International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075, Volume-8 Issue-7, May, 2019

Graphical User Interface Testing Using GUITAR and Hadoop Unit

Sumit Kumar, Nitin

***Abstract:** The testing of the graphical user interface is very complicated and time-consuming job. The complexity of the task came into existence when small changes are introduced in the software, and there is the need for regression testing. The most appropriate way to test the software for the possible states of GUI rather than relying on the randomly generated test suites. This paper focuses on reducing the time consumption of the testing the GUI using the hadoop environment with the help of event flow graph, and test cases generated using the guitar testing tool. From the result it can be observed that there is an exponential growth in the number of test cases as the length of the test sequence containing the events is increased. As the experimental setup defined in our study, two virtual machine with hadoopunit. As in the testing we can execute only one test case at time so here two test cases can be executed concurrently. The maximum speedup that can be achieved is always less than 2.0 but we are successful in reducing the time to approximately half as the speedup achieved is 1.96.*

***Keywords:** Graphical User Interface, Regression Testing, hadoop, hadoopunit;*

I. INTRODUCTION

Graphical user interface testing is the critical phase of testing. GUI is the essential part of the software under test. The GUI helps the new user to interact efficiently with the business logic of the system. The User at the front end of the software may or may not be an expert of the system. He may be a simple person with little knowledge about the technical details of the system. If there is a bug in the GUI, then this will give a negative impact on the reputation of the software. The most of the GUI application is in the form of web interface situated on the remote servers or standalone desktop applications. The testing of the GUI requires the generation of the test cases, implementation of the test cases and the result generation logs to record the output of the test oracles. The various testing techniques have been used in the past to generate the test cases which helps in the testing of the software. The most common technique of generating the test cases is using the random approach. In this approach, the various events are fired randomly. The fired events may be a valid event or maybe a false event. If an event is a valid event, the result obtained is recorded in the output repository. The output here refers to the correct result, the incorrect result (logical error) or a mismatch between the actual result and the expected result. The main advantage of using the random approach is the diversity in the test oracles generated by the system. Another idea is to generate the adaptive random test case prioritisation to generate the test cases based on the coverage basis[15]. The study conducted by [24] concludes that the random test case generation seems a useful aid to jump start manual unit testing [24]. The process of the test case generation can be simplified to some extent if the event flow diagram has been used. It is a state-based finite state machine in which the numbers of states are fixed this reduces the invalid states that are the part of the test cases in as in the earlier case. Even though the finite state machines are used for testing the GUI still the main challenges that a tester has to face are listed below:

1) A vast number of valid test cases are generated which are to be executed.
2) The system under the test in case of web application/ Desktop application will be deployed on high-end servers/Desktops that are limited in number and have fixed communication cost for the clients or to the database engines.

A large number of test cases generated must be tested to ensure the triumphant satisfaction of the user. The execution of these test cases will require a significant amount of time to do the task in an efficient manner it is necessary to divide the task into parallel jobs. The second issue is the testing of application using the client network will add the extra cost of communication, and this will affect the performance of the system. So for the quick processing, it will be better deploy the application on the same machines that have the associated data related to that application.

II. BACKGROUND

We have divided the study in three basic parts. In the first part we have discussed the Hadoop framework and in second section we have discussed the GUI testing tools and in third section we have discussed the hadoop.

A. *Hadoop Framework*

Hadoop is an open source software structure that uses the simple programming construct but enables distributed storage and data processing on the cluster of machines [5, 30]. The two central pillar of Hadoop are:

1) Hadoop Distributed File System

HDFS is a Hadoop file storing system, which is used for storing and retrieving the data in a distributed environment along with the high fault tolerance and can handle a vast amount of data and very high speed. The main idea behind the HDFS lies, in fact, moving the computation cost less in comparison transferring a massive amount of data from the source to the site of computation.

Revised Manuscript Received on May 06, 2019

Sumit Kumar,Computer Science and Engineering, Uttarakhand Technical University, Dehradun, India.
Nitin, Information System, Indian Institute of Management, Shillong, India,

HDFS also has the portable nature. The underlying architecture of the HDFS is just like master/slave architecture. Here the master is the referred as a NameNode. The NameNode is used for the management purpose which monitors the access of the files by the clients. The NameNode is performing all the necessary file system operations like reading, write, creating a directory and many other operations. Another primary task of the NameNode is to handle large files as they are spliced into blocks and are stored on data nodes. The namenode also handles the mapping of blocks to the datanode. The data nodes are responsible servicing the client request for reading and write operations and block management like replica creations/deletion as per the order of the NameNode.

