IJCCS (Indonesian Journal of Computing and Cybernetics Systems)
Vol.12, No.1, January 2018, pp. 63~72
ISSN (print): 1978-1520, ISSN (online): 2460-7258
DOI: 10.22146/ijccs.28121

Selenium-Based Multithreading Functional Testing

Khabib Mustofa*[1], Sunu Pinasthika Fajar[2]
[1]Department of Computer Science and Electronics, FMIPA, UGM, Yogyakarta, Indonesia
[2]Magister of Computer Science FMIPA UGM, Yogyakarta, Indonesia
e-mail: ***[1]khabib@ugm.ac.id**, [2]sunupf@gmail.com

Abstrak

Pada proyek pengembangan perangkat lunak, pengujian (testing) merupakan aktivitas yang dapat menghabiskan waktu, usaha atau biaya hingga 35%. Untuk mengurangi hal tersebut, pengembang dapat memilih pengujian secara otomatis. Pengujian otomatis, khususnya pengujian fungsional, pada aplikasi web dapat dilakukan dengan memanfaatkan alat bantu, salah satunya adalah Selenium. Secara default, pengujian mengunakan Selenium dilakukan secara berurutan dan tanpa memanfaatkan multithreading, yang berdampak pada waktu yang cukup panjang.

Dalam penelitian ini dikembangkan sebuah platform yang memungkinkan pengguna Selenium melakukan pengujian dan memanfaatkan multithreading dengan bahasa Ruby untuk mempercepat pengujian. Multithreading pada Ruby terbukti mampu mepercepat pengujian fungsional pada aplikasi web secara bervariasi. Variasi terjadi tergantung pada fungsional yang diuji, metode pengujian dan juga jenis perambah yang digunakan.

Kata kunci*— pengujian perangkat lunak, pengujian fungsional, Selenium, multithreading*

Abstract

In a software development projects, testing is an activity that can spend time, effort or cost up to 35%. To reduce this, developers can choose automatic testing. Automated testing, especially for functional testing, on web applications can be done by using tools, one of which is Selenium. By default, Selenium testing is done sequentially and without exploiting multithreading, which has an impact a sufficiently long time.

In this study, a platform that allows Selenium users to test and utilize multithreading with Ruby language to speed up testing was developed. Thr result shows that Ruby's multithreading has proven to be capable of speeding functional testing up on various web applications. Variations occur depending on the functionality being tested, the testing approach and also the type of browsers used..

Keywords—*software testing, functional testing, Selenium, multithreading*

1. INTRODUCTION

Finding defects in a software before its official release is compulsory, as studies revealed that fixing bugs in early steps of software development will minimize cost significantly. Finding such defects in the maintenance phase might incur cost up to 100% compared to that of the design phase [1]**.** One of the approach to achieve that saving is by carrying out a *software testing*. Unfortunately, software testing itself is an activity that might consume about 30% up to 35% of the total effort in the project [2][3]. To overcome such situation, developers choose to apply *automatic testing*, which means that the test can be repeated many times on the software

Received September 7th,2017; Revised January 5th,2018; Accepted January 22th,2018

under test. This automatic testing is carried out also using tools or software.

Among existing testing types, *functional testing* is the most frequently carried out testing using automatic testing approach, including for a web application. Functional testing aims at verifying the software under test whether or not it has met predefined requirements and focusing on comparing input given into its forms with output and status that should be accepted [4]. Several tools are available for functional testing for a web application, one of them is Selenium. By default, script in Selenium is executed in a single thread, leading to consequence that all test cases will be done sequentially or serially. In carrying out functional testing for a web application, access to I/O devices and network device will be much greater than access to CPU, and the testing script will be waiting for more (for the data to be completely processed). This waiting time will be minimized by increasing the access to CPU by multithreading approach. By now, it is still difficult to find tools for web application testing which provide support for multithreading and direct access to web driver API.

Based on what has been described above, this research will discuss the use of Selenium as a tool for functional testing with several test cases provided by users, and they will be executed in multithreading way to improve efficiency and shorten the testing time duration.

An automated testing tools based on Selenium, combined with Fitnesse, was introduced to and enabled users to carry out a testing collaboratively [5]. It was mentioned in the paper that even though Selenium is not a tool to solve any problems in testing, the existence of an increase in its community, both either users or developers, indicates that Selenium was quite promising to be used in functional system testing in future.

