Moonstone: A Framework for Accelerating Testing of Software

Atsuji Sekiguchi, Tomohiro Ohtake, Toshihiro Shimizu,
Yuji Hotta, Taichi Sugiyama, Takeshi Yasuie and Toshihiro Kodaka
Cloud Computing Research Center
FUJITSU LABORATORIES LIMITED
Kawasaki, Japan
e-mail: {sekia, ohtake.tomohiro, shimizut, yhotter, sugiyamataichi, yasuie.takeshi, and tkodaka}@jp.fujitsu.com

***Abstract*—Enterprises must speed up software development and releases so that they can quickly verify business ideas. We have developed a framework called "Moonstone" that can be used to speed up the testing that is included in a release operation. Moonstone has the following two functions to support testing. 1) Function to construct test environment: this function is used to automatically construct test and production environments on a cloud platform. This function uses hint information of a system configuration included in source code and configuration files, and templates of system configurations. 2) Function to prepare and execute test: this function is used to automatically create and run test scenarios by replaying captured network packets. Because testing in a release operation phase can be done efficiently with these functions, the time required for a release operation can be reduced. We used Moonstone in a trial environment and obtained the following results: 1) a reduction of more than 80% of the time required for the construction of a test environment, 2) a reduction of 33% of the time required for the testing.**

Keywords-continuous delivery; software development; software test; cloud platform; traffic replay

I. INTRODUCTION

Global business competition has been intensifying with the spread of Internet and cloud computing [1]. Enterprises must therefore realize business ideas as products and services and then improve them so that they can survive this competition.

Lean Startup [2] describes a method of carrying out product development by verifying business ideas (hypothesis) quickly in the market. In this method, the effects of each hypothesis are quantitatively measured while verifying one hypothesis at a time in the market. An enterprise can learn which hypothesis is effective because the method can individually measure the effect of each hypothesis. Because one hypothesis is verified in the market in a certain period, the amount of development needed to prove the hypothesis decreases, and development can be sped up. Moreover, because the product or service can be quickly released to the market with this method, the enterprise can change its business direction according to the result of the verification.

However, there is the following problem in practicing "lean startup" in software development and release. The release operation contains testing and deployment processes [3]. Testing is done to inspect software, configurations, and environments from functional [4] and non-functional [5] aspects (such as execution time, performance, quality of service, and security). Deployment is carried out to distribute software and configurations to test and production environments. Even if the system will be slightly changed, testing of the entire system is required to confirm that the changed system will run correctly. Thus, the release operation imposes a constant workload, which is not in proportion to the amount of development. If the amount of development is not changed, the workload increases when the number of releases increases. Thus, the hypothesis cannot be verified quickly.

Therefore, we have developed a framework called "Moonstone" to speed up the release operation. Moonstone has the following two functions to support testing. 1) Function to construct a test environment: this function is used to automatically construct test and production environments on a cloud platform. This function uses hint information of a system configuration included in source code and configuration files, and templates of system configurations. 2) Function to prepare and execute a test: this function is used to automatically create and run test scenarios by replaying captured network packets. Because testing in a release operation phase can be done efficiently with these functions, the time required for the release operation can be reduced. Furthermore, reproducibility of testing can be increased.

The rest of this paper is organized as follows. First, we describe the Moonstone architecture in Section II. Next, we explain the method and result of an evaluation in a trial experiment that uses Moonstone in an environment on a cloud platform in Section III. In Section IV, we discuss related work. Finally, we end with the conclusion and future work in Section V.

