ASTES

Advances in Science, Technology and Engineering Systems Journal Vol. 3, No. 2, 252-256 (2018)

www.astesj.com

Special Issue on Advancement in Engineering Technology

ASTESJ

ISSN: 2415-6698

Frameworks for Performing on Cloud Automated Software Testing Using Swarm Intelligence Algorithm: Brief Survey

Mohammad Hossain*, Sameer Abufardeh, Sumeet Kumar

Department of Math Science and Technology, University of Minnesota Crookston, Crookston, MN 56716, USA.

ARTICLE INFO

Article history:
Received: 01 March, 2018
Accepted: 16 March, 2018
Online: 11 April, 2018

Keywords:
Swarm Intelligence
Ant Colony Algorithm
Bee Colony Optimization
Cloud Based Testing
Test Case Prioritization

ABSTRACT

This paper surveys on Cloud Based Automated Testing Software that is able to perform Black-box testing, White-box testing, as well as Unit and Integration Testing as a whole. In this paper, we discuss few of the available automated software testing frameworks on the cloud. These frameworks are found to be more efficient and cost effective because they execute test suites over a distributed cloud infrastructure. One of the framework effectiveness was attributed to having a module that accepts manual test cases from users and it prioritize them accordingly. Software testing, in general, accounts for as much as 50% of the total efforts of the software development project. To lessen the efforts, one the frameworks discussed in this paper used swarm intelligence algorithms. It uses the Ant Colony Algorithm for complete path coverage to minimize time and the Bee Colony Optimization (BCO) for regression testing to ensure backward compatibility.

1. Introduction

With the increase of code complexity in modern software, chances of errors are exponentially increasing. These errors can cause loss of money and innocent human lives [1]. For example, in April 24 1994, a China airline airbus A-300 crashed, due to a software bug, resulting the death of 264 innocent lives [2, 3]. Another famous incident was reported in April 1999, where a small software bug in a military satellite was behind a $1.2 billion loss: one of the costliest unmanned accidents in the history of Cape Canaveral launches [4, 5].

Therefore, to reduce the risks of errors, researchers have developed a variety of testing techniques to find and fix software bug early and before the deployment of the software. One of the most critical tests is unit testing, where each module of program is tested separately. Another critical test is integration testing that occurs after unit testing, where individual software modules are combined and tested as a group. Since unit testing requires access to the system code, it is done during the initial stages of a program, detecting an estimated 65% of the errors [6, 7, 8]. Other types of tests includes, system testing and acceptance testing. In system testing, the system is tested as a whole to verify that it meets the specified requirements. After that, acceptance testing is done to verify that, the system meets the client/user requirements.

*Corresponding Author: Mohammad Hossain, Crookston, MN 56716, 218-281-8222, Email: hossain@crk.umn.edu

Testing is generally a lengthy and costly process. Therefore, automated software testing generally intended to reduce the time and the cost of testing. It also can increase the depth and scope of tests to help improve software quality. However, most automated testing tools fails to provide efficient results, because they only focus on specific testing techniques [9, 10] and in sometimes they may be unsuitable for large-scale software.

This survey paper focuses on testing based on Cloud platforms tools in order to develop cost effective, efficient and time saving tools that follows the rules of research based techniques to produce software free of or with few errors or bugs.

2. Testing

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to identify and understand the risks of software implementation. Testing techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs (errors or other defects). It is the process of validating and verifying a software program, application or product [11]. Software testing is a huge domain, but it can be broadly categorized into two areas: manual testing and automated testing:

2.1. Manual Testing

An individual or a group of individuals performing all of the software quality assurance testing, checking for errors and defects, is knows as manual testing.

2.2. Automated Testing

The main purpose of this testing is to replace manual testing with automated cloud based testing without loss of efficiency, in such a way that it can not only save time but also produce high quality software.

3. Terminologies and Abbreviations

The following abbreviations and terminologies were used in this research paper.

