Designing of secure cloud computing method for Software Testing Environment

S.SaravanaKumar[1], K.PazhaniSamy[2], P.DevAnand[3]

[1]*IT Department,*
[2,3]*CSE Department,*
Panimalar Institute Technology, Chennai.

[1]saravanakumars81@gmail.com
[2]pazhani57@gmail.com
[3]dev.panchu@gmail.com

***Abstract*— various information systems are widely used in information society era, and the demand for highly dependable system is increasing year after year. However, software testing for such a system becomes more difficult due to the enlargement and the complexity of the system. In particular, it is too difficult to test parallel and distributed systems sufficiently although dependable systems such as high-availability servers usually form parallel and distributed systems. To solve these problems, to propose a software testing environment for dependable parallel and distributed system using the cloud computing technology, named D-Cloud. D-Cloud includes Eucalyptus as the cloud management software, and Fault VM based on QEMU as the virtualization software, and D-Cloud frontend for interpreting test scenario. D-Cloud enables not only to automate the system configuration and the test procedure but also to perform a number of test cases simultaneously, and to emulate hardware faults flexibly.**

In this paper, present the concept and design of D-Cloud, and describe how to specify the system configuration and the test scenario. Furthermore, the preliminary test example as the software testing using D-Cloud was presented. Its result shows that D-Cloud allows to set up the environment easily, and to test the software testing for the distributed system.

***Keywords*— D-Cloud, QEMU, Eucalyptus, Fault VM, FAU machine.**

I. INTRODUCTION

According to shifting advanced information society, various information systems are used everywhere. Since such systems are closely related to daily life, they must employ highly dependable facilities to avoid undesirable behaviour caused by the underlying bugs and the interference from the external environment. In order to certificate the depend-ability of such systems, these should be tested sufficiently. However, as recent information system becomes larger and more complicated, software testing for such a system be-comes more difficult. In order to check whether components work correctly, tremendous test cases are needed for various input patterns, and environment to execute a great number of tests immediately should be provided. Especially, although highly dependable systems such as high-availability servers likely to form parallel and distributed systems, the testing of large-scale parallel and distributed system is troublesome job in real world after deployment. When a failure occurs in parallel and distributed systems, the reproducibility of the actual system is so poor that the detection of the defective part has been serious problem. On the other hand, a highly dependable system should be equipped with the combination of multiple functions of fault tolerance against hardware faults. Even though testing of fault tolerant facilities should be done under hardware fault conditions or anomaly loads, it is too difficult to destroy a specific part of actual hardware or to concentrate an unrealistic overload in a hardware device. To solve these problems, proposed a software testing environment for reliable distributed systems using cloud computing technology, named "D-Cloud" In this paper, to present the concept and design D-Cloud, discuss the description of the system configuration and the test scenario, and report the preliminary test example using D-Cloud.

II. CONCEPT OF D-CLOUD

A large-scale software testing environment using cloud computing technology for dependable distributed systems, named "D-Cloud." In this section, describe the concept of D-Cloud including the background of this research.

In present information society, as the system scale enlarges and it complicates the behaviour of the system, sufficient software testing has become increasingly harder. Since each test consumes the actual execution time depending on the software size and complexity, and the only way for speedup of software testing process is that a lot of tests should be performed in massively parallel. In order to manage massive computing resources, introduce the cloud computing infrastructure to the software testing.

Meanwhile, the demand for highly dependable system is increasing year after year. In a highly dependable system, fault tolerance is important capability so that the system can tolerate hardware failures and anomaly behaviours. To realize fault tolerance, the system must be formed by the redundant configuration.