Another research related to Selenium has also been conducted on the topic of automated browsing on AJAX website [6]. In the research, Selenium can recognize only some specific events, meanwhile in AJAX requests may happen in various types of event. The reduction of the event was aimed at avoiding event flooding, that is the number of events captured by listener such that it is beyond being effective and efficient.

Beside Selenium, Sahi is also a quite popular tool for automatic testing on a web application domain. Both Selenium and Sahi have their own advantages and disadvantages. In [7], both are compared in terms of test-case creation time, test execution speed, installation and configuration, record and playback, logging and reporting, cost, and platform compatibility. It disclosed that Selenium was leading in terms of test-case creation time, single thread execution time, logging and reporting, and ease of installation, while Sahi provides a feature of executing test cases in parallel which enable users to shorten test execution time if such feature is activated.

Other research on automatic testing using Selenium disclosed that Selenium-based functional tests for the web application are modified as the project evolve [8]. In other words, the paper called that it co-evolved with WAUT (web application under test).

Selenium comes in two version: IDE and Webdriver. Both versions were discussed in [9] which compares *capture-replay* web testing (using Selenium IDE) with *programmable* web testing (using Webdriver). Testing an application requires the development of test suites. These facts mean more time is needed. When programmable web testing is adopted, such time is more expensive but the test suite maintenance is less expensive. In cumulative, using programmable web testing, cost becomes lower compared to that of capture-replay approach.

By default, Selenium does not provide configuration file which makes ease users in doing testing based on existing configuration. An effort to improve testing features based on Selenium by incorporating configuration file, Junit report and parallel testing for each browser into Selenium model is discussed in [10]. This testing was executed in Python.

In [11], an extension of Selenium to accommodate testing web application which accesses database claim that the proposed approach could reduce efforts of executing test case suites up to 88% compared to semiautomated strategy, and 92% compared to manual strategy. The extension will read the contents of the database and compare them with the test output. The access to a database is carried out at the same time the test case is being executed.

2. METHODS

2.1 System Architecture

In this research, a computer will be used with the following configuration:

1. The main computer is running natively on Windows 10 which will act as the tester.
2. A virtual machine is running on top of the Windows to execute Linux Ubuntu version14.10 with a web server inside. The application to be tested is running on this web server.
3. On the main computer, text editor, Selenium Webdriver and Visual Studio have been installed as the tools for the system development.

2.2 Testing the Platform

The proposed platform uses a command line as the interface. It was developed in Ruby and utilizing JSON for storing data and testing configuration to be executed. Basic use of the script to start a testing platform is as follows:

```
$ coba test configuration_file [browser_name]
```

In the above command, the name of the platform is `coba` and its first parameter, `test`, is the command to do the testing. The next parameter is the address of the configuration file. This configuration file might include default browser, address of the Selenium script to be executed, address of the test case file for data input, number of thread to be used (minimum 1), and configuration of the input form to be tested. While the last parameter, *`browser_name`*, may be used if the user wants to override browser configuration. *Figure 1* shows the architecture of the testing platform.

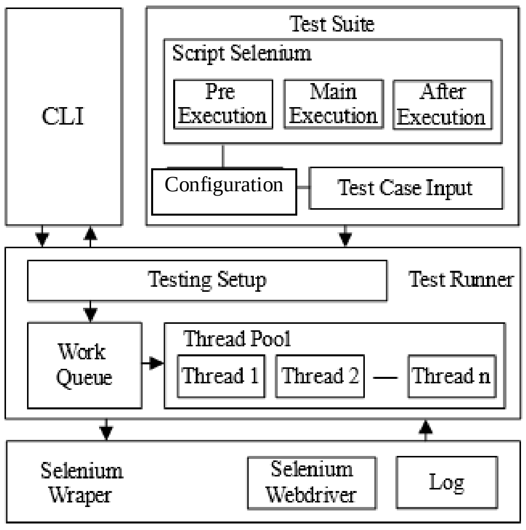

Figure 1 Architecture of the Testing Platform

From *Figure 1*, it can be explained that:

1. Test Suite consists of scripts, configuration, and test-case input

2. Test Suite will be thrown to Test Runner. Here, all the data will be processed as the preparation step before testing
3. In this Test Runner, there will be adjustment or assignment of the port (which will be used to communicate between Selenium with driver), work queue, and thread pool (which will arrange execution of each work queue into thread). As it is important in multithreading to follow *thread-safety*, it is necessary to implement the thread such that shared variables are not overlapping.
   In Ruby, a class named Queue which has the characteristic of following thread-safe concept is used in the implementation of the platform. Besides, as there are some lines of code which need to be executed in interleaving way, a class Mutex (Mutual Exclusion) is used to ensure that the execution of threads is synchronized, thus, consequently, the data consistency can be maintained.
4. Each thread executes Selenium script through Selenium Wrapper. This wrapper contains functions which will execute our scripts. It will also store a log of execution in JSON format and report the execution status via command line interface.

2.3 Test-case Generator

Similar to the testing platform, Test-case Generator runs with command line interface. Test-case Generator is basically a script, written in Javascript, whose main task is to generate test-case consisting of input and output. *Figure 2* shows the architecture of the test-case generator.

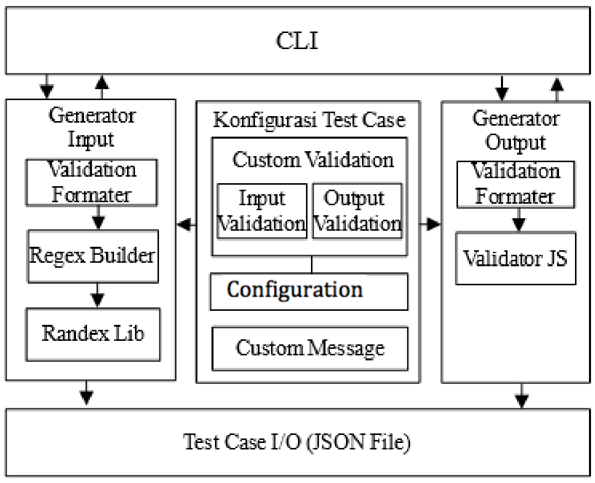

Figure 2 Test-case Generator Architecture

To generate input, a validation is required by the application to be tested. As an illustration, if there is a field `name` of input form that *must* be filled in with minimum 4 characters and maximum 20 characters, then this input field will have validation: *required, min:4* and *max:20*. These validations are then used to generate input for the input field.

There are several validations supported by the proposed platform:
1. required
2. max:x
3. min:y
4. emai
5. alpha

Those validations need be written in a configuration file copied from the template on the phase of test suite composition. For the example of input field `name`, the validation can be represented in the form of an array as shown in *Figure 3*.

It was mentioned above that there is a limited number of validation offered in this platform. If needed, user or tester may add custom validation by creating a file in the validation folder and being saved in Javascript format. All of the input test cases generated will be saved in a JSON file defined in the configuration file.

```
"input":[{
      "type":"name",
      "selector": "[name='name']",
      "name":"name",
      "validation":["required","min:4","max:20"]
   }]
```

Figure 3 Validation for an input field in a configuration file

In generating input test case, the platform utilizing total combination or number of subset formula to determine the number of possible values for the input validation defined. To illustrate the process, the following description will refer to the case as explained in *Figure 3*.

Based on *Figure 3*, it is clear that the validation consists of three elements: *required*, *min:4* and *max:20*. Using total combination formula, it can be calculated that there should be 8 subsets as depicted in *Table 1*.

Table 1 Total combination of input test case with 3 validation

Number of possible combination for 3 elements: 2^3 = 8	
[ ]	[requried]
[min:4]	[max:20]
[required, min:4]	[required,max:20]
[min:4,max:20]	[required,min:4,max:20]

Suppose that an input field satisfies only proper subset of the set containing all validation elements, then it can be said that there is/are missing rule(s), and this missing rule(s) will be used to generate a conclusion determined using a regular expression. In the proposed platform, the regular expressions are formed using Javascript library called *randexp.js*. For a case with more than one input fields, the formula will be adjusted by applying the product of all sets involved. As an illustration, if a form has two input fields: the above-mentioned `name` field and a field whose validations are *email* and *required*. From the second field, it can be derived that the subsets generated would be 4. Thus, there will be 8 x 4 = 32 possible combination of validation cases. In general, the number of possible combination of validation cases could be represented as Eq. 1.

$$TotalInputValidationCase = \prod_{i=1}^{n} C_i \qquad \textit{(Eq. 1)}$$

where $C_i = 2^{k_i}$, k_i is number of validation elements for field *i*.

