra, "Architecture for Service Request Driven Solution Delivery Using Grid Systems," *Proceedings of International Conference on Services Computing*, Chicago, Sept. 2006.

[12] Smith, J., R. Nair, "Virtual Machine: Versatile Platforms for Systems and Processes," *Morgan Kaufmann Publishers*, 2005.

[13] The Xen Team, "Users' Manual Xen v2.0 for x86", http://www.cl.cam.ac.uk/research/srg/netos/xen/documentation.html.

[14] Vallee, G., T. Naughton, S. L. Scott, "System Management Software for Virtual Machine," *Proceedings of the forth International Conference on Computing Frontiers*, 2007.

[15] Virtual Machine Manager, http://virt-manager.et.redhat.com/.