II. MOONSTONE ARCHITECTURE

To support phases from development to release of applications, Moonstone can cooperate with various functions such as an issue tracker (Redmine [6]) for development task management, a version control system (Subversion [7] or git [8]) for management of an application's source code, and a continuous integration [10] tool (Jenkins [9]) as shown in Figure 1. In addition, to speed up the testing, Moonstone has two functions: a function to construct a test environment, and a function to prepare and

 ISBN: 978-1-61208-271-4

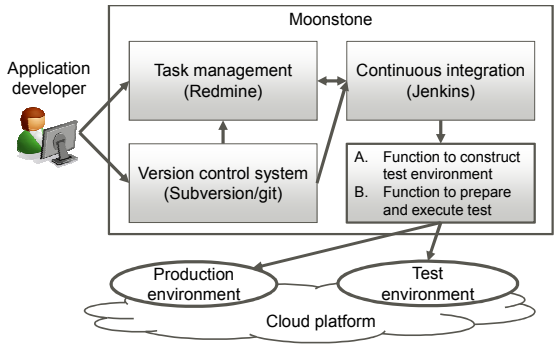

Figure 1. Moonstone architecture

execute a test. Moonstone can realize continuous delivery [11] with these cooperative actions and functions.

A. Function to construct test environment

Application developers generally develop their application in consideration of the characteristics of a system (such as a web server type, an application server type, a database server type, or their connectivity), but they cannot change configurations of the system without permission (such as changing the configuration of a database server from one server to a master-slave configuration). System construction operators are responsible for changing the system. To cooperate with the system construction operators, the application developers must make many preparations such as documentation. Thus, system configuration cannot be changed quickly when a change of system configuration is needed because of the growth of the application.

We have developed a function that extracts the characteristics of an application from its source code and configuration files, and automatically constructs a system by using the most suitable template for the characteristics. The proposed function enables the system to be changed quickly because the application developers can change the system by themselves without needing to have cooperation from the system construction operators.

We will explain the flow of processing on the basis of Figure 2.

1. Application characteristics extraction: In this process, the proposed function extracts the characteristics of an application from its source code and configuration files by using extraction patterns. In many cases, when middleware is used in a system, descriptions for the middleware exist in the source code and the configuration files (e.g., if JDBC (Java Database Connectivity) [12] and MySQL [13] is used in the system, the name of JDBC driver for MySQL appears in the source code or the configuration files, such as "*com.mysql.jdbc.Driver*"). So, to detect the used middleware, we wrote a pattern beforehand to detect the middleware as an extraction pattern (e.g., detection of "*com.mysql.jdbc.Driver*" in the source code or the configuration files). As a result, the proposed function can detect the characteristics of middleware composition such as a mail server, a message queuing server, and a cache server. Moreover, we wrote hint information in the source code beforehand for the characteristics that could not be detected by the above-mentioned process. The proposed function only supports the Java language currently. In Java, the function utilizes some of the annotations [14] in the source code as hint information. For instance, the hint information is described as the ratio of reading and writing in a class that operates the database, such as *"@ReadWriteRatio"*. The proposed function extracts the middleware used and the hint information as characteristic information of the application.

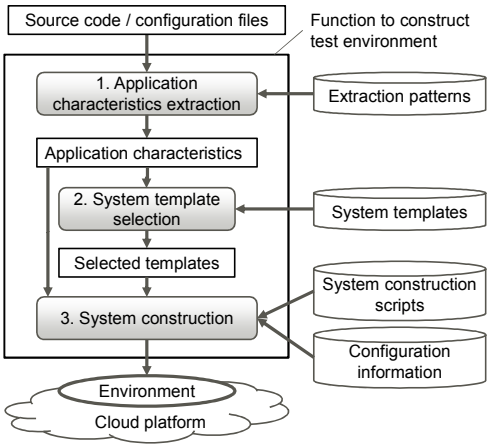

Figure 2. Function to construct test environment

2. System template selection: We defined system configuration patterns (such as AWS Cloud Design Pattern [15]) on a cloud platform as "system templates" beforehand. Each system template contains its characteristic information. In this process, this function selects a template that matches the characteristics of the application. For instance, when a name of a database is detected from the configuration files, the proposed function selects a template that installs and sets up the database (e.g., *"MySQL"*). When hint information such as *"@ReadWriteRatio (value = 5.0)"* (means "the reading frequency of the database is five times the writing") is extracted from the source code, the proposed function decides on a template of a composition that suits such purpose (e.g., a database template for a master-slave configuration).