Output test case generation can only happen when input test case generation has already been completed. The output test case generator will access the file (if exist) containing the results of input test case generation, and then process it to obtain values of possible response from the application to be tested, otherwise, it will throw an error message.

The process of output generation is implemented using library *validatorjs* written in Javascript. The library is suited for Laravel framework. As it provides API to extend or re-define new validation and error message, in this platform a new file *message.js* is created to enable users to overwrite messages by writing the name of validation and its corresponding message in JSON format in a generic format as follows:

```
{validation-name : "error message"}
```

2.4 Testing Design

Testing is carried out using the proposed platform by considering the following factors:

2.4.1 Environment Condition

In order to achieve a good and reliable quality of data, the environment of testing should be adjusted to be in a relatively similar condition. As the testing platform runs on Windows, the operating system is ensured to run in a *clean boot*; that is the OS only loads services necessary for the OS itself and without any third party services running. This conditions can be achieved among others by:

1. disable all non-Windows services. Users can do this step by using dialog provided by Windows from *msconfig* command and then configure such options.
2. disable *startup* program. Users do this configuration by disabling all items in Start Up tab available in Task Manager.

2.4.2 Testing Process

The followings are basic steps in testing the proposed platform.

1. Preparing the test. In this stage, some files must be prepared, depending on the features to be tested. At the minimum, four files should exist: three Selenium scripts (script before testing, testing script, and script after testing) and one configuration file
2. Generating input and output test cases. Validations on the forms to be tested are added into the configuration file. If the testing does not use input/output test case generator, testers/users may create JSON file containing arrays of object accordingly.
3. Testing script creation. A testing script using Selenium is created. The script will be run before the real testing to ensure that the script behaves as expected.
4. Testing on the platform. The script is run on the platform developed. The platform is tested against some functional features of the WAUT: *login*, *register, profile update, activity creation,* and *sub-activity creation*. The testing is done on browsers which have support for Selenium: Internet Explorer version 11.0.15063.0, Firefox version 54.0.1, Chrome version 60.0.3112.90, and PhantomJS version 2.1. The Selenium used in the testing is of version 3.4.4., including its corresponding web driver. When the script is running, elapsed time to run the script on the platform is recorded. There are two different methods for the tests:
   a. Using `sendKeys` method to fill in the form. The method is the method recommended by Selenium as it resembles the input given by real users. The method fills in the input following the generic format

      ```
      element.send_keys parameterList
      ```

      So, an execution of

      ```
      element.send_keys "tet", :arrow_left, "a"
      ```

      will imitate of sending value `"test"`.

b. Using Javascript to fill in the form. The method is introduced as an alternative as sometimes browser does not respond well the execution of the `sendKeys` method. The method fill in the input as illustrated below:

```
driver.execute_script(javaScript_Expression)
```

So, an execution of

```
element = driver.execute_script("return document.body")
```

will assign variable `element` to document body element of an HTML file. Furthermore, the element can be used or referred in the next instruction, as

```
driver.execute_script("return arguments[0].tagName",
                      element)
```

c. Using multithreading. In this method, Javascript method with 4 (four) threads is used.

For each testing, a folder will be created to hold folders representing features of WAUT, such as folder *login, register, profile update,* etc. Each folder will contain configuration and test case different one with another.

Database migration. As the condition of the database might change during the testing process, so it is necessary to restore the database state to the initial condition. This step is done manually each after the completion of the test for each browser.

3. RESULTS AND DISCUSSION

Based on some scenario or functional features as stated above, the following tables (**Error! Reference source not found.**, *Table 3*, *Table 4*, *Table 5* and *Table 6*) show some results of the testing:

1. The percentage in the second row of each table means that there is *positive* or *negative* speedup of the single-thread Javascript compared to the single-thread sendKeys. Negative value means that there is slowdown in performance of the single-thread Javascript testing against the single-thread sendKeys ones.
2. The first percentage of the third row means that there is *positive* or *negative* speedup of the multi-thread Javascript compared to the single-thread sendKeys.
3. The second percentage of the third row means that there is *positive* or *negative* speedup of the multi-thread Javascript compared to the single-thread Javascript.