Parallel and distributed systems can provide the solution by the redundant resources because of multiprocessor and multiple nodes. However, in this case, the software testing has several serious problems. First, since each process runs in parallel independently, the behaviour of the software may become nondeterministic on the actual hardware. It means that it is too difficult to reproduce the same failure after a failure occurred on such a system. Toward this problem, virtual machine technology helps the reproducibility by adding the management mechanism for the time synchronization. Second, in the case of a large-scale distributed system, to build the test environment becomes impossible. In order to test such a system, usually the preliminary test with restriction is done in the small-scale system, and then the comprehensive test under the full-scale environment is conducted. However, it may stretch the time and raise the cost for the system test unless the test system almost similar to the target environment is prepared. On this point, the cloud services based on IaaS (Infrastructure as a Service) also provide an answer, that is, they permit the use of huge number of computing nodes, and the emulation of entire system without the modification of the source codes using a virtual machine on each node..

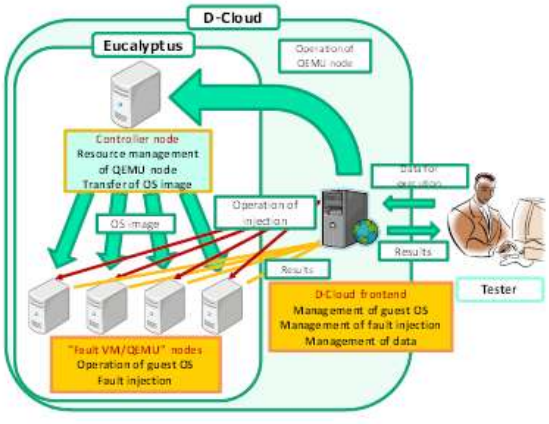

Figure 1. Structure of D-Cloud

Furthermore, although testing of fault tolerant facilities is important in the highly dependable system, it is too difficult to make the specific hardware fault conditions or to generate anomaly loads in real world. The solution of this problem is to use virtual machine technology to provide the fault injection facility, and it can emulate hardware faults of several devices within the virtual machine according to the request from the tester.

Based on above discussions, D-Cloud aims for the realization of the software testing environment as follows:

1) By the use of computing resource provided by the cloud computing system, a number of test case can be performed simultaneously, thus software testing can be accelerated.

2) By the description of the system configuration and test scenario, a series of complex test procedure can be automated.

3) Hardware fault and anomaly state can be emulated flexibly as many times as needed.

4) The target parallel and distributed system can be built onto the cloud computing system, and the execution of the system on the cloud helps the detection of the timing bug and the reproduction of the failure.

In providing various properties of dependability, since an operating system plays a key role, to develop a dependable operating system, which is based on Linux with safe extension mechanism for adding dependable feature as kernel modules, and to provide several components as loadable kernel modules, daemons, and tools. D-Cloud is also useful for the testing of dependable systems using a dependable operating system.

III. D-CLOUD SOFTWARE TESTING ENVIRONMENT

To develop a D-Cloud for software testing environment, D-Cloud consists of multiple virtual machine nodes, which execute guest operating systems with fault injection, a controller node, which controls all of the guest operating systems, and a frontend, which manages the hardware and software configurations and the test scenarios. Figure 1 shows the structure of D-Cloud.

A. Virtual machine with fault injection facility

In D-Cloud, it has been implementing FaultVM based on QEMU as the virtualization software by adding the fault injection facility. The advantages of using QEMU are described below.

- QEMU is open-source software. This allows the modification to the emulation codes of the device for adding the fault injection facility, and the improvement for the reproducibility by adding the management of time synchronization
- QEMU can support various processor architectures. Especially, emulators for several embedded processors such as ARM and SH are already available.
- QEMU can emulate a number of hardware devices. Thus QEMU may treat several hardware faults in the guest OS.

B. Management of computing resources using Eucalyptus

In order to execute many tests simultaneously, a large amount of resources must be managed efficiently and flexibly. Therefore, introduce Eucalyptus as the cloud management software. Eucalyptus is a cloud computing infrastructure that manages machine resources flexibly using a virtual machine, and an open-source implementation having the same API as AmazonEC2.

The roles of Eucalyptus in D-Cloud are shown as follows:

- Management of various guest OS images on the controller node
- Transfer of the specified guest OS images from the controller node to appropriate QEMU nodes
- Beginning and completion of guest operating systems on QEMU nodes

By these features, the tester does not need to be aware of the allocation for computing resources provided by D-Cloud.

C. Automated system configuration and testing

D-Cloud automates the system setup and the test process, including the fault injection, based on a scenario written by a tester. "D-Cloud frontend" manages guest operating systems, configures system test environments, transfers various data from the tester to guest operating systems for the execution of testing, and collects testing results from guest operating systems.

D-Cloud front end performs the following acts:

- Reception of a test scenario, a test program, input data, and a script including execution commands from a tester.
- Interpretation of the test scenario written in XML
- Transfer of the test program, the input data, and the script to the guest operating system
- Issue of the request for the startup of a guest operating system to the Eucalyptus controller node
- Issue of the fault injection command for the target guest operating system to the appropriate virtual machine
- Collection of the output data, logs, and snapshots from the guest operating system.

IV. DESCRIPTION OF SYSTEM CONFIGURATION AND TEST SCENARIO

As described above, D-Cloud performs preparation and test according to a scenario written in XML. By providing multiple scenario files, various systems can be tested simultaneously. Furthermore, since the cloud controller manages the computing resources appropriately, the tester can submit the test items one after another regardless of available computing resources.

Testing scenario statement consists of four parts as follows.

- machine Definition: Descriptions for the hard-ware configuration

TABLE I: MACHINE DEFINITION ELEMENT

Element name	Meaning
machine	Delimiter for definition of the hardware
name	Name definition of the hardware environment
Cpu	Number of CPUs
Mem	Size of memory
Nic	Number of NICs
Id	ID of the used OS image

TABLE II: SYSTEM DEFINITION ELEMENT

Element name	Meaning
system	Delimiter for definition of the software environment
name	Name of the software environment
host	Delimiter of the testing host
hostname	Name of the host
machine name	Name of the used machine element
config	Designation of the configuration file

- System Definition: Descriptions for the software environment.
- Injection Definition: Definitions of faults for injection.
- Test Definition: Procedures of the entire test.

A.. Configuration for the hardware environment

The description of the hardware configuration is given by the "machine Definition" element. Table I lists the contents of the "machine Definition" element. All hardware components used in the test must be defined by each "machine" element. The "machine" element must include five elements, "name," "cpu," "mem," "nic," and "id." The "name" is referred in the "system Definition" element described in the following sub-section. The "cpu" and "nic" indicate the number of CPUs and NICs, respectively, and "mem" represents the allocation size of the main memory. The "id" element designates the identifier for the system image to be used. Eucalyptus provides each system image with a unique identifier in the cloud system, and the identifier is also used in D-Cloud.

B. setting for the software environment

The description of the software environment is given by the "system Definition" element containing elements shown in Table II. The entire software environment used in the test must be defined by each "system" element. The "system" element must include two elements, "name" and "host." The "name" is referred in the "test Description" element. Moreover, the "host" element contains three elements, "host-name," "machine name," and "config." The "hostname" determines the name of the host; the "machine name" is selected from the "name" of "machine" within the "machine Definition" element. The "config" designates a file containing the various kinds of parameters.