2) Map Reduce Programming model

Map Reduce is a parallelization of the task done on a large number of machines which may not have a homogenous configuration or may have a different platform. Map Reduce can handle the failures in the nodes and communicates in such a way that the resources can be effectively utilized.The detailed diagram describing the Hadoop architecture is described in Figure 1.

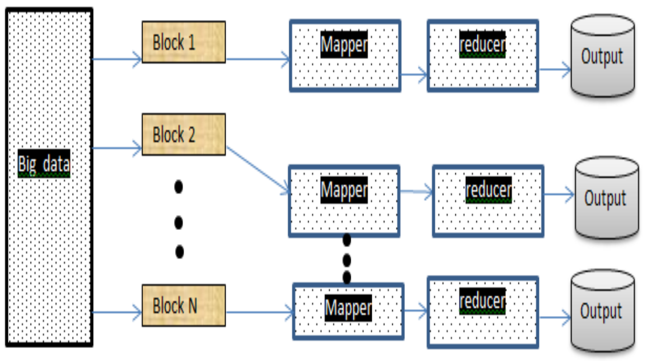

Fig 1. Describes the Hadoop Architecture

The Hadoop framework is widely used in the various applications of machine learning [7,11,14,18,23,29,32] and data mining [12,25,28].The applications are not only limited to the basic application, but not various models also can be used in various real situations like intrusion detection[21].

B. Hadoopunit

The basic use of Hadoop unit is to perform the testing in the clustered environment and is capable of running JUnit test. The performance improvement of using the cluster for the testing has already being highlighted in past [27] .The main advantage of using the Hadoopunit is that it hides the complexities of the Hadoop administration like cluster management, job scheduling and resource allocation. This make easy for the tester to focus on the basic task of testing. The hadoopunit consist of 3 main component's

1) Test case extraction: This component of hadoopunit gathers and manages all the test cases that are required for the testing of the system. It basically contains the name of the test case a basic command line script that is used to execute the particular test cases. It corresponding key/value pair so that they can be successfully mapped into the hadoop environment.
2) Map function: This function is responsible for performing the execution of the test cases in the hadoop environment. These test cases are executed in such a manner that two test cases when executing concurrently are on separate machines or execute on after another. The result obtained key/value pair and result are intermediary and are processed by the reduce function.
3) Map function: This function is responsible for performing the execution of the test cases in the hadoop environment. These test cases are executed in such a manner that two test cases when executing concurrently are on separate machines or execute on after another. The result obtained key/value pair and result are intermediary and are processed by the reduce function.

C. Review of various GUI Testing Tools

Every GUI testing tool consists of three basic components that are agent, scripts and an IDE. The details of each of the following is discussed as follows

- An Agent: It is the small frame work that is installed inside the system under test. This framework supports the testing of the system.
- A Script or A Feature: There is requirement of set of instructions that are to be passed to the system for performing a particular task. These tasks are scripted in the different programming languages like Objective-C [8, 16], JavaScript [2, 3], CoffeeScript [4], or Cucumber [31] and this mainly depends upon the technology employed in the testing software.
- An IDE: An IDE is used for executing the test cases and taking the record of all the activities and also helps in interaction with agent to perform the interaction with the components inside the system under test

In practice, GUI testing tools use two significant approaches. In the first approach, the scripting languages like Selenium WebDriver are too used to generate the customized test cases. These manually designed test cases are then invoked by the program to generate the GUI events. A unit test case consists of method calls which programmatically invoke GUI events. The test cases when executed and are verified by the tester. As the manual scripting of the test case is a tedious job, so the tools based on another approach are called capture/record can be used. Examples of capture/replay tools include Selenium IDE7 and Quick Test Pro8. The various tools under the study are described below:

1) Monkey:

This tool is a random walking tool specifically designed for Android-based applications. There is no specific protocol for generating the test cases. Monkey is based on the random approach. Here the single instance of the application is used, and one by one different event is fired. It is the total automated tool and is helpful for the coverage test of the Android applications. The major drawback of the Monkey is that the tool is incapable of generating the test cases according to the need of the tester.

As custom test cases cannot be generated this limits the uses the tool to test the android application for the specific types of error like crashes, session timeouts and credential error [6, 26].