On the test using the sendKeys method, in general, Internet Explorer is much slower than others. While Firefox, Chrome, and PhantomJS are taking turns recorded the best testing time. But when the test uses Javascript method, inconsistency in performance is shown by PhantomJS and Firefox, as sometimes those browsers experience negative speedup.

There are cases where multi-threaded testing becomes slower. It only occurs in Firefox. The diagnosis of this case might be that high Firefox driver initiation time even becomes higher when multi-threaded testing is performed. This happens because basically multithreading on Ruby does not execute commands fully in parallel, but just being concurrent, and the driver initiation process is a CPU bound. Coupled with a small number of test cases, fast test execution time is not comparable to a decrease in the speed at the driver initiation time.

Table 2 Result of testing for account creation (registration) function

Input Method	Firefox	Chrome	Internet Explorer	Phantomjs
Single Thread (Send Keys)	622.63 s	574.4 s	2,922.56 s	529.96 s
Single Thread (Javascript)	550.76 s (+13.05%)	393.24 s (+46.07%)	1,198.48 s (+143.85%)	379.96 s (+39.48%)
Multi Thread (Javascript)	216,5 s (+187.55% / +154.36%)	167.18 s (+ 243.58% / +135.22%)	390.61 s (+ 648.20% / + 206.82%)	164.92 s (+221.34% / +141.30%)

Table 3 Result of testing for login function

Input Method	Firefox	Chrome	Internet Explorer	Phantomjs
Single Thread (Send Keys)	24.36 s	17.93 s	81.12 s	22.05 s
Single Thread (Javascript)	15.09 s (+ 49.72%)	11.72 s (+ 52.97%)	30.17 s (+ 168.88%)	15.96 s (+ 38.16%)
Multi Thread (Javascript)	17.17 s (+ 41.88% / - 13.78%)	10.90 s (+ 64.50% / + 7.00 %)	16.28 s (+ 398.28% / + 46.04%)	15.34 s (+ 43.74% / + 7.00%)

Table 4 Result of testing for profile update function

Input Method	Firefox	Chrome	Internet Explorer	Phantomjs
Single Thread (Send Keys)	6,446.92 s	8,984.86 s	90,725.02 s	4,940.34 s
Single Thread (Javascript)	9,199.77 s (- 42.70%)	7,925.83 s (+ 13.36%)	8,664.17 s (+ 947.13%)	6,645.12 s (- 34.51%)
Multi Thread (Javascript)	1,300.01 s (+ 395.91% / + 607.67%)	1,251.31 s (+ 618.04% / + 533.40 %)	1,853.60 s (+ 4,794.53% / + 367.42%)	1,123.77 s (+ 339.62% / + 491.32%)

Table 5 Result of testing for activity creation function

Input Method	Firefox	Chrome	Internet Explorer	Phantomjs
Single Thread (Send Keys)	180.67 s	227.95 s	1,251.66 s	182.99 s
Single Thread (Javascript)	103.52 s (+ 74.53%)	103.14 s (+ 121.01%)	239.85 s (+ 421.85%)	100.44 s (+ 82.19%)
Multi Thread (Javascript)	64.64 s (+ 179.50% / + 60.15%	54.11 s (+ 321.27 / + 90.61 %)	79.65 s (+ 1,472.45% / + 201.13%	48.6 s (+ 276.52% / + 106.67%)

Table 6 Result of testing for activation email resend function

Input Method	Firefox	Chrome	Internet Explorer	Phantomjs
Single Thread (Send Keys)	24.57 s	22.50 s	56.09 s	28.38 s
Single Thread (Javascript)	24.71 s (- 0,57%)	23.80 s (- 5,78%)	44.40 s (+ 26,33%)	29.59 s (- 4,28%)
Multi Thread (Javascript)	19.64 s (+ 25.10% /	13.83 s (+ 62.69% /	20.35 s (+ 175.63% /	15.55 s (+ 82.51 % /

	+ 25.81%)	+ 72.09%)	+ 118.18%)	+ 90.29%)

4. CONCLUSION

The experