World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:6, No:11, 2012

CASTE: a Cloud-Based Automatic Software Test Environment

Fuyang Peng, Bo Deng, and Chao Qi

***Abstract*—**This paper presents the design and implementation of CASTE, a Cloud-based automatic software test environment. We first present the architecture of CASTE, then the main packages and classes of it are described in detail. CASTE is built upon a private Infrastructure as a Service platform. Through concentrated resource management of virtualized testing environment and automatic execution control of test scripts, we get a better solution to the testing resource utilization and test automation problem. Experiments on CASTE give very appealing results.

***Keywords*—**Software testing, test environment, test script, cloud computing, IaaS, test automation.

I. Introduction

SOFTWARE testing is an important means to ensure software quality. How good it does directly determines how good the software quality is.[1] However, as the number of platforms on which software runs increase and different software versions coexist, the demand for testing environments and testing personnel also increases. For example, to test a software patch or upgrade, the number of testing environments is the product of the number of running environments the software supports and the number of coexisting versions of the software. Obviously it is an expensive invest to build up in ordinary way such a large number of testing environment. Worse still, it becomes more and more difficult to maintain and configure the testing environments.[2-6]

Cloud computing, as a new computing model, has been used in many domains as its advantage in improving IT resource utilization and reducing resource investment by virtualization and on demand provision.[7,8] In our opinion, it is a feasible solution to tackle the above mentioned problems in software testing resources investment and management using cloud computing techniques. In cloud environment, different virtual testing machines can be created for different running environments and different running versions. In such a way physical resource demands can be dramatically reduced. Large number of testing machines can be replaced by a few of machines with a lot of virtual testing machines running on them. By cloud-based automatic testing tool, setting up and configuring testing environments may be an easy job for testers.

In this paper we present the design and implementation of a cloud-based automatic software testing environment, CASTE. CASTE is built upon a private IaaS(Infrastructure as a Service) platform. Through concentrated resource management of virtualized testing environment and automatic execution control of test scripts, we get a better solution to the testing resource utilization and test automation problem.

Fuyang Peng, Bo Deng and Chao Qi are with the Software Division at Beijing Institute of Systems Engineering, Beijing 100101, China (e-mail: fuyang_peng@ sina.com).

II. Design and Implementation

A. The Architecture of CASTE

The architecture of the cloud-based automatic software testing environment (CASTE) we propose is shown in Figure 1. CASTE runs on the private infrastructure cloud Eucalyptus [9, 10] which runs on a server cluster. CASTE mainly consists of a test execution controller TEC, a test database TDB and a set of virtual test environment VMs running on the test unit TU. TEC is responsible for the management of test resources and the execution control of test scripts, including obtaining required virtual machines, deploying test scripts to the appropriate VMs, driving the execution of the test program, collecting test records and results, and preparing test report. TDB stores such information as test data, test status, test log and test results in XML format. The virtual test environments are disk images composing of the hardware capacities (CPUs, memory, disk volumes, network bandwidth, etc.), operating systems, databases and the software to be tested.

TEC is the core of CASTE. It is implemented in three class package: the test engine package, the data access package and the user interface package.

B. Test Engine Package

The test engine package is designed for monitoring and managing the test resources and for deploying and running the test tasks. The main classes in it include CController, CListener and CTestTask.

The CController Class

CController manages the test resources. It controls the test environments by calling the cloud interface. The main control functions include: (1) adding, deleting and renaming host group; (2) connecting, disconnecting and reconnecting to the host; (3) adding, deleting and renaming virtual machine group; (4) operating VM power (power on/off, reset); (5) setting the virtual machines(changing attribute parameter, renaming); (6) saving and restoring VM images.

The CListener Class

CListener monitors the test resources in real time in order to make users know in detail what resources are available. By invoking the cloud environment interface, CListener can communicate with the virtual machines on the TU and instantly

obtain the attribute information of the VMs. It can listen all test resource references and it can also listen in real time to a particular test resource.

CListener compares the obtained information with the relevant data in TDB. If there exist differences, first the CDataAccess class is invoked to update the TDB information, and then the change is feedback to the client

Through listening to specific test resource, testers can know in time the status of the resource, e.g., the number of host servers connected to a user-defined host group, the host server a certain VM resides in, and usage status of a VM.