2) NModel

The NModel tool basically works on the basis of the model specifications generated by the tester. The NModel tools were specifically designed for the testing of C# programs. The NModel tools are unable to read the layout of the GUI software directly. The main advantage of using the NModel is that the customized scripts can help in specifying the more accurate details of the software. The scripts are written for the testing are coded into model itself and can be directly used on the scripting alternative allows for more precise model defination at the expense of scalability, as manual specifying a model correctly may take a great deal of time for non-trivial applications. With NModel, assertions can be coded into the model itself and applied during test execution as a test oracle.

3) Quick Test Pro

It is the proprietary tool. The tool has the capability of capturing the test cases automatically and also has the option to provide the scripts for the remaining tools. But the major drawback of this tool is that the tester is provided with limited customization. As the tool is proprietary the various platform versions are available [17].

4) Selenium

Selenium is basically used for the testing of websites. Selenium has the plug-in for the browser it has the support of various scripting languages for the testing web application. The major advantage of using the selenium is that it also has the support for the grid testing [17].

5) Frank

Frank has been used by Pete Hodgson for testing of iOS basically used native iOS app. It has used Cucumber and JSON scripts for controlling the application under the test from IDE. The main advantage of using the cucumber is that little knowledge makes it easy understandable to non-technical persons in the project. Frank was treated as the selenium for the iOS platform. Another major tool named as Symbiote provides the facility to use the web browser for handling the complete system under test. This gives the added advantage to the tool that the testing is done on the exact system for which the GUI is designed.

6) KIF

KIF was created by Square, the company that designs the Square credit card reader for the Android and iOS devices. The tool was designed for the automated testing even though the tool is similar to tool like Frank as it allows the tester to automate iOS GUI testing. Tool uses the Objective-C built on top of OCUnit and does not uses the scripts language this provides better integration with iOS applications. The major disadvantage is the requirement of developer for the development of test cases.

7) GUITAR

It is free open source tool used for performing the testing of GUI. It is a model-based approach. In the model-based approach uses the GUI flow from one state to another. These changes from one GUI element to another GUI element are usually known as the states. The transition from Graphical element generates the model. This model is usually a finite state machine (FSM). The main advantage of the Model-based exploration approach is that it utilizes the flow diagram, so it has the excellent coverage in comparison to random approach. The main drawback of this approach is that only the changes in the GUI will become the new states in the model[3] which invites the regression testing of the complete system even if only one change is introduced in the system[22].

8) Details of GUITAR Framework

The GUITAR framework allows for the development of four primary tools. They are:

- RIPPER: It is a tool used to perform the reverse engineering on the software under test with the help run-time states of the application. The resultant output is a structural model of the GUI application under test [19]. The result generated by the software is stored in an XML file.
- Graph convertor: This tool uses the result obtained from the ripper as the input and generates the Event-flow graph(EFG) based upon the run times states of the application. The EFG of the software under test is stored as XML file but can be converted into a graph with the provided application tool which can be viewed with Graphviz tool [20].
- Case Generator: Again the Case generator used the event flow graph generated in the previous step as the input and generated the various cases of the various lengths as provided by the user. These test cases are stored separately in the XML files [10].
- Replayer: As per the choice of the particular user set of test cases are executed, and the results are stored in the separate folder of user choice.

After we have got the results of the test case, we can go through the expected results and obtained results to identify the faults in our system. Figure 2 describes the various layers from top to bottom that is to perform to get the log files that can be used to identify the fault in the desired software under test.

III. Experimental Setup

For performing the testing of software under test, we have configured a virtual machine with the configurations as described in table 1.

Table 1: Describing the Configuration of the Hadoop environment

S.no.	Parameter	Specification
1	CPU	Intel® Core™ i5-7200U @2.5 GHz
2	RAM	8 GB
3	Software	Virtual Box 5.2.2
4	VM OS	Ubuntu 14.04 LTS
5	JDK	1.8
6	Hadoop Environment	Hadoop unit 2.7
7	Number of VM'S	2
8	Number of Mapper and Reducers	Two mappers and two reducers

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

The detailed configuration of the virtual machines used for the testing of the software using the hadoopunit is as follows. The machines have the 64 bit Ubuntu operating system with the base memory of 3096 MB. The video memory of 12 MB has been assigned to each of the virtual machines.

The both machines are communicating with the help of virtual network established between the two nodes for their effective communication.

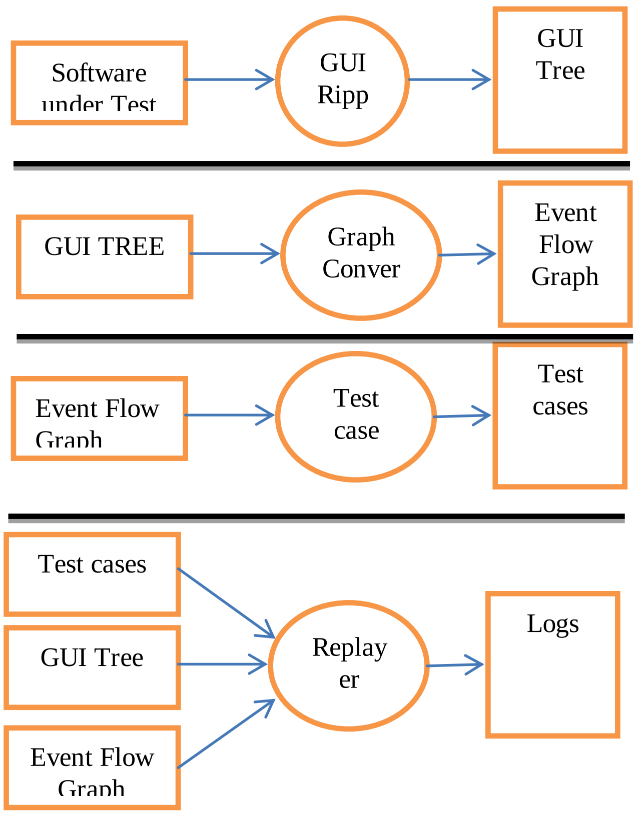

Fig 2. Describing the various levels of tools in guitar that are executed to get the logs for analysis

To test the performance of test case execution, we have taken the example of simple radio button application that contains a simple GUI. This is the example that has been referred in the Guitar manuals and in [22]. The GUI of the software under test is described in figure 3 given below:

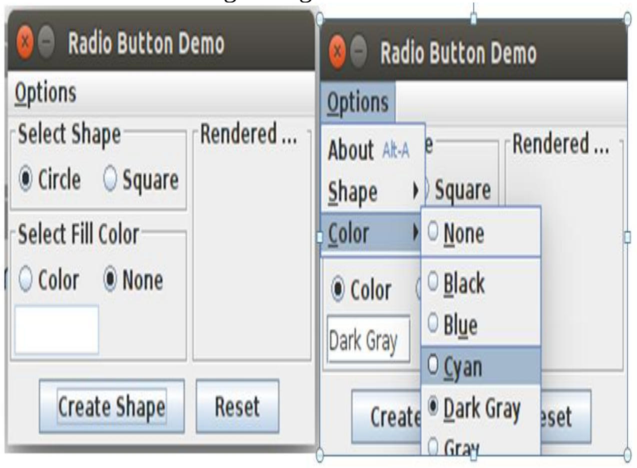

Fig.3. Describes the Snapshot of a Software under test

A. Proposed Algorithm for performing the software GUI Testing

Step 1: Run the GUI ripper on the software under test.

Step 2: Create the Event Flow Graph.

Step 3: Generate the Test cases by the test case generator.

Step 4: Map the fixed number of test case based on their ids.

Step 5: Reduce just stores the result of the test cases in the final repository.

1) Running the GUI Ripper

Before running the GUI Ripper, we have to provide some extra inputs to the tools so that it can identify what exception and terminations in the current system for which test cases are not to be generated are. This extra knowledge is to provide by the user.

2) Creating the Event flow Graph

The Event flow graph is a very helpful tool because without the event flow graph there will be many states that are invalid for testing this was the main disadvantage of generating the test cases with the random approach as it will lot of test cases that are actually invalid. The event flow graph of the software under test generated with the help of reverse engineering is described in figure 4. From the figure 4 has been generated with the help of EFG.XML generated from the tree generated by the ripper.

3) Generate the Test cases by the test case generator

Once we have got the event flow graph now, we can use this graph to generate the cases of the specified sequence length. Each test case generated is a possible transition from the event flow graph. Each test case of the software under test is separate from the other entity, so they are stored separately in the different files by the test case generator.

4) Map the fixed number of test case based on their ids

Once all the test cases are generated the critical task is for the execution of these test cases. As there are a large number of test cases and each execution requires the interaction with the GUI, so it will be a very time-consuming process when done manually, even if done automatically it will require much time. As we can observe that there are a large number of possibilities of test case generation even for the simple application, this can be observed from the Event flow graph described in figure 4. In our proposed idea we have been mapping the test cases based upon their filenames which are the input for the replacer to execute the module. The results generated are stored separately by each of the mapper nodes.

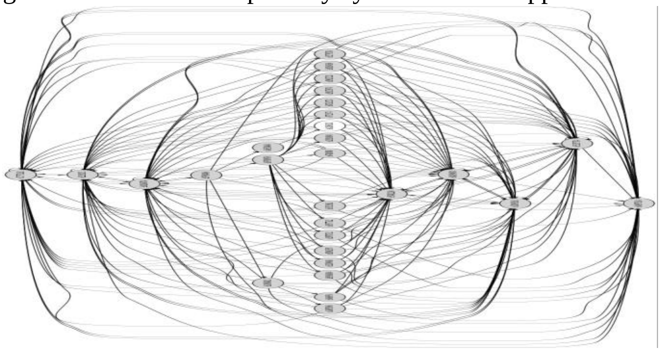

Fig.4. Describing the event flow graph of the software under test

5) Reducer

The results from the mapper are transferred from the mapper to the reducer and the logs detailing the complete snapshot of the system is generated which is analysed by the expert to get fault-prone areas of the software.

IV. Performance Analysis

To test the performance of the software, we have generated the test case for the test sequence length n=2 to n=5. The total numbers of test cases generated are described in Table 2. It can be observed from the table that initially for n=2, we have very limited number of test cases that are 237 as the size of test case increases exponentially to 1719 at n=3. On comparing the relation between the test cases we can observe that in every increase in the size of the test case the number of test cases are increasing by seven times. If the numbers of test cases are increasing exponentially definitely the time required to write these test case to the disk will also increase in that proportion. For the n=2 the process of writing the test cases to the disk requires very less time that was only 8 seconds but in=6 where we have 645219 test cases the time required has increases significantly to 9105.

Table 2. Describing the total number of test cases generated for different sequence number

S.no.	Length of the test suite	Total number of possible sequence generated	Time required to Write the test suite (Seconds)
1	2	237	8
2	3	1719	25
3	4	12379	194
4	5	89399	990
5	6	645219	9105

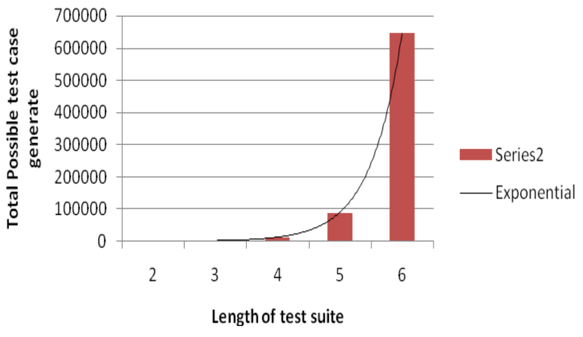

Fig. 5. Comparing the test case generate versus length of the suite

From the figure 5, it is clear that as we are going to increase the length of the suite the complexity to for testing the software is also growing exponentially. As the numbers of test cases are increasing the time taken to generate these test cases is also going to increase. In our study, we have considered all the possible test cases because of the fact whenever the regression testing is performed all the possible test cases are necessary to execute for the satisfaction of the customer. The comparative chart showing the comparison of the time required to generate the test cases are described in figure 6.

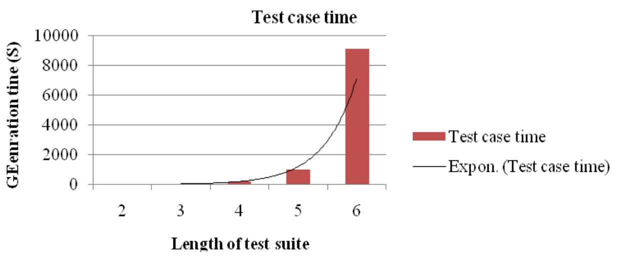

Fig. 6. Comparing the test case generate time versus length of the suite

This time was merely required to generate the test cases for the system what is our objective is to reduce the time complexity of testing the system. This task is done with the help of the replayer tool of the guitar. Initial these test cases are stored on the HDFS. As our design, we can divide the test cases into 4 part as there are four mapper nodes and four reducer nodes. In the mapping phase, the specific test cases are picked at executed and in the reducing phase log files of each of the mapper are combined and placed on the disk for further analysis.

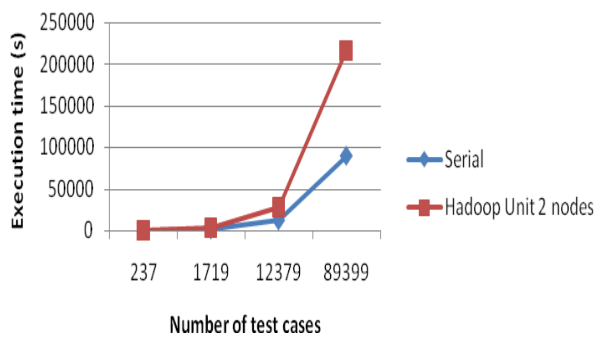

Fig.7. Showing the comparison of the result obtained

From the figure 7, we can observe that we can reduce the significant amount of time by concurrently running the test cases. The time is a very crucial factor for testing the software. As in the testing phase the software may have the changes at the alpha and beta level of testing and the feedback about the issue creates a pressure on the development team to respond quickly. If the team could respond quickly with the tested solution then the software team may not be sure to delivered product is functional properly. The executions of the test cases are taking huge amount of time of the machines. It can be noted that we required almost three days to get into the log reports of the results for the 89399 test cases. So in our study we have reduced our study to n=5 and skipped n=6 as the testing was done on the hardware that is need to run for more that 4 to 5 days which was not possible on the machine we are using. It can be observed that for n=5 that contains the test suite of 89399 test cases took only 1.2 day to complete to its counterpart sequential version which took 2.5 day. The performance can be further reduced by increasing the number of nodes because of the fact there is no interdependency in the test cases but still, some amount of extra time is devoted in writing the final logs by the reducer to the disk.

The study of the speedup achieved has been presented in figure 8. Speedup is the ratio between the times consumed by the application when executed in the serial mode to the time consumed by the same application in the parallel mode. According to Amdhal's law[13] the maximum speedup that can be achieved is always having the upper limit of number of parallel machines employed. This is due to the fact that in any jobs there will be some dependent portion of the task which stops the parallelization [1, 9]. The equation 1 described the formula we have used for the computation of the speedup.

$$Speedup = \frac{Execution\,time\ \ for\ sequential\ version}{Execution\,time\ \ for\ parallel\ version}$$

In the figure 8, initially for n=2 we have very few test cases that are 237, here the execution time of the test cases is significantly very small even though the test cases are uniformly distribute to the two cluster but time required in other operations has affected the performance and the speedup achieved is 1.8. For n=3 this value has improved for n=3 having 1719 test cases to 1.94 similarly for n=4 having 12379 test cases the speedup achieved is 1.95 and for n=5 having 89399 test cases the speedup achieved is 1.96. Here we can see that for 1719 test cases to 89399 test cases the increase in speed up is only by the 0.02. This is due to the fact the maximum speedup that can be achieved by this machine is fixed to less than 2 as the number of cluster or VMS involved are fixed to 2 with the single core.

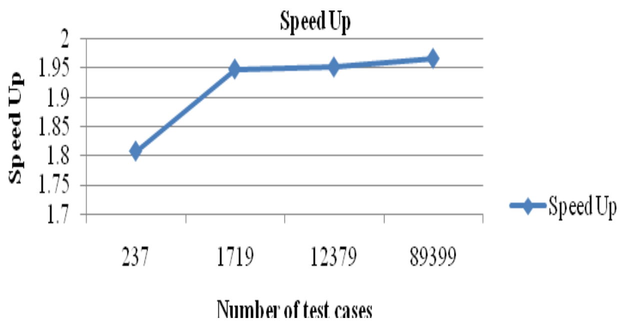

Fig.8. Describing the speed up achieved

V. Conclusion and Future Work

From the results, it is clear that we can reduce the total time required for executing the test cases to almost half. As we have only two cluster nodes so the maximum speedup that can be achieved is limited to 2.0. From the results, we can observe that the maximum speedup achieved is 1.96. We can further improve the results if we can get a large number of the cluster on commercialized machines or in a cloud environment. In future, we are planning to implement the analysis of log files in the Hadoop environment to reduce the time required for the completion of the test cases. Further, the customization in the test case generation part is required to generate the separate test case and reduce the time required for test case generation even though this time is very small in comparison of the running of the test cases.

REFERENCES

1. Ananthanarayanan, G., Malhotra, G., Balakrishnan, M., & Sarangi, S. R. (2013). Amdahl's law in the era of process variation. International Journal of High Performance Systems Architecture, 4(4), 218. https://doi.org/10.1504/IJHPSA.2013.058984
2. Artzi, S., Dolby, J., Jensen, S. H., Møller, A., & Tip, F. (2011). A framework for automated testing of javascript web applications. In Proceeding of the 33rd international conference on Software engineering - ICSE '11 (p. 571). New York, New York, USA: ACM Press. https://doi.org/10.1145/1985793.1985871
3. Artzi, S., Kiezun, A., Dolby, J., Tip, F., Dig, D., Paradkar, A., & Ernst, M. D. (2010). Finding Bugs in Web Applications Using Dynamic Test Generation and Explicit-State Model Checking. IEEE Transactions on Software Engineering, 36(4), 474–494. https://doi.org/10.1109/TSE.2010.31
4. Ashkenas, J., & others. (n.d.). CoffeeScript, 2010. URL Http://coffeescript. Org.
5. Cao, Z., Lin, J., Wan, C., Song, Y., Taylor, G., & Li, M. (2017). Hadoop-based framework for big data analysis of synchronised harmonics in active distribution network. IET Generation, Transmission & Distribution, 11(16), 3930–3937. https://doi.org/10.1049/iet-gtd.2016.1723
6. Choi, W., Necula, G., & Sen, K. (2013). Guided GUI testing of android apps with minimal restart and approximate learning. In Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications - OOPSLA '13 (pp. 623–640). New York, New York, USA: ACM Press. https://doi.org/10.1145/2509136.2509552
7. Chu, C.-T., Kim, S. K., Lin, Y.-A., Yu, Y., Bradski, G., Olukotun, K., & Ng, A. Y. (2007). Map-reduce for machine learning on multicore. In Advances in neural information processing systems (pp. 281–288).
8. Goadrich, M. H., & Rogers, M. P. (2011). Smart smartphone development. In Proceedings of the 42nd ACM technical symposium on Computer science education - SIGCSE '11 (p. 607). New York, New York, USA: ACM Press. https://doi.org/10.1145/1953163.1953330
9. Gustafson, J. L. (1988). Reevaluating Amdahl's law. Communications of the ACM, 31(5), 532–533. https://doi.org/10.1145/42411.42415
10. Hackner, D. R., & Memon, A. M. (2008). Test case generator for GUITAR. In Companion of the 13th international conference on Software engineering - ICSE Companion '08 (p. 959). New York, New York, USA: ACM Press. https://doi.org/10.1145/1370175.1370207
11. Hadoop MapReduce. (2015). Retrieved from http://hadoop.apache.org/
12. Han, L., Liew, C. S., van Hemert, J., & Atkinson, M. (2011). A generic parallel processing model for facilitating data mining and integration. Parallel Computing, 37(3), 157–171. https://doi.org/10.1016/j.parco.2011.02.006
13. Hill, M. D., & Marty, M. R. (2008). Amdahl's Law in the Multicore Era. Computer, 41(7), 33–38. https://doi.org/10.1109/MC.2008.209
14. Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: a review. ACM Computing Surveys, 31(3), 264–323. https://doi.org/10.1145/331499.331504
15. Jiang, B., Zhang, Z., Chan, W. K., & Tse, T. H. (2009). Adaptive Random Test Case Prioritization. In 2009 IEEE/ACM International Conference on Automated Software Engineering (pp. 233–244). IEEE. https://doi.org/10.1109/ASE.2009.77
16. Joorabchi, M. E., & Mesbah, A. (2012). Reverse Engineering iOS Mobile Applications. In 2012 19th Working Conference on Reverse Engineering (pp. 177–186). IEEE. https://doi.org/10.1109/WCRE.2012.27
17. Kaur, H., & Gupta, G. (2013). Comparative study of automated testing tools: Selenium, quick test professional and testcomplete. International Journal of Engineering Research and Applications, 3(5), 1739–1743.
18. McNabb, A. W., Monson, C. K., & Seppi, K. D. (2007). Parallel PSO using MapReduce. In 2007 IEEE Congress on Evolutionary Computation (pp. 7–14). IEEE. https://doi.org/10.1109/CEC.2007.4424448
19. Memon, A. M., Banerjee, I., & Nagarajan, A. (2003). GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing. In WCRE (Vol. 3, p. 260).
20. Memon, A. M., Soffa, M. Lou, & Pollack, M. E. (2001). Coverage criteria for GUI testing. ACM SIGSOFT Software Engineering Notes, 26(5), 256. https://doi.org/10.1145/503271.503244
21. Natesan, P., Rajalaxmi, R. R., Gowrison, G., & Balasubramanie, P. (2017). Hadoop Based Parallel Binary Bat Algorithm for Network Intrusion Detection. International Journal of Parallel Programming, 45(5), 1194–1213. https://doi.org/10.1007/s10766-016-0456-z

22. Nguyen, B. N., Robbins, B., Banerjee, I., & Memon, A. (2014). GUITAR: an innovative tool for automated testing of GUI-driven software. Automated Software Engineering, 21(1), 65–105. https://doi.org/10.1007/s10515-013-0128-9
23. Qian, J., Miao, D., Zhang, Z., & Yue, X. (2014). Parallel attribute reduction algorithms using MapReduce. Information Sciences, 279, 671–690. https://doi.org/10.1016/j.ins.2014.04.019
24. Ramler, R., Winkler, D., & Schmidt, M. (2012). Random Test Case Generation and Manual Unit Testing: Substitute or Complement in Retrofitting Tests for Legacy Code? In 2012 38th Euromicro Conference on Software Engineering and Advanced Applications (pp. 286–293). IEEE. https://doi.org/10.1109/SEAA.2012.42
25. Srinivasan, A., Faruquie, T. A., & Joshi, S. (2012). Data and task parallelism in ILP using MapReduce. Machine Learning, 86(1), 141–168. https://doi.org/10.1007/s10994-011-5245-8
26. Takala, T., Katara, M., & Harty, J. (2011). Experiences of System-Level Model-Based GUI Testing of an Android Application. In 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation (pp. 377–386). IEEE. https://doi.org/10.1109/ICST.2011.11
27. Tilley, S., & Dechokul, K. (2014). Testing iOS Apps with HadoopUnit: Rapid Distributed GUI Testing. Synthesis Lectures on Software Engineering, 2(2), 1–103. https://doi.org/10.2200/S00614ED1V01Y201411SWE003
28. Verma, A., Llorà, X., Goldberg, D. E., & Campbell, R. H. (2009). Scaling Genetic Algorithms Using MapReduce. In 2009 Ninth International Conference on Intelligent Systems Design and Applications (pp. 13–18). IEEE. https://doi.org/10.1109/ISDA.2009.181
29. Wa'el, M. M., Agiza, H. N., & Radwan, E. (2009). Intrusion detection using rough sets based parallel genetic algorithm hybrid model. In Proceedings of the World Congress on Engineering and Computer Science (Vol. 2, pp. 20–22).
30. White, T. (2012). Hadoop: The definitive guide. O'Reilly Media, Inc.
31. Wynne, M., & Hellesoy, A. (2012). The cucumber book: behaviour-driven development for testers and developers. Pragmatic Bookshelf.
32. Zhao, W., Ma, H., & He, Q. (2009). Parallel K-Means Clustering Based on MapReduce (pp. 674–679). https://doi.org/10.1007/978-3-642-10665-1_71

AUTHORS PROFILE

Sumit Kumar is currently working as Ph.D. Scholar in Uttarakhand Technical University, Dehradun, India.He has over 13 years of experience with leading Institutions. He has published more than 10 research papers in reputed journals and international conferences. His research interest area is Finite State Testing of Graphical User Interface. He has received Excellence Award for Best Teaching and Learning Practices(April-2012) by Prof S K Kak (Vice Chancellor MTU).

Nitin is a member of Information Systems Area. In the past he has worked as Professor Educator in the Department of EECS at University of Cincinnati, Cincinnati, OH, and First Tier Bank Professor at Peter Kiewit Institute, University of Nebraska at Omaha (UNO), Omaha, USA. Recently he has been awarded 2017-2018 Outstanding Services to the EECS Department by UC. He has earned all of his degrees: D.Sc., Ph.D., M.Engg. and B.Engg. in the field of Computer Science & Engineering. He has work experience of more than 15 years and published more than 180 Research Papers in peer reviewed International Journals and Conferences. He has bagged more than 40 Academic and Industrial Awards and till now he has guided and awarded 10 Doctoral and 19 Masters Thesis in the area of Computer Science & Engineering and Information Technology. He is a IBM certified engineer and Life Member of IAENG, Senior Member of IEEE and IACSIT and Member of SIAM and ACIS.