3. System construction: The proposed function constructs a system on the basis of the selected system template. In each system template, we associated the template with scripts that automatically construct the system beforehand. We wrote construction scripts for each server type such as Web, application, and database using Chef [16]. For example, in a case of Web server, our construction script prepares virtual machines (VMs) through the cloud platform's API, and deploys Web server program (e.g., httpd) and contents to each VM.

This function is useful not only for constructing a test environment for confirming an application behavior but also for constructing a production environment. Furthermore, this function can increase the recyclability of the automation scripts because the function calls the scripts to suit the application's characteristics.

 ISBN: 978-1-61208-271-4

B. *Function to prepare and execute test*

A system that has begun providing services for customers should be stable. When an application or its system will be changed, the correctness of the behavior of the application should be checked (tested). To test operations in the entire system, load test tools [17] or devices [18] are usually used to run a test efficiently. The test contains two steps: preparation and execution. In the preparation step, these tools or devices require preparation of test scenarios. In the execution step, the test is executed on the basis of the test scenarios, and success or failure is judged from the results of the test scenarios. The test scenario consists of definitions such as access patterns to the system, request messages for access to the system, and response messages that the system should make in response to the requests. There is a problem that making test scenarios requires a great deal of skill and much time.

Therefore, we have developed a function that automates the test scenario making and the test execution [19]. In the preparation step, a module of this function captures network packets of the request/response messages that are exchanged between clients and servers in the production environment. The request/response messages and their access timing are used as the test scenarios. In the test execution step, the proposed function replays the request messages in the captured packets and compares response messages between the captured packets and the test environment. While replaying, the proposed function translates network/application data of the packets from data in the original environment to data in the test environment. Thus, because the test scenario can be made without a need for much skill or time, the test can be sped up. We implemented the proposed function as a C program on Linux.

We will explain the flow of processing on the basis of Figure 3.

1. Packet capture: In this process, the proposed function captures packets between clients and servers of the system using a packet capture tool (such as Wireshark [20]). At this time, the environment of servers is the production or the test. Users of IaaS (Infrastructure as a Service) [1], which is one of the cloud platforms, cannot capture the packet in the network layer because the network layer is usually hidden in IaaS. Thus, the proposed function captures the packets in each server and collects them. The packet capture tool can capture packets on various networks (such as Ethernet or Infiniband) because it captures packets OS obtained.

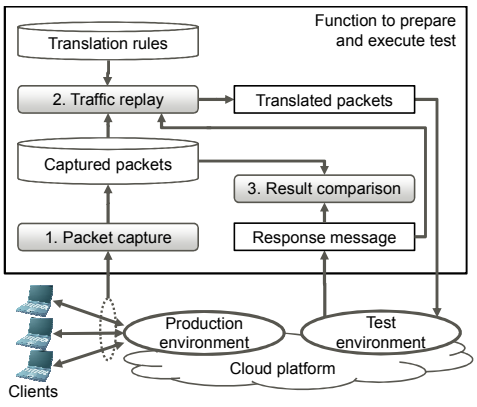

Figure 3. Function to prepare and execute test

2. Traffic replay: The proposed function generates access loads for the test in the test environment. The access loads are generated based on the test scenario that consists of the request/response messages and their access timing on the captured packets. While sending the packets, the proposed function translates the packets from environment-dependent data in the captured packets to data for the test environment. The translation is based on translation rules. This rule is a definition of how to rewrite data. The data are of the network layer such as Ethernet/IP headers, TCP connections, and HTTP headers, and of the application layer such as session information and authentication tokens. When a response of a server contains session information, the proposed function dynamically rewrites sending packets with the information. We made rules beforehand to match the captured packets to the test environment.

