IVRIDIO: Design of a Software Testing Framework to Provide Test-first Performance as a Service

Alim Ul Gias*, Asif Imran*, Rayhanur Rahman* and Kazi Sakib†
Institute of Information Technology, University of Dhaka
Ramna, Dhaka–1000
*{bit0103, bit0119, bit0101}@iit.du.ac.bd
†sakib@univdhaka.edu

***Abstract*—Test-first Performance (TFP) is a testing paradigm that focuses on performance testing from the early stage of development. For performance oriented applications like a web service, TFP approach can reduce the overall cost of software testing. Given this potential benefit, TFP is yet to be incorporated in existing cloud testing frameworks. This paper proposes the design of a testing framework which introduces TFP as a Service (TFPaaS) named as IVRIDIO. It includes the Plugin for TFP in the Cloud (PTFPC) that will provide instant feedbacks to fix critical performance issues. To simplify the TFPaaS availing procedure, the Convention over Configuration (CoC) design paradigm has been applied. A configurable project template is designed using the CoC to maintain TFP test cases. Furthermore, necessary directions are given to prototype the framework as a testbed for related research.**

***Index Terms*—Cloud Testing, Test-first Performance, Convention over Configuration**

I. Introduction

Test-first Performance (TFP) [1] is a performance testing methodology with an approach similar to Test Driven Development (TDD) [2]. TFP consists of two test suites, critical and master, which are designed in advance by a performance architect. In order to get fast feedback, the critical test suite is created to provide early warning of performance problems. However, to realize the overall system's behavior under heavy workloads the master test suite is designed and executed after the implementation of the whole system. A potential benefit of the TFP approach is that it can minimize the overall cost of software testing for performance oriented applications like web services [1].

W. Jun et al. have identified performance testing as one of the ideal candidates to be executed in the cloud [3]. Since TFP is an approach for performance testing, organizations may get motivated to use the resources provided by the cloud for executing the test cases. In addition, it should be kept in mind that critical test suites of the TFP approach will be executed during the development phase. Thus the existing cloud testing service should provide certain components that will work alongside the developers IDE and able to give instant feedbacks after the test case execution.

Although there is a good volume of literature addressing cloud testing frameworks, to the best of the authors' knowledge those frameworks do not specifically address TFP. Existing frameworks focus on generic architecture [4], [5] and issues such as efficiency [6], automation [7], [8], communication protocol [9], etc. For example, D-Cloud [7] is a testing framework that automates the testing procedure using descriptions of the system configuration and test scenario. Cloud9 [6] is another example which achieves high level of efficiency by using symbolic execution. AGARIC [9] on the otherhand, provides the Test Flow Control Protocol (TFCP) for communicating within different nodes of the service. However, it is arguable that the existing frameworks include all the components necessary to support TFP.

This paper proposes the design of a cloud testing framework IVRIDIO which extends D-Cloud [7] and AGARIC [9]. To address the inadequacy of supporting TFPaaS, two sets of components have been designed. The first set of components is at the client's end which includes the Test Script Validator, TFP Service Communicator and TFP Application Identifier which bundled together, are named as the Plugin for TFP in the Cloud (PTFPC). The design paradigm Convention over Configuration (CoC) [10] has been adopted with PTFPC to simplify the service availing procedure by providing a project template to maintain TFP test cases.

The second set of components is at the TFP Service's end which includes the Test Receiver, Test Script Parser and Test Task Manager. Although these components persist in the existing frameworks, their functionality have been modified to support the TFP scheme. The service will receive test cases from the PTFPC and provide results by means of SOAP [11] messages. This coordination within the PTFPC and TFP Service will ensure that the clients' are receiving instant feedbacks to address performance issues.

The framework can be prototyped by partitioning the system into three subparts. The PTFPC can be implemented by using Plugin-Development Environment (PDE) [12], given that the IDE will be Eclipse [13]. The TFP Service should run in a public cloud platform and thus platforms like Google App Engine [14] can be used to deploy the service. Test cases should be executed in a separate cloud environment, preferably a private cloud, and the environment can be developed using an open source software like Eucalyptus [15].

The rest of the paper is organized as follows: Different frameworks for testing in the cloud are discussed in Section II. In Section III, the proposed cloud testing framework IVRIDIO is described along with how it offers TFPaaS. Section IV

presents the guideline to prototype the system for testbed implementation. Lastly, this paper is concluded with a discussion about proposed framework and future research directions.

II. Related Work

Lian Yu et al. developed a prototype of Software Testing as a Service (STaaS) on cloud and evaluated the scalability of the platform by increasing the test task load [4]. They have analyzed the distribution of computing time on test task scheduling and test task processing over the cloud and compared the performance of their proposed algorithm to existing schemes. Their work have addressed four major issues which include- clustering tenant requests, scheduling clustered tasks, monitoring testing resources and task status, and managing cloud processes, processors and virtual machines using a dynamic approach.

Cloud-based Performance Testing System (CPTS) for web services has been introduced in [5]. The CPTS architecture consists of three layers- user management layer, system control layer, and cloud computing service layer. The user management layer provides the front end to handle user interactions. The system control layer is used for managing and distributing performance test tasks, providing the test distribution and test run-control function, managing test scripts and test data, processing test results and controlling the whole platform. Lastly, the cloud computing service layer consists of the cluster server and the node server on which every process will execute.

A software testing environment named D-Cloud is described in [7], [8]. D-Cloud uses the cloud computing technology and focuses on an important issue that highly dependable systems must be fault tolerant for software and hardware failures. The testing environment uses the virtualization aspect of the cloud to test hardware fault tolerance. D-Cloud automates the testing procedure by using descriptions of the system configuration and the test scenario to execute tests. This methodology can also be used in other testing frameworks to automate the execution of test cases and thus increasing the efficiency of the system.

AGARIC [9] is a hybrid cloud based test platform which uses both centered cloud resources and diverse distributed user owned resources to organize the test network. There are three types of nodes in AGARIC which are- test control node, test center node and test daemon node. In order to organize diversely distributed test daemon nodes into a resource pool and to communicate among three nodes, involved researchers have also proposed a communication protocol- Test Flow Control Protocol (TFCP). The protocol uses stateless HTTP as a base and a basic set of actions is used for performing the communication.

Cloud9 is a testing service that allows to upload and test applications swiftly as a part of the development cycle [6]. It achieves a significant level of automation using symbolic execution. This technique can explore all feasible execution paths in a program, and thus is an ideal candidate for test automation. For improving the scalability of Cloud9's parallel symbolic execution engine, multiple challenges were addressed including path explosion, constraint solving overhead, memory usage, the need to do blindfolded work partitioning, distributing the search strategy without coordination, and avoiding work and memory redundancy.

The review of the state of the art frameworks shows that though multiple issues have been considered in designing a framework for cloud testing, none of the framework offer a service based on instant feedback which is essential for paradigm like TFP. In order to have seamless execution of a TFP test case, clients' end must not be overlooked and new tools should be provided to developers for effective integration of TFP along with the development phase. IVRIDIO is one such framework that offers TFPaaS and ensures the proper execution of TFP by delivering a set of new components that will work alongside clients' development environment.

III. IVRIDIO: Introduction to TFPaaS

IVRIDIO is a cloud testing framework that focuses on offering TFPaaS by means of cloud computing technology. The architectural components of IVRIDIO are bound to provide the service using defined interactions within themselves. In addition, to simplify the service availing process, CoC is being applied during the design. This section aims to clarify those details of IVRIDIO with an assumption that a web service, which demands high performance, is being tested. Besides, an overview of the TFPaaS is given to understand the sequence of actions needed to avail the service.

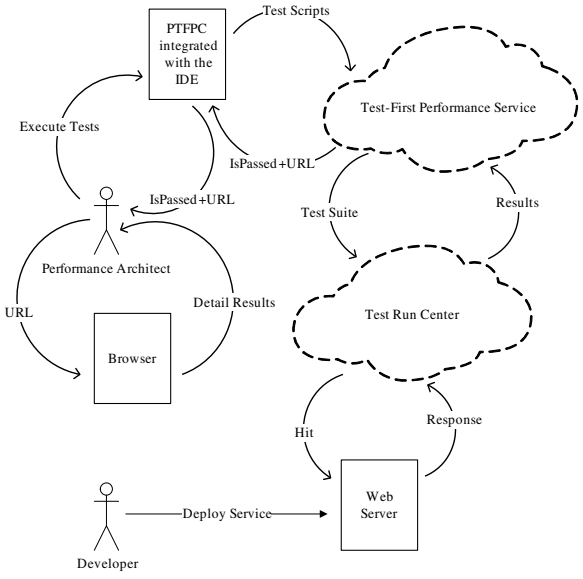

Fig. 1. Top level View of IVRIDIO

A. System Architecture

The top-level overview of the proposed architecture consisting of three core components is shown in Figure 1. The **first** component is the PTFPC which is designed to be assimilated

with the clients' IDE. PTFPC is designed by centering around that fact that it will aim to provide a seamless execution of the critical test suite. Such execution is provided, alongside the regular software development, by providing instant feedbacks required to fix different performance problems. The instant feedback will also ensure the location transparency which is essential in distributed systems like the cloud.

The **second** component is the Test-first Performance Service (TFPS) which converts test scripts to a runnable set of test suites. These test scripts are provided to the TFPS by the PTFPC when it receives the command for executing test cases from the performance architect. Performance architect does so, as soon as a developer deploy a specific service to the web server. The **third** component is the Test Run Center that executes runnable test suites provided by the TFPS. Results which are obtained after executing test suites are returned to the service. The service will store those results in a repository and a summarized version will be provided to performance architect via PTFPC. A URL will also be given for viewing the detailed results by using a simple web browser.

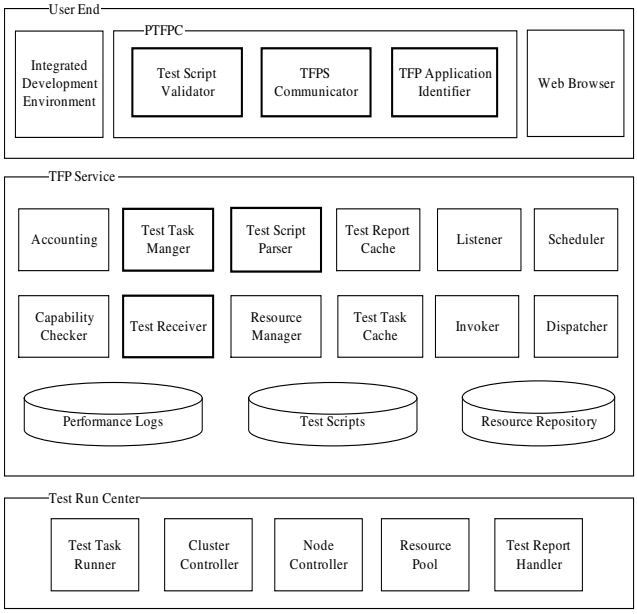

Fig. 2. The component stack of IVRIDIO

The component stack of the proposed framework is represented in Figure 2. Components having thick borders are core elements which aid in offering TFPaaS and will be focused in this paper. These components are either newly introduced or modified from existing frameworks like D-Cloud and AGARIC. Other components are supporting elements and similar to elements of [4], [5].

The **top layer** of the stack is composed of User End components. As stated earlier, this layer consists the newly introduced PTFPC. The plugin is subdivided into three parts. The **first** component is the test script validator that will run on the background and continuously validate test scripts which are being written. In existing frameworks this component resides in the cloud. However, as validation process does not demand huge resources offered by the cloud, it can be moved to the user end. Besides, as the validator will continuously run on the background, testers will be aware of their mistakes instantly when they are writing test scripts. Moreover, a tester will not like to get a feedback from the cloud that there is an error in the test script, rather it is better to identify the error right from her/his own machine.

The TFP Service (TFPS) Communicator is the **second** component of the plugin and will handle the communication with the TFP Service. To be more precise, the communicator will forward test scripts to the service and receive results after the test case execution. Test scripts can be written following the pattern of scripts in state of the art framework like D-Cloud. As shown in Figure 3, the validator will provide a validated test script to the communicator in time of creating the SOAP message.