3.1. Regression Testing

Software undergoes constant changes. Such changes are necessitated because of defects to be fixed, enhancements to be made to existing functionality, or new functionality to be added. Anytime such changes are made, it is important to ensure that, first changes or additions work as designed. Second changes or additions are something that is already working and should continue to work. Regression testing is carried out to ensure that any new feature introduced to the existing product does not adversely affect the current [11].

3.2. Traceability

Traceability is defined as the ability to describe and follow the life of a requirement, in both forward and backward direction, throughout the software life cycle. Traceability relations can assist with several activities of the software development process such as evolution of software systems, compliance verification of code, reuse of parts of the system, requirement validation, understanding of the rationale for certain design decisions, identification of common aspects of the system, and system change and impact [12,13].

3.3. White-Box Testing (WBT)

White-Box Testing, also known as clear box testing, gives verification engineers full access to the source code and the internal structure of the software. It is the detailed investigation of internal logic and structure of the code [14]. In WBT, it is necessary for a tester to have full knowledge of source code. Some important types of WBT techniques includes Statement Coverage (SC), where tester tests every single line of code, and Condition Coverage (CC) in which all the conditions of the code are checked by providing true and false values to the conditional statements in the code.

3.4. Black-Box Testing (BBT)

BBT treats the software as a “Black Box” without any knowledge of internal working of the system and it only examines the fundamental aspects of the system. In BBT the tester has knowledge of the system architecture but he/she does not have access to the source code [15].

4. Swarm Intelligence Algorithms to Optimize Regression Testing.

Regression Testing ensures that any changes or enhancement made to the system will not adversely affect the functionality of software. The execution of all test cases can be an costly and time consuming process. With this in hand, prioritization of test cases can help in reduction in cost of regression testing. Swarm intelligence is an emerging area in the field of optimization and researchers have developed various algorithms by modeling the behaviors of different swarm of animals and insects such as ants, termites, bees, birds, fishes [16]. These algorithms are being used to reduce the time and cost of testing in general and more specifically regression testing [11, 17]. Two such widely used algorithms are Ant Colony Algorithm and Bee Colony Optimization.

4.1. Ant Colony Algorithm (ACA)

Ant colony algorithms are based on the behavior of a colony of ants when looking for food. In their search they mark the trails they are using by laying a substance called pheromone. The amount of pheromone in a path tells other ants if it is a promising path or not. This observation inspired Colorni, Dorigo and Maniezo [18] for proposing a metaheuristics technique: ants are procedures that build solutions to an optimization problem. Based on how the solution space is being explored, some values are recorded in a similar way as pheromone acts, and objective values of solutions are associated with food sources. An important aspect of this algorithm is parallelism: several solutions are built at the same time and they interchange information during the procedure and use information of previous iterations [19]. In [20] Li and Lam proposed to use UML State chart diagrams and ACA for test data generation. The advantages of their proposed approach are that this approach directly uses the standard UML artifacts created in software design processes and it also automatically generated feasible test sequence, non-redundant, and it achieves the stated coverage criteria.

4.2. Bee Colony Optimization (BCO)

Bee swarm behavior in nature is characterized by autonomy and distributed functioning, and it is self-organizing. Recently, researchers started studying the behavior of social insects in an attempt to use the Swarm Intelligence concept in order to develop various Artificial Systems [21]. In software engineering, BCO can be used as a method of regression testing and traceability. Its purpose is to verify that the current version of the software is compatible with pervious test results and the data is comparable to previous versions of the software.

Karnavel and Santhoshkumar proposed a fault coverage regression system exploiting the BCO algorithm discussed in [11]. The idea is based on the natural bee colony with two types of worker bees that are responsible for the development and maintenance of the colony: scout bees and forager bee. The BCO algorithm developed for the fault coverage regression test suite is based on the behavior of these two bees. The algorithm has been formulated for fault coverage to attain maximum fault coverage

in minimal units of execution time of each test case. Two examples were used whose results are comparable to the optimal solution [22, 23]. The system was divided into the following components: *Testing Phase, Traceability Phase, Exploration Test, Generating Reports, and Storing in Database* (figure 1):

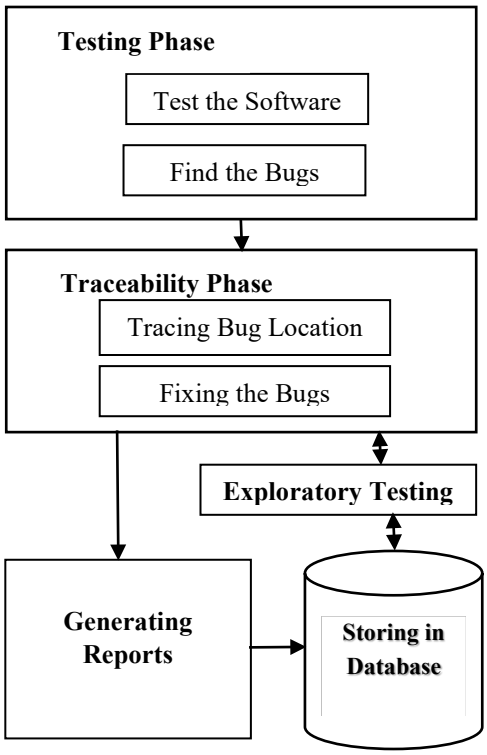

Figure 1: Architecture of BCO based fault coverage regression test.

5. Cloud Testing Framework Example

Large software requires a huge number of test cases. Often, these test cases require a lot of time and effort even with automated testing [24, 25, 26]. Because each test consumes execution time, the execution time required generally decreases when parallelization is used. Cloud Testing Frameworks presented in [24, 26] improves the execution process time by performing the parallel execution of test cases using current computational resources without source code modification [24, 26, 27].

Using a distributed and parallelized test can result in significant reduction of time required for test cases execution. It can also reduce the needed to identify and correct faults. Hence, reducing the total cost of development. Furthermore, the proposed framework in [24] increases the reliability of the test results by using heterogeneous environment that can result in the exposure of hidden failures ahead of the production phase [24].

One of the earliest Cloud Testing Framework "*CouldTesting*" presented by Oliveira and Duarte in 2013 is shown in figure 2. The framework distributes the unit tests using reflection on the local classes and then schedules the machine on cloud. It then the load is distributed over machines, using the round robin scheduling algorithm to ensure even distribution of the requests to the available machines in the test infrastructure [24].

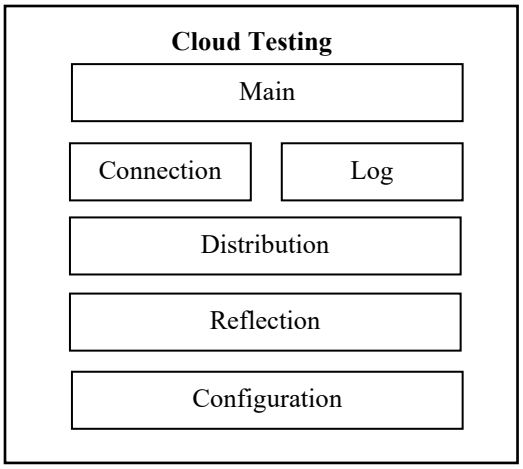

Figure 2: Cloud testing components from [24]

Figure 2 presents the architectural components of the framework. The main components of the framework are: Configuration, Reflection, Distribution, Connection, Log and Main.

The *configuration* component help in defining information for *paths, hosts,* and for *load balancing* [24]. This component deals with issues related to local storage space allocation for test results, selecting the libraries needed for proper test execution, and file access permission. It includes the list of machines and the parameters for the load balancer. The Reflection component extracts the tests cases [24].

Figure 3 depicts the distribution component being used to intermediate the execution of test suites over a parallel infrastructure. To work with a given IDE and parallel infrastructure the framework must be extended to include specific plugins. The *connection* component provides an interface on the client side to communicate with the cloud provider. In the cloud site this component provides a service that manages the execution of each test and it sends the test results back to the client in real-time. The log component records events generated in the process. The main component is a facade that encapsulates the components [24].