3. Result comparison: The proposed function compares two response messages of the test environment and the response messages in the captured packets. The function regards the test as a success when these responses are the same, and regards the test as a failure when these responses are different. The function can also compare both of access timing of the response messages, and can regard the test as a success when the response time of the test response is less than a threshold time.

When these two functions are combined, the test can be executed on demand. Thus, when testing is needed, the proposed functions construct the test environment on a cloud platform, execute the test, and return the environment after finishing the testing.

III. EVALUATE

To evaluate the two functions described in Section II, we tried these functions in the following environment.

A. *Evaluation environment*

This environment is a system on a cloud platform to provide a service for consumers through the Internet. The system was constructed for starting the service. The system uses FGCP/S5 (Fujitsu Global Cloud Platform) [21], which is one of IaaS. The system is a typical three-tier one that consists of tens of VMs such as Web servers, application servers, database servers, and load balancers.

B. *Evaluation method*

To examine the proposed function described in II.A, we constructed a test environment of the same composition as a production environment with the proposed function and via manual operation respectively, and measured the respective elapsed times. Manual operation was executed by experts, who are the system construction operators. We calculated the reduction time and reduction rate by using the proposed function from the result. The construction targets were Web servers, application servers, and databases. Strictly, installed software and settings of each server were a little different

 ISBN: 978-1-61208-271-4

from each other. Then, we grouped the tens of servers to three types with the rough role such as Web. Constructing these servers involves starting VMs, making settings for an OS, installing and making settings for middleware such as HTTP server, Java EE server and database server, and installing Web contents and applications. We measured the elapsed time of construction in each type of server. We calculated the average value for the same type of server. The elapsed time of the system was calculated from the type of server and each number.

To examine the proposed function described in II.B, we made and ran the test with the function and via manual operation respectively, and measured the respective elapsed times. Manual operation was also executed by the experts. We calculated the reduction time and reduction rate by using the proposed function from the result. The test contained many test scenarios to confirm the correctness of software, configurations, and environments from functional [4] and non-functional [5] aspects (such as execution time, quality of service, security, usability, and safety). For the manual operation, we made the test scenarios for JMeter [17]. To use the proposed function, the function makes it possible to make the test scenarios automatically by capture and replay of the traffic between clients and servers in the production environment. However, in this trial, we made the test scenarios as follows to clarify a comparison in these two cases: we manipulated a Web browser through each test scenario in manual operation, captured packets through the manipulation, and treated the packets as each test scenario.

C. Evaluation result

The results of the evaluation of II.A are shown in Table I. “Server type” means the type of server such as Web, application, or database. “Elapsed time (manual) [A]” means the elapsed time in the case of manual operation. “Elapsed time (trial) [B]” means the elapsed time in the case of using the proposed function. “Reduction time” is the difference between the elapsed time (manual) and the elapsed time (trial), and is calculated as A−B. “Reduction rate” is the ratio of the reduction time to the elapsed time (manual), and is calculated as (A−B)/A.

An elapsed time of 80% or more was able to be reduced in any server type as shown in Table I. The reduction time of one server was between 4.5 and 9 hours. Because the environment consisted of tens of servers, we were able to reduce the elapsed time of hundreds of hours in total.

The result of the evaluation of II.B is shown in Table II.

TABLE I. SERVER CONSTRUCTION TIME, REDUCTION TIME, AND REDUCTION RATE

Server type	Elapsed time (manual) [A]	Elapsed time (trial) [B]	Reduction time [A−B]	Reduction rate [(A−B)/A]
Web	5h 40m	1h 7m	4h 33m	80%
Application	10h 46m	1h 48m	8h 58m	83%
Database	8h 17m	0h 55m	7h 22m	89%

TABLE II. ELAPSED TIME OF TEST, REDUCTION TIME, AND REDUCTION RATE

Operation type	Elapsed time (manual) [A]	Elapsed time (trial) [B]	Reduction time [AB]	Reduction rate [(A−B)/A]
Test scenario making	40h	16h	24h	60%
Test execution	32h	32h	0h	0%
Total	72h	48h	24h	33%

“Operation type” means an operation of the test such as test scenario making and test execution, and also contains their total. “Elapsed time (manual)”, “Elapsed time (trial)”, “Reduction time” and “Reduction rate” mean the same as those in Table I.