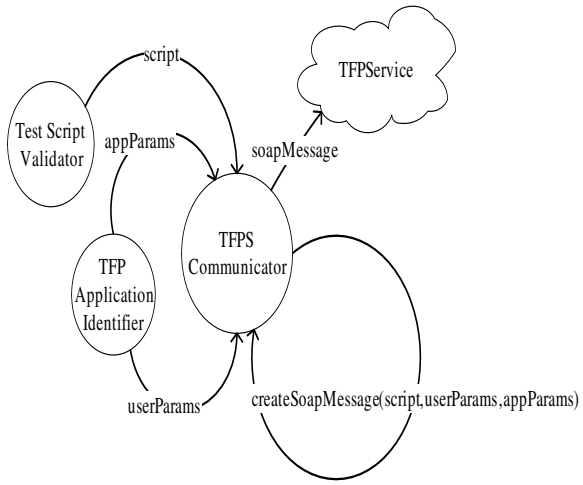

Fig. 3. The interaction within the three components of the PTFPC

The **third** component- TFP Application Identifier will provide an application identity which will be stored in the TFP Service's database to track the specific application. Besides, it will also provide user credentials and for that, the user have to be logged in into the IDE. After receiving the required data from the validator and identifier, the communicator will create a SOAP message and send it to the TFPS.

The generic structure of a SOAP message for communicating with TFPS is shown in Figure 4 having three subparts. The first part is for the identification of the application in the cloud's end. The second part consists of the test case designed in format similar to D-Cloud. The last part consists of the performance criteria based on which it will be decided that whether the service meets certain performance standards. A description of the message's child nodes which are related to the test case is provided in Table I.

The **middle layer** of the proposed framework is the TFP Service layer. Three major components, focusing on an efficient execution of test cases are the Test Receiver, Test Task Manager and Test Script Parser. Figure 5 illustrates