TABLE III: INJECTION DEFINITION ELEMENT

Element name	Meaning
injection	Delimiter for definition of the fault injection
name	Name definition of the fault injection
Fault	Delimiter for configuration of the injection
location	Designation of device type
target	Designation of target device
kind	Type of fault
time	Duration of the fault event

TABLE IV: TYPES OF FAULT INJECTION

Device	Fault	Value
Hard disk	Specified sector returns error	bad block
	Specified sector is read-only	read-only
	Error is detected by ECC	ecc
	Received data contains error	corrupt
Network	1bit error of packet	1bit
	2bit error of packet	2bit
	Error is detected by CRC	crc
	Packet loss	loss
Memory	Bit error	Bit
	Byte at specified address	

C. Definition of fault injection

The definition of fault injection items is given in the "injection Definition" element containing elements shown in Table III. It may have multiple "injection" elements, each of which has a "name" element and multiple "fault" elements. The "injection" element is assigned to each fault injection event. The "name" is referred in the "test Description" element. The "fault" element must include four elements, "location," "target," "kind," and "time." The "location" and "target" specify the target device type and device name to inject a fault, respectively. The "kind" indicates the selection of fault injection elements listed in Table IV. The "time" represents the duration of fault injection.

D. Description for the automatic test procedures

The execution of the test is described in the "test Definition" element using the contents shown in Table V. The "run" element is used for the independent test definitions, and multiple "run" elements may exist in a "test Definition" element. The "name" element defines the name of the system test to be performed. The output file containing test result is created with the file name based on the content of "name" element. The "system name" indicates the name in the "system Definition" element. The "halt" element with "when" attribute decides the finish time of the entire system test. The "script" element includes four

elements, "on," "put File," "exec," and "inject" for each needed host. The "on" specifies the host name defined in the "system Definition" element. The "put File" and "exec" specify the file name for the transfer to the host and the execute command,

TABLE V: TEST DEFINITION ELEMENT

Element name	Meaning
Run	Delimiter for definition of the test scenario
Name	Name of the test scenario
system name	Name of the used system element
Halt	Ending time of the test
Script	Delimiter for definition of
On	Execution host
put File	File transmitted to the guest OS
Exec	Designation of the script file
Inject	Execution of the fault injection

Respectively, the "inject" is selected from the name defined in the "injection Definition" element. The "inject" element also has "when" attribute, which specifies the duration of the fault incidence.

In addition to the description by XML, consider the support for building the system environment and for the execution of the system testing by introducing the dynamic scripting language. This supplement helps the tester perform the desired test easily and flexibly. Moreover, by the use of the scripting language, the stylized description may improve the portability of the test process.

V. PRELIMINARY TEST EXAMPLE USING D-CLOUD

Preliminarily evaluate D-Cloud by testing the actual dependable system. It have proposed and developed a fault tolerant and high-performance interconnection network based on the multi-link of Gigabit Ethernet (GbE) named RI2N (Redundant Interconnection with Inexpensive Network) Here, to assume simplified system using RI2N. Client1 is connected with server1 by two Ethernet links, network0 and network1. In this case,

network0 and network1 form the RI2N logical link. Network2 is also available for issuing the command from D-Cloud frontend to each node and the collection of measurement results to D-Cloud frontend. Moreover, to assume the test scenario as follows;

1) Client 1 performs burst data transfer to server 1 using RI2N continuously. In this case, throughput is expected to be twice as high as single link.
2) After 200 seconds from the power-on, the network interface "eth0" of client1 is down during 60 seconds. RI2N link will be down immediately, however, throughput should recover to the level of the single link after a few seconds.
3) After that, "eth0" interface on client1 is alive again. RI2N will detect the link recovery, and throughput should recover to the same level as in the beginning condition.
4) Finally, the system is halted 300 seconds after the power-on.

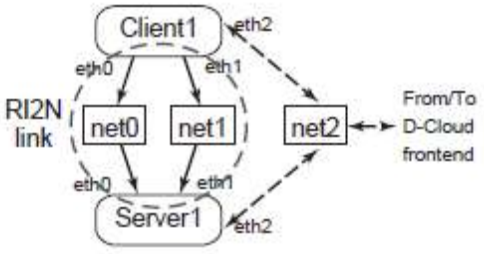

Figure 2. Simplified system example using RI2N

Based on this scenario, the description by XML can be denoted. It is notable that step 2 can be expressed as the fault injection of the packet loss against eth0 of client1.

To demonstrate the part of the web interface for the management of test scenarios in D-Cloud, and it shows that three test scenarios (nic0.xml, nic1.xml, and nic2.xml) are running simultaneously on D-Cloud.

To indicate the results obtained by the above scenario. Red arrow indicates the duration of the fault injection (60 sec.). In this result, when the fault is injected to eth0 of client1, throughput falls transiently, and soon throughput recovers to lower level than before. After eth0 is alive again, with a few seconds of delay, the throughput recovers to the same level as in the original condition. The absolute values of the throughput are incorrect in current D-Cloud. It is because each packet transfer is performed via real network while the behaviours of client1 and server1 are emulated within each virtual machine. Even

though, confirm that the fault tolerant and recovery detection capability of RI2N work correctly by relative tendency of the results.

VI. RELATED WORKS

Recently, Large-scale software testing has been studied. Grid Unit executes software tests automatically on the grid by distributing the execution of JUnit test suites with minimum user intervention. Grid Unit is naturally limited to the execution of JUnit test code by Java. When test nodes are crashed and stopped in Grid Unit, they cannot execute remaining program tests. ETICS also provides automated test environments for grid and distributed software on a grid computing platform using Condor as a workload management system. Unlike D-Cloud concept, uses a cloud computing environment, and enables to create and execute VM instances for program tests through a web portal. Cloud is proposed as a cloud computing facility for software testing, and performs parallel symbolic execution based on the source code.

```
<jobDescription>
    <machineDefinition>
      <machine>
        <name>server</name>
        <cpu>1</cpu> <mem>512</mem> <nic>3</nic>
        <id>emi-1D8C0CAA</id>
      </machine>
      <machine>
        <name>client</name>
        <cpu>1</cpu> <mem>512</mem> <nic>3</nic>
        <id>emi-0ACC0C2D</id>
      </machine>
    </machineDefinition>
<systemDefinition>
      <system>
        <name>systemA</name>
        <host>
          <hostname>server1</hostname>
          <machinename>server</machinename>
          <config>serv. conf</config>
        </host>
        <host>
          <hostname>client1</hostname>
          <machinename>client</machinename>
          <config>client.conf</config>
        </host>
      </system>
    </systemDefinition>
    <injectionDefinition>
      <injection>
        <name>injectionA</name>
        <fault>
          <location>network</location>
          <target>eth0</target>
          <kind>loss</kind>
          <time>60</time>
        </fault>
      </injection>
    </injectionDefinition>
    <testDescription>
      <run>
        <name>testA</name>
        <systemname>systemA</systemname>
        <halt when="300">down</halt>
        <script>
          <on>client1</on>
          <putFile>test.sh</putFile>
          <exec>test. sh</exec>
          <inject when="200">injectionA</inject>
        </script>
      </run>
    </testDescription>
</jobDescription>
```

Figure 3. Example test scenario for RI2N by XML

On the other hand, fault injection techniques in program tests have been proposed. DOCTOR is a software fault injector, which supports memory faults, CPU faults, and communication faults. Although software fault injection needs modification of the source codes to be tested, this approach need not modify the source codes at all for fault injection. FAU machine performs a software test using virtual machines for fault injection mechanism. However, since FAU machine does not provide an automated test environment, the tester must configure the test environment manually.

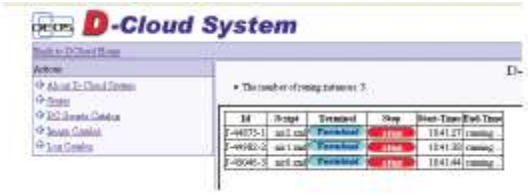

Figure .4 Current management screen of D-Cloud

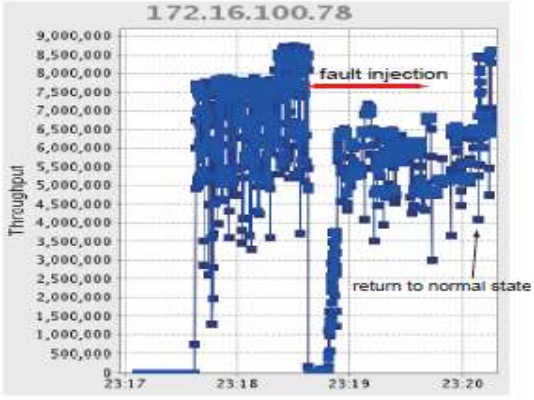

Figure. 5 Test results obtained by D-Cloud

VII. CONCLUSION AND FUTURE WORK

To present the concept and design of the software testing environment using the cloud computing technology, named D-Cloud. D-Cloud permits the automatic configuration, testing with fault injection along the description of the testing scenario. It has been developing D - Cloud using Eucalyptus as a cloud management software and QEMU as virtualization software. As the software testing using D-Cloud, the preliminary test example was denoted, and the result demonstrated that D-Cloud allows to set up the environment easily, and to test the software testing for the distributed system. At present, D-Cloud can obtain the testing results including the virtual console logs and the syslog outputs by the running processes and operating system in FaultVM/QEMU on each node. In general use, it should consider more sophisticated way to gather the results and detect the fault from large amount of logs.

In future work, it should append the management mechanism to D-Cloud for keeping reproducibility by time synchronization in coarse grain among related virtual machines without sacrificing the performance. Further, to introduce the model simulator written by the system description language to D-Cloud in order to test various systems including embedded systems with proprietary hardware's.

In proposed DS-Bench as a dependability benchmarking framework for a dependable operating system. D-Cloud is so useful as the virtual platform for DS-Bench since anomaly loads can be generated automatically from the request given by the scenario file using D-Cloud.

REFERENCES

[1] Large-Scale Software Testing Environment using Cloud Computing Technology for Dependable Parallel and Distributed Systems. Toshihiro Hanawa, Takayuki Banzai, Hitoshi Koizumi, Ryo Kanbayashi, Takayuki Imada, and Mitsuhisa Sato Department of Computer Science Center for Computational Sciences University of Tsukuba

[2] T. Banzai, H. Koizumi, R. Kanbayashi, T. Imada, H. Kimura, T. Hanawa, and M. Sato, "D-Cloud: Design of a software testing environment for reliable distributed systems using cloud computing technology," in Proc. 2nd International Symposium on Cloud Computing (Cloud 2010) in conjunction with CCGrid2010, May 2010, (To be appeared).

[3] Y. Ishikawa et al., "Towards an open dependable operating system," in Proc. 12th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, Mar. 2009, pp. 20–27.

[4] Nurmi et al., "The eucalyptus open-source cloud-computing system," in Proc. 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid '09), 2009, pp. 124–131.

[5] Amazon elastic compute cloud (Amazon EC2).[Online].Available: http://aws.amazon.com/ec2/
[6] S. Miura, T. Hanawa, T. Yonemoto, T. Boku, and M. Sato, "RI2N/DRV: Multi-link Ethernet for high-bandwidth and fault-tolerant network on PC clusters," in Proc. The 9th Workshop on Communication Architecture for Clusters (CAC) in IPDPS, May 2009, pp. 1–8.
[7] Duarte, W. Cirne, F. Brasileiro, and P. Machado, "GridUnit: software testing on the grid," in Proc. 28th international conference on Software engineering (ICSE '06), 2006, pp. 779–782.
[8] M.-E. Begin et al., "Build, configuration, integration and testing tools for large software projects: ETICS," in Proc. Rapid Integration of Software Engineering Techniques, ser. Lecture Notes in Computer Science, vol. 4401, Sep. 2007, pp. 81–97.
[9] Open Solaris test farm. [Online]. Available: http://opensolaris. org/os/community/testing/testfarm