An elapsed time of 60% (24 hours) was able to be reduced in the making of test scenarios. The elapsed time of the test execution was the same as the manual operation, and the reduction rate was 0%. As a result, the elapsed time of 33% (24 hours) was able to be reduced in total.

IV. RELATED WORK

Continuous integration [10] is the practice of enhancing the quality of source code by automatically integrating, compiling, and testing the source code every day during development. Continuous delivery [11] is a practice that automates deployment of the application in addition to continuous integration. Tools that support these practices exist [9][22]. However, activities such as constructing a test environment and making a test scenario are not being offered by those tools.

There are several approaches to constructing an environment. Tools to support automation of the construction exist [16][23], and a method to automatically construct an environment from a policy-based environment definition is also known [24]. Those approaches require a great deal of skill and much time, because the user has to describe the definition of the composition of the environment correctly. Our method detects the characteristics of the application by using information on system configuration contained in source code and configuration files, and hint information written in the source code as annotations [14]. Then, the method determines the most suitable system template for the characteristics, and automatically constructs an environment based on the selected template. In the case of our method, the user does not need to describe the definition of the composition of the environment.

An approach of the abstraction of clouds' API [25], and an approach of the definition of common data model [26] exist. Though we used the cloud's own API and data model, these approaches can be helpful for portability.

Tools to help automate the making of the contents of the testing exist [17][27][28]. These require a great deal of skill and much time to create a test scenario for the automation. Moreover, it is difficult to imitate real clients’ traffic load patterns. Therefore, because some problems are often

 ISBN: 978-1-61208-271-4

overlooked, serious troubles occur when the application runs on the production environment. Traffic replay is an approach that captures packets in the production environment and uses the captured packets for testing [29][30][31][32][33]. However, it is not easy to conduct traffic replay in an ad hoc test environment on a cloud platform. Because the test environment is different from the environment in which the packets are captured, various parameters such as MAC addresses, IP addresses, or TCP port numbers are different. Packets cannot reach servers if the captured packets are simply replayed. In the case of testing an application, application-specific information such as HTTP session IDs and timestamps should be also adjusted to the environment and the time of the testing. Our method can test a long transaction by carrying out a traffic replay with a packet conversion by using not only the difference between these environments but also application-specific information.

V. CONCLUSION AND FUTURE WORK

We have developed a framework called "Moonstone" for speeding up testing that is included in a release operation. Moonstone has two functions: 1) a function to construct a test environment, and 2) a function to prepare and execute a test. In our trial, we confirmed that these functions make it possible to reduce the elapsed time for the testing.

We will tackle the following problems in the future. In 1), when a production environment has a lot of servers, a long elapsed time is required to construct a test environment that is similar to the production environment. In 2), when the proposed function replays captured packets, we must synchronize databases of the test environment beforehand. When the size of the databases is large, the synchronization requires a long elapsed time.

REFERENCES

[1] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition", ACM SIGCOMM Computer Communication Review, Vol. 39 Issue 1, Jan. 2009, pp. 50-55.

[2] E. Ries, "The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses", Crown Business, 2011.

[3] Cabinet Office, "ITIL Service Transition 2011 Edition (Best Management Practices)", The Stationery Office, 2011.

[4] W. E. Howden, "Functional Program Testing", Software Engineering, IEEE Transactions on, Vol. SE-6, Issue 2, Mar. 1980, pp. 162-169.

[5] W. Afzal, R. Torkar, and R. Feldt, "A systematic review of search-based testing for non-functional system properties", Information and Software Technology, Vol. 51, Issue 6, Jun. 2009, pp. 957-976.

[6] Redmine, Available: http://www.redmine.org/, retrieved: Mar. 2013.

[7] Subversion, Available: http://subversion.apache.org/, retrieved: Mar. 2013.

[8] git, Available: http://git-scm.com/, retrieved: Mar. 2013.

[9] Jenkins, Available: http://jenkins-ci.org/, retrieved: Mar. 2013.

[10] P. M. Duvall, S. Matyas, and A. Glover, "Continuous Integration: Improving Software Quality and Reducing Risk", Addison-Wesley Professional, 2007.

[11] J. Humble and D. Farley, "Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation", Addison-Wesley Professional, 2010.

[12] JDBC (Java Database Connectivity), Available: http://www.oracle.com/technetwork/java/javase/jdbc/index.html, retrieved: Mar. 2013.

[13] MySQL, Available: http://dev.mysql.com/, retrieved: Mar. 2013.

[14] Java Annotations, Available: http://docs.oracle.com/javase/1.5.0/docs/guide/language/annotations.html, retrieved: Mar. 2013.

[15] AWS Cloud Design Pattern, Available: http://en.clouddesignpattern.org/, retrieved: Mar. 2013.

[16] Chef, Available: http://www.opscode.com/chef/, retrieved: Mar. 2013.

[17] JMeter, Available: http://jmeter.apache.org/, retrieved: Mar. 2013.

[18] Avalanche, Available: http://www.spirent.com/Products/Avalanche/Avalanche_Latest_Release, retrieved: Mar. 2013.

[19] T. Sugiyama, T. Yasuie, and Y. Nomura, "A Study for System Verification with Captured Packets" [in Japanese], Proc. of the Society Conference of IEICE 2011 Communication (2), Aug. 2011, p.392.

[20] Wireshark, Available: http://www.wireshark.org/, retrieved: Mar. 2013.

[21] FGCP/S5 (Fujitsu Global Cloud Platform), Available: http://welcome.globalcloud.global.fujitsu.com/, retrieved: Mar. 2013.

[22] IBM SmaterCloud Continuous Delivery, Available: http://www-142.ibm.com/software/products/us/en/continuousdelivery/, retrieved: Mar. 2013.

[23] Puppet, Available: https://puppetlabs.com/puppet/puppet-open-source/, retrieved: Mar. 2013.

[24] A. Sahai, S. Singhal, R. Joshi, and V. Machiraju, "Automated Policy-Based Resource Construction in Utility Computing Environments", Proc. of Network Operations and Management Symposium, Vol. 1, Apr. 2004, pp. 381-393.

[25] Deltacloud, Available: http://deltacloud.apache.org/, retrieved: Mar. 2013.

[26] Cloud Infrastructure Management Interface (CIMI) Model and RESTful HTTP-based Protocol, Available: http://dmtf.org/sites/default/files/standards/documents/DSP0263_1.0.1.pdf, retrieved: Mar. 2013.

[27] JUnit, Available: http://junit.org/, retrieved: Mar. 2013.

[28] Selenium, Available: http://seleniumhq.org/, retrieved: Mar. 2013.

[29] Tcpreplay, Available: http://tcpreplay.synfin.net/, retrieved: Mar. 2013.

[30] S. Hong and S. Wu, "On Interactive Internet Traffic Replay", Proc. Eighth International Symposium on Recent Advances in Intrusion Detection, Sept. 2005, pp. 247-264.

[31] IBM Rational Test Virtualization Server, Available: http://www-01.ibm.com/software/rational/products/rtvs/, retrieved: Mar. 2013.

[32] CA LISA, Available: http://www.ca.com/us/products/detail/CA-LISA.aspx, retrieved: Mar. 2013.

[33] Oracle Enterprise Manager: Application Quality Management, Available: http://www.oracle.com/technetwork/oem/app-quality-mgmt/index.html, retrieved: Mar. 2013.

 ISBN: 978-1-61208-271-4