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/
2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:TFPService xmlns:m="URI">
      <m:application>
         <m:appId></m:appId>
         <m:userName></m:userName>
      </m:application>
      <m:case>
        <m:url></m:url>
        <m:method></m:method>
        <m:message></m:message>
      </m:case>
      <m:criteria>
        <m:response></m:response>
        <m:tps></m:tps>
        <m:bps></m:bps>
      </m:criteria>
    </m:TFPService>
  </soap:Body>
</soap:Envelope>
```

Fig. 4. The generic structure of a SOAP message for communicating with the TFP Service

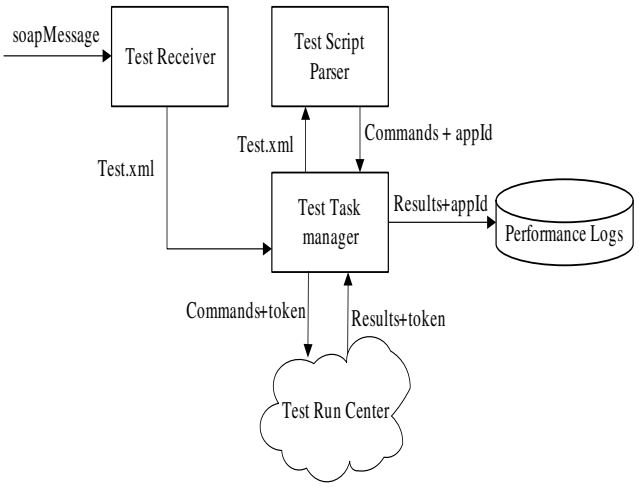

Fig. 5. The interaction among the three core components of TFP Service

TABLE I
DESCRIPTION OF THE TEST CASE ATTRIBUTES WITHIN THE SOAP MESSAGE

Attribute	Description
appId	The application identity which will be provided by the PTFPC Application Identifier
userName	The username of the current logged in user into the IDE
url	The URL of the web service which will be under test, for example www.example.com/TFP/
method	HTTP GET or HTTP POST
message	The message that should be posted on the provided URL
response	The expected response time, for example 3 millisecond
tps	The expected throughput i.e transactions per second, for example 30
bps	The expected bits per second, for example 1048576

interactions among those components which will took place after receiving a test case by the Test Receiver. The receiver will provide the SOAP message to the test manager in a XML format. The test manager will forward this file to the test script parser which will parse the file to provide a set of instructions and application identity to the manager.

After receiving instructions, the manager will send those to the **final layer**- Test Run Center via protocol similar to TFCP [9]. The result will also be received by the manager after being sent by the test run center upon executing the test suite. The result will be stored in the database and the notification will then be provided to the PTFPC.

B. Convention over Configuration

The principle of CoC design paradigm is to reduce the humanly decision to gain simplicity without loosing the flexibility. The principle is incorporated during the design of IVRIDIO to reduce complexities of availing TFPaaS. This was accomplished by providing a project template along with PTFPC. Given that the template is followed, the plugin will be able to reduce number of steps required to execute the test case for a specific service. As IVRIDIO is a cloud-based testing framework it will certainly help to reduce the overall development cost [16] however, the benefit prvided by CoC in the context of a large project will further reduce development cost due to the fact that it is removing certain human involvements from the testing procedure.

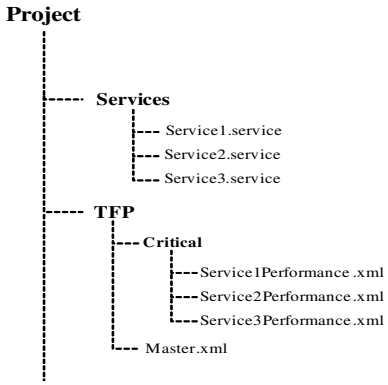

Fig. 6. The directory structure according to the project template provided by the PTFPC

The project template provides the directory structure shown in Figure 6. It can be observed that the name of a test case is given in the format *<ServiceName>Performance*. All critical test scripts are kept under the directory *TFP\Critical* and the master test script is kept in the *TFP* folder. This directory pattern eliminates the task of binding each service with its test case and thus the PTFPC will be able to identify and execute test cases for each service without any configuration file being written. However, the flexibility will be kept such that the developer will also be able to change this directory structure, according to their own convenience, by maintaining a configuration file.

The directory pattern will yield multiple other benefits if

certain features could be ensured during the implementation. The pattern should allow users to create the test case file corresponding to the service by simply right-clicking on the service file and then giving the command to create critical test case file. Similarly, test cases can be executed by right-clicking in the service file and giving the command to run critical test case. However, these options will not be applicable for the master test suite as it does not require frequent execution and is not related to one specific service. Although, feature can be added such that the master test case file will be created whenever a TFP project is being opened. Moreover, options can be added to execute the master test suite by right-clicking on the project root-folder and then giving the command to run master test suite.

C. Test-First Performance as a Service

IVRIDIO provides TFPaaS using the algorithm provided for TFP by Johnson et al. in [1]. Initially performance test cases should be designed and after the design completion the developer has to implement a functionality. The implementation will continue until it passes its functional test. Upon passing the functional test, the service will be deployed in a web server. These preliminaries are illustrated in Figure 7.

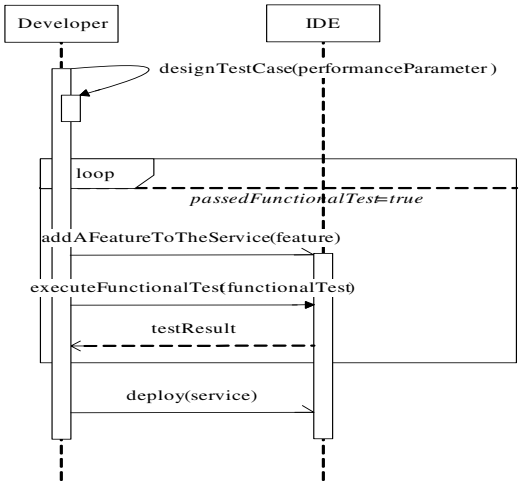

Fig. 7. A sequence of actions showing how a functionality will be implemented using the TFP principle

Certain steps that should be performed by the PTFPC to send a test case to the TFP Service is shown in Figure 8. The performance architect will give the command to execute test case for the corresponding service. PTFPC will sort out the respective test script and forward it to the TFP Service. If the test case for the service is not written then an error message will be generated.

Finally the test case will be passed to the service as shown in Figure 9. After the service receives the test case, it will parse the test case into a set of instructions. These instructions will be forwarded to the test run center for execution. Upon executing those instructions, a performance log will be generated which will later be used for judging the performance of the web service.

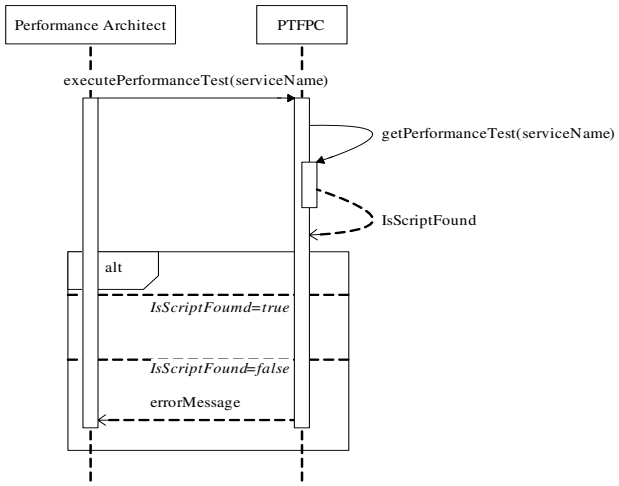

Fig. 8. Sequential steps followed by the PTFPC to send a test case to the TFP Service by getting the service name

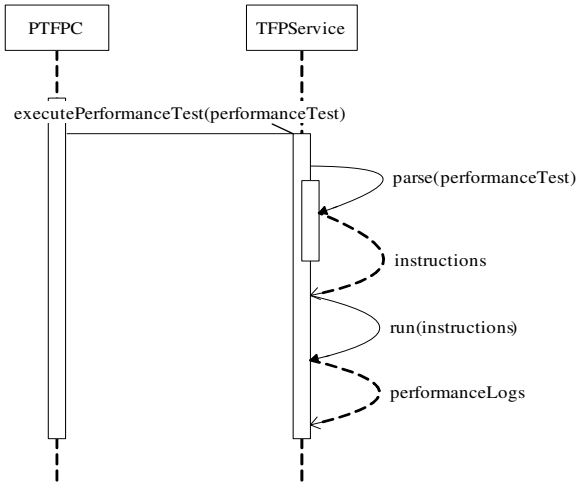

Fig. 9. Sequential interaction between the PTFPC and TFP Service for executing a test case and storing the result in a repository

TFPaaS offered by IVRIDIO ensures that the location transparency is established. This can be understood by observing the above mentioned procedure to avail TFPaaS. The developers will pass the command to execute the test case from their IDE and the rest of the task will be handled by PTFPC. Moreover, instant feedbacks will be provided after the test case execution, ensuring that the developers will feel like working within the same environment. Achieving this is an essential requirement for any distributed system.

IV. Prototyping

For research purpose, the proposed framework may get implemented as a testbed. It will be easier to implement if the framework is considered as three separate components interacting within themselves. These three components are PTFPC, TFPS and Test Run Center. The implementation of PTFPC will differ based on the IDE on which it will reside. Although the implementation of the TFPS and Test Run

Center may also differ based on the used technology, they should be deployed in a public and private cloud environment respectively. A possible scheme to implement these three components is illustrated below.

PTFPC can be implemented for an IDE like Eclipse [13]. The PDE [12] can be used to develop the plugin and integrating the project template with it. An issue may arise with application identifier regarding the generation of unique application identity. This can be resolved using identifier standard like Universal Unique Identifier (UUID) [11] to specifically track an individual application. The test scripts will be written in XML and the Java API for XML Processing (JAXP) can be used to handle those scripts.

TFPS should be deployed in a public cloud platform like Google App Engine (GAE) [14]. GAE will provide a platform for the developer to build the service using Java Web Services API. During the processing period, some of the files may require to be stored in the cache for faster access and GAE will allow to implement that feature by providing the Memcache Java API. Besides for storage purpose, the developers will have options to chose from the Datastore or Blobstore offered by the App Engine. Moreover, GAE will provide the Secure Data Connector (SDC) [17] that allows data communication from behind a secure firewall to Google Apps. This benefit should facilitate the implementation of the test run center in a private cloud environment.

Test run center can be implemented using an open source software like Eucalyptus [15]. The component that will run the test tasks, using a separate performance testing software, should be developed using any preferred programming language. Apache JMeter [18] should be a good choice for the performance testing software as it can be easily installed in a distributed system like the cloud. For providing a graphical representation of the results, gnuplot can be used. The test task runner that can be written using Shell Scripts, will interact with JMeter and gnuplot to execute test cases and provide results respectively.

V. Conclusion