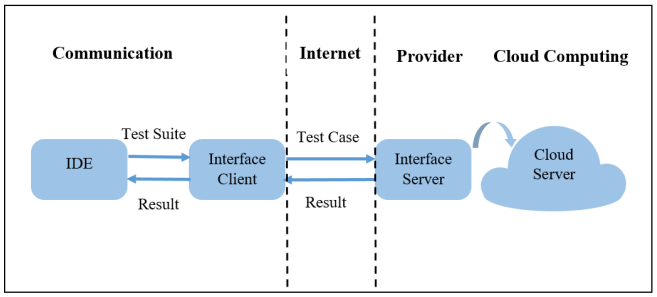

Figure 3: Distribution component being used to intermediate the execution of test suites over a parallel infrastructure from [24].

6. Test Case Prioritization Techniques

With the limited testing budget, it has been always a big challenge of software testing to optimize the order of test case execution in a test suite, so that they detect maximum number of errors. Three solutions to this problem were discussed in [28] such as test suite reduction, test case selection, and test case prioritization.

As the name suggests, the test suite reduction removes test cases from a test suite, which are redundant, and test case selection selects the most fault revealing tests based on a given heuristic. On the other hand, Test Case Prioritization (TCP) considers all the test cases without removing any of them. Instead it ranks all existing test cases thus prioritizes the test cases. While testing is performed testers executes the test cases with the higher priorities first as long as the testing budged supports [25]. Following techniques are discussed in [25, 28] to implement TCP.

6.1. Code/Topic Coverage Based

Most popular technique that has been reported in the literature is the Code coverage-based TCP that prioritize the test cases effectively [29]. It requires knowing the source code information of the software to measure the code coverage. This technique is not applicable in black- box systems tests because of lack of information about the code coverage. In [30] Thomas et. al proposed a modified a TCP technique, which they called topic coverage, provides an alternative concept to code coverage. The goal was to rank tests so that they cover more topics sooner [25, 28].

6.2. Text Diversity Based

Another common technique for TCP is to diversify the test cases. In [25] authors described a test case diversifying techniques that analyzes the test scripts directly hence this approach is called a text diversity-based TCP. The technique treats test cases as single, continuous strings of words. Then it applies different string distance metrics, such as the Hamming distance, on pairs of test cases and determines their dissimilarity. The idea is that the more two test cases are dissimilar textually the more they are likely to detect faults in different part of the source code [25, 28].

6.3. Risk Based Clustering

This TCP needs to access to execution results of the previous test cases, typically examining only the last execution of the test cases. It can be extended to as many previous executions as possible. This technique must ensure to run those test cases that failed in their previous execution provided that they are still relevant [31]. The technique might be combined with other TCP techniques, as well. For example, one can prioritize the previously failed test cases using a coverage-based approach to provide a full ordering of the test cases. In [28] authors modified this approach to have several clusters of test cases of different riskiness factor rather than having only two clusters of failed and non-failed test cases. In their approach, the highest risk is assigned to the tests that failed in the immediate version before the current version. The next riskiest cluster are tests that did not fail in the previous version but failed in the two versions before the current version, and so on.

7. Proposed Frameworks Discussed

As we discussed earlier, Swarm Intelligence algorithms as an emerging technique in the field of optimization are being integrated in many of the automation testing frameworks. These algorithms were instrumental in improving the performance and the efficiency of software testing on the cloud. In this section, we briefly highlight few of the frameworks that used Swarm Intelligence algorithms:

In [23] A. Kaur and S. Goyal proposed a Bee Colony Optimization algorithm for fault coverage-based regression test suite prioritization. The framework imitates the behavior of two types of worker bees found in nature. The behavior of the bees has been observed and mapped to prioritize software test suite.

A similar system has been presented by Karnavel and Santhoshkumar in [11] that used Bee Colony Optimization algorithm for test suite prioritization. The authors modified an existing framework to reduce the number of test cases from the retest test pool.

G. Oliveira and A. Duarte presented one of the aerialist frameworks called 'CloudTesting'. The framework executed test cases in parallel over a distributed cloud infrastructure [24]. In the framework, cloud infrastructure is used as the runtime environment for automated software testing. Their experimental results indicate remarkable performance gain without significantly increasing the cost involved in facilitating the cloud infrastructure. The framework also simplified the execution of automatic tests in distributed system.

The framework presented by S. Faeghe and S. Emadi in [32] used the Ant Colony algorithm to automate software path test generation. The authors proposed a solution based on ant colony optimization algorithm and model-based testing for faster generation of test paths with maximum coverage and minimum time and cost. According to authors' evaluation, the framework showed better performance over the existing methods in terms of cost, coverage and time.

A. Kaur and D. Bhatt in [29] proposed a regression testing based on Hybrid Particle Swarm Optimization (HPSO). The HPSO is an algorithm where a Genetic Algorithm (GA) is being introduced to the Particle Swarm Optimization (PSO) concept. The authors used the hybrid approach to prioritize tests for regression testing. The authors also mentioned that use of algorithm improved the effectiveness of their proposed framework.

8. Conclusion

Testing is one of the most complex and time-consuming activities. Automated testing on the cloud is one of the most popular solutions to reduce the time and the cost of software testing. In this paper, we discussed examples of automated frameworks proposed for testing software on the cloud [11, 23, 24, 29, 32]. Based on our review to such frameworks, it is evident that the use of the cloud as runtime environment for software testing are more efficient and effective solution when compared to traditional methods. Furthermore, on the cloud automated testing frameworks that used Swarm Intelligence Algorithms such ACA and BCO were able to produce significant reduction in the time required to execute large test sets and cover a diverse and heterogenic testing coverage. The use of such effective algorithms facilitates and enhances parallel execution and distribution of large testing loads on the cloud. In addition, cloud-testing frameworks generally simplifies the execution of automatic tests

in distributed environments, hence gains in performance, reliability and simplicity of configuration.

Conflict of Interest

The authors declare no conflict of interest.

Acknowledgment

Authors would like to thank the reviewers of this work for their help. This work was accomplished by the support of University of Minnesota Crookston.

References

[1] R. Hower, "What are some recent major computer system failures caused by software bugs," http://www.softwareqatest. com/qatfaq1.html,[February 2015], 2005.
[2] P. Ladkin, "The crash of flight ci676." [Online]. Available:http://www.rvs.uni-bielefeld.de/publications/Reports/taipei/taipei.html
[3] P. Krömer, J. Platoš, V. Snášel, "Nature-inspired meta-heuristics on modern GPUs: state of the art and brief survey of selected algorithms." International Journal of Parallel Programming 42.5 (2014): 681-709.
[4] B. Beizer, Software testing techniques. Dreamtech Press, 2003.
[5] M. Mavrovouniotis, C. Li, S. Yang, "A survey of swarm intelligence for dynamic optimization: algorithms and applications", Swarm and Evolutionary Computation 33 (2017): 1-17.
[6] N. Gupta, "Different approaches to white box testing to find bug" International Journal of Advanced Research in Computer Sciency and Technology (IJARCST 2014) 2.3 (2014): 46-9.
[7] M. Nouman, U. Pervez, O. Hasan, K. Saghar, "Software testing: A survey and tutorial on white and black-box testing of c/c++ programs," in Region 10 Symposium (TENSYMP), 2016 IEEE. IEEE, 2016, pp. 225–230.
[8] B. Balamurugan, J. Sridhar, D. Dhamodaran, P. Venkata Krishna. "Bio-inspired algorithms for cloud computing: a review." International Journal of Innovative Computing and Applications 6, no. 3-4 (2015): 181-202.
[9] R. Khalid, "Towards an automated tool for software testing and analysis" In Applied Sciences and Technology (IBCAST), 2017 14th International Bhurban Conference on, pp. 461-465. IEEE, 2017.
[10] E. Pacini, C. Mateos, C.G. Garino. "Distributed job scheduling based on Swarm Intelligence: A survey." Computers & Electrical Engineering 40.1 (2014): 252-269.
[11] K. Karnavel1, J. Santhoshkumar, "Automated Software Testing for Application Maintenance by using Bee Colony Optimization algorithms (BCO)", Information Communication and Embedded Systems (ICICES), 2013 International Conference on. IEEE, 2013.
[12] U. Salima and A. Askarunisha, "Enhancing the Efficiency of Regression Testing Through Intelligent Agents" International Conference on Computational Intelligence and Multimedia Applications 2007, pp. 103-108.
[13] S. Dick, A. Kandel. Computational intelligence in software quality assurance. Vol. 63. World Scientific, 2005.
[14] M. E. Khan, F. Khan. "A comparative study of white box, black box and grey box testing techniques." Int. J. Adv. Comput. Sci. Appl 3.6 (2012).
[15] M. Khan, "Different Approaches to Black Box Testing Technique for Finding Errors," IJSEA, Vol. 2, No. 4, pp 31-40, October 2011
[16] K. Dervis, B. Akay. "A survey: algorithms simulating bee swarm intelligence." Artificial intelligence review 31.1-4 (2009): 61.
[17] A. Kaur, D. Bhatt "Hybrid Particle Swarm Optimization for Regression Testing", International Journal on Computer Science and Engineering, Vol. 3 No. 5 (May-11), pp1815~1824, ISSN: 0975-3397
[18] A. Colorni, M. Dorigo, V. Maniezzo. "Distributed Optimization by Ant Colonies". First European Conference on Artificial Life, 134-142, 1991.
[19] S. Mazzeo, and I. Loiseau. "An ant colony algorithm for the capacitated vehicle routing." Electronic Notes in Discrete Mathematics 18 (2004): 181-186.
[20] H. Li, P.L. Chiou, "Software Test Data Generation using Ant Colony Optimization." International conference on computational intelligence. 2004.
[21] D. Teodorovic, M. Dell'Orco. "Bee colony optimization–a cooperative learning approach to complex transportation problems." Advanced OR and AI methods in transportation (2005): 51-60.
[22] O. Gotel, and A. finkelstein "An analysis of the Requirements Traceability Problem", International conference on Requirements Engineering, USA, 1994
[23] A. Kaur and S. Goyal, "A Bee Colony Optimization Algorithm for Fault Coverage Based Regression Test Suite Prioritization", International Journal of Advanced Science and Technology Vol. 29, April, 2011
[24] G. Oliveira, A. Duarte "A Framework for Automated Software Testing on the Cloud", 2013 International Conference on Parallel and Distributed Computing, Applications and Technologies
[25] Y. Ledru, A. Petrenko, S. Boroday, and N. Mandran, "Prioritizing test cases with string distances," Automated Software Engineering, vol. 19, no. 1, pp. 65–95, 2011.
[26] A. Duarte, W. Cirne, F. Brasileiro andP. Machado, "Gridunit: software testing on the grid." In Proceedings of the 28th international conference on Software engineering (pp. 779-782). ACM, May 2006
[27] T. Banzai, H. Koizumi, R. Kanbayashi, T. Imada, T. Hanawa, M. Sato. "D-cloud: Design of a software testing environment for reliable distributed systems using cloud computing technology." In Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on, pp. 631-636. IEEE, 2010.
[28] H. Hemmat, Z. Fang, M. V. Mantyla , "Prioritizing Manual Test Cases in Traditional and Rapid Release Environments", Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on. IEEE, 2015.
[29] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," Software Engineering, IEEE Transactions on, vol. 27, no. 10, pp. 929–948, 2001.
[30] S. Thomas, H. Hemmati, A. Hassan, and D. Blostein, "Static test case prioritization using topic models," Empirical Software Engineering, vol. 19, no. 1, pp. 182–212, 2014
[31] A. Onoma, W. Tsai, M. Poonawala, H. Suganuma, "Regression testing in an industrial environment," Communications of the ACM, vol. 41, no. 5, pp. 81–86, 1998.
[32] S. Faeghe, S. Emadi. "Automated generation of software testing path based on ant colony." In Technology, Communication and Knowledge (ICTCK), 2015 International Congress on, pp. 435-440. IEEE, 2015.