The CTestTask Class

CTestTask is designed for the test task deployment and execution. It executes the automatic test script of the test task according to the tester's request in two steps, first assigns the test script to the appropriate virtual machine(s), then initiates the execution of the script according to certain strategy. The methods in the class include SetTask, SetDeployFlag, SetDriveFlag, TaskDeploy and TaskDrive.Driving the test script

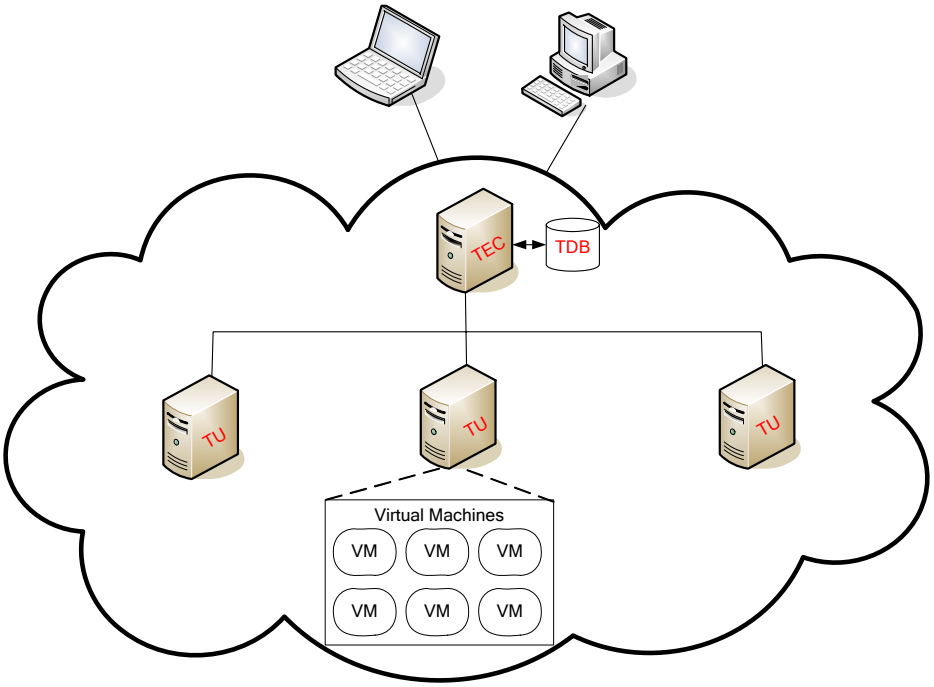

Fig. 1 The architecture of CASTE

needs the help of the methods in CController to change the setting of the virtual machines, as shown in Fig. 2. When the run_script_flag of the virtual machine is set to TRUE, the test script in the VM-shared folder can be driven to run.

C. Data Access Package

CDataAccess is the most important class in the data access package. It is responsible for the interaction between the test engine package and the test database. According to the characteristics of the data to be stored, we use XML format to store them. XMLDOM parser is used to parse the XML files. Detailed description is omitted here due to space limitation.

D. User Interface Package

The main role of the user interface package is to provide a GUI-based control program to help software testers to conveniently and uniformly manage the distributed virtual resources and schedule test tasks to run. Following is its main functions:

1. To dynamically display the information about available resources on resource management interface, including user-defined host groups, hosts, VM groups and the virtual machines installed on a host.

2. To provide the detailed information of each category of resources, such as the number of hosts in a host group, the number of virtual machines in a VM group, the connect status of the host servers, the hardware and software configuration of the virtual machines, and the running status of the VMs. The information can be used to determine which resources can meet the resource requirement of a test.

3. To define test task (including assign test script to VMs). When test case is running, VMs' real time status message obtained through CListener is assigned to the variables of the test cases.

4. To output the host server log file and the test case execution results after the test case finishes execution.

The main classes of the user interface package are CTestResManager, CTestTaskManager and CTestReport.

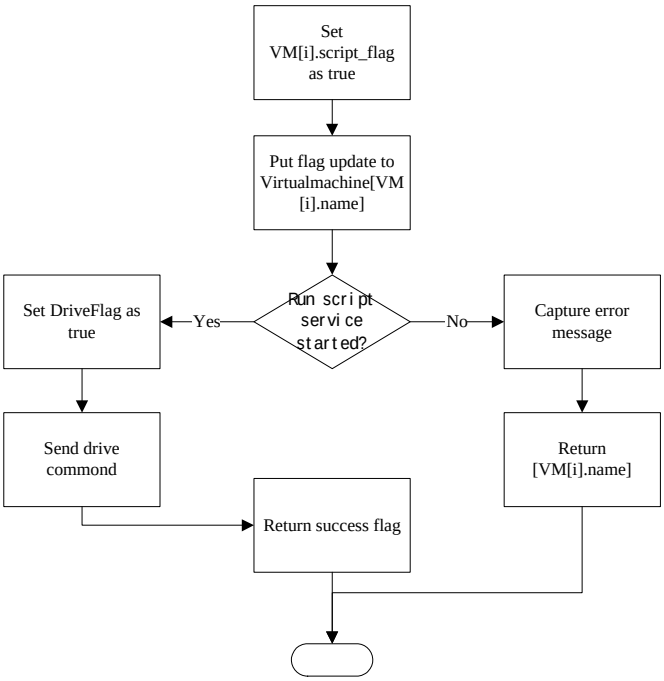

Fig. 2 Control flow in test script execution

The CTestResManager Class

This class is used to centrally manage the distributed virtual test resources through a control interface.

As we define all the resources in a tree-like structure, TreeView control is a natural choice for their display, with the topmost node of it represents the host group,. Users can add host server into the host group. Users can also add VM group into it.

To better manage the various resources, we define control for each test resource. When user clicks the different node types on the TreeView, the CTestResManager can determine the control type based on the node attributes and invokes the corresponding control interface to show the test resource information. The controls can be classified into two types. The first type is the control for showing resource attribute information, the second type is for the uniform management and control of the distributed resources

Using these two types of controls, software testers can get the following information: the VM to be used is installed on which TU, whether the TU is occupied by other tester, whether it's in open state, whether the hardware and software configuration of the VM is appropriate for the test, whether the VM has got an effective IP address, etc.. Software testers can also change the power on/off state of the VMs, or save the snapshot of a running VM.

After obtaining the basic information of the virtual test environment and tuning the state of the VMs, software testers can conveniently select the VM resources to be used in the test and add them to the test tasks.

The CTestTaskManager Class

This class provides a visual interface for software testers to define test tasks, such as assigning automatic test scripts to test VM environment. After test task being defined, software tester can instruct executing of the defined task or deleting of some tasks.

When software tester push the *Renew a task* button, (s)he can select the VM node on the TreeView control, the selected node will be added into the VM set automatically.

After the test VM resources are set ready, software tester can click the *Browse* button to browse and select the script file. When VM list and test script all set, add both to the test task. A test task is stored and an ID is automatically assigned to the task.

It will take several steps to run a test task. First the VMs' images should be restored to the initial state with the help of the CController. Secondly the VM set assigned to the test task is monitored and the obtained information is used to set the script deployment's parameters. And thirdly drive the test script to run.

The whole process of running a test task is transparent to software tester. Software tester don't need to care about the restoration of VMs; images, nor to care about VM's states.

The CTestReport Class

Once a test gets started, the test cases in the test script are executed in the composed order. CASTE will create and output the running log information after the test cases in the script finish running. The CTestReport class will produce the final test results based on the log information and feedback them to the software tester.

Each test task has three state: can't run due to VM failure; test cases being running; test cases being finished. When a test case is finished, it will give one of the two results: Pass or Fail. Software tester can read the execution status of all the test tasks, can check which test tasks are waiting for executing, are executing, or are finished executing, can view visually the ratio of the number of finished test tasks to the number of all test tasks presented to CASTE .

III. Analysis And Evaluation

CASTE has the following advantages:

- Less test environment cost. Using virtualization technology in cloud computing, one can run multiple virtual machines for test environments on one host server, thus reducing hardware purchase cost, system management cost, and the space occupied by the test devices.
- Isolation of virtual machines. The isolation of test virtual machines means crash of one VM will not affect the use of other VMs. It also means severe bugs in the software being tested will not affect the physical hardware and the system software on it.
- Functional snapshot. VM snapshot is a useful means to save and restore test environment status. It can be used in such situations as (1) when rapid restoration of a corrupted test task execution is needed; (2) when a

certain test environment needs to be reused; (3) when a very complex and skillful test environment needs to be saved.

- The test environments are stored as VM image files that can be freely copied or moved. The automatic tool can test software in 24X7.

To verify the effectiveness of CASTE, we select four test tasks to experiment. For each test task, both traditional manual execution method and CASTE-based approach are used. The aim is to compare the performance of the two in terms of test time and the number of test persons participated. Results show that the number of persons participating in testing decreases by 70% and test time reduces by 40%.

REFERENCES

[1] Weinberg, Gerald M. "Quality Software Management: Systems Thinking". Vol 1. Dorset House. 2002.
[2] Kaner, Cem. "Improving the Maintainability of Automated Test Suites." Presented at Quality Week. 1997.
[3] Brad Long，Paul Strooper. "A Case Study in Testing Distributed Systems" DOA '01. Proceedings. 3rd International Symposium on. Distributed Objects and Applications, 2001:20-29.
[4] Hendrickson, Elisabeth. "Making the Right Choice: The Features you Need in a GUI Test Automation Tool." Software Testing and Quality Engineering Magazine (May): 21-25. http://www.qualitytree.com/feature/mtrc.pdf . 1999.
[5] Linz, Tilo and Matthias Daigl. "How to Automate Testing of Graphical User Interfaces." European Systems and Software Initiative Project No. 24306 (June). 1998.
[6] Pettichord, Bret. "Success with Test Automation." Presented at Quality Week (May). 2006.
[7] Velte, A. T., et al., Cloud Computing: A Practical Approach, McGraw Hill, 2010.
[8] Rittinghouse, John W. and Ransome, James F., Cloud Computing - Implementation, Management, And Security, CRC Press, 2010
[9] Nurmi, D., et al., Eucalyptus: A Technical Report on an Elastic Utility Computing Architecture Linking Your Programs to Useful Systems. Tech. Rep. 2008-10, University of California, Santa Barbara, October 2008.
[10] Milojicic D, Wolski R. Eucalyptus: delivering a private cloud [J]. Computer. 2011, 44(4): 102104.