This paper introduces a testing framework named IVRIDIO which offers TFPaaS. IVRIDIO emphasizes on the client's end by adding the PTFPC that allows the developers to execute critical test suites along with the regular development phase. Moreover, the PTFPC focuses on providing a fast feedback which is one of the prime requirement of TFP. The framework also incorporates the paradigm Convention over Configuration to illustrate its effectiveness in reducing overall test effort. The directions to prototype the framework are provided so that it can be used as a testbed in related research.

The solution has the additional advantage of lower test costs since it is based on cloud platform. A future challenge evolving from this research is to design a plugin which will support different cloud based testing services. The plugin proposed in IVRIDIO is specifically designed to support TFP. However, issues will arise if someone wants to avail the service offering another testing methodology. Thus the further research challenge is to design a generic architecture for a plugin of this kind.

Acknowledgement

This work is supported in part by Institute of Information Technology, University of Dhaka and Panacea Systems LTD, Bangladesh.

References

[1] M. Johnson, C. Ho, E. Maximilien, and L. Williams, "Incorporating performance testing in test-driven development," *IEEE Software*, vol. 24, no. 3, pp. 67–73, 2007.

[2] B. George and L. Williams, "A structured experiment of test-driven development," *Information and Software Technology*, vol. 46, no. 5, pp. 337–342, 2004.

[3] W. Jun and F. Meng, "Software Testing Based on Cloud Computing," in *Proc. of International Conference on Internet Computing & Information Services*. IEEE, 2011, pp. 176–178.

[4] L. Yu, W. Tsai, X. Chen, L. Liu, Y. Zhao, L. Tang, and W. Zhao, "Testing as a Service over Cloud," in *Proc. of International Symposium on Service Oriented System Engineering*. IEEE, 2010, pp. 181–188.

[5] L. Zhang, Y. Chen, F. Tang, and X. Ao, "Design and implementation of cloud-based performance testing system for web services," in *Proc. of International ICST Conference on Communications and Networking in China*. IEEE, 2011, pp. 875–880.

[6] L. Ciortea, C. Zamfir, S. Bucur, V. Chipounov, and G. Candea, "Cloud9: A software testing service," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 4, pp. 5–10, 2010.

[7] T. Banzai, H. Koizumi, R. Kanbayashi, T. Imada, T. Hanawa, and M. Sato, "D-cloud: Design of a software testing environment for reliable distributed systems using cloud computing technology," in *Proc. of International Conference on Cluster, Cloud and Grid Computing*. IEEE Computer Society, 2010, pp. 631–636.

[8] T. Hanawa, T. Banzai, H. Koizumi, R. Kanbayashi, T. Imada, and M. Sato, "Large-scale software testing environment using cloud computing technology for dependable parallel and distributed systems," in *Proc. of International Conference on Software Testing, Verification, and Validation Workshops)*. IEEE, 2010, pp. 428–433.

[9] J. Wu, C. Wang, Y. Liu, and L. Zhang, "Agaric– A hybrid cloud based testing platform," in *Proc. of International Conference on Cloud and Service Computing*. IEEE, 2011, pp. 87–94.

[10] J. Miller, "Convention over Configuration," *MSDN Magazine*, 2009.

[11] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI," *IEEE Internet Computing*, vol. 6, no. 2, pp. 86–93, 2002.

[12] D. Drusinsky, "TLtoSQL: Rapid post-mortem verification using temporal logic to SQL code generation in the Eclipse PDE," in *Proc. of International Conference on System of Systems Engineering*. IEEE, 2009, pp. 1–5.

[13] G. C. Murphy, M. Kersten, and L. Findlater, "How are Java software developers using the Elipse IDE?" *IEEE Software*, vol. 23, no. 4, pp. 76–83, 2006.

[14] A. Bedra, "Getting started with google app engine and clojure," *IEEE Internet Computing*, vol. 14, no. 4, pp. 85–88, 2010.

[15] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Proc. of International Symposium on Cluster Computing and the Grid*. IEEE, 2009, pp. 124–131.

[16] A. Imran, A. U. Gias, and K. Sakib, "An Empirical Investigation of Cost-Resource Optimization for running Real-Life Applications in Open Source Cloud," in *Proc. of International Conference on High Performance Computing and Simulation*. IEEE, 2012, pp. 718–723.

[17] J. Feng, Y. Chen, W.-S. Ku, and P. Liu, "Analysis of integrity vulnerabilities and a non-repudiation protocol for cloud data storage platforms," in *Proc. of International Conference on Parallel Processing Workshops*. IEEE, 2010, pp. 251–258.

[18] Q. Wu and Y. Wang, "Performance testing and optimization of J2EE-based web applications," in *Proc. of International Workshop on Education Technology and Computer Science*, vol. 2. IEEE, 2010, pp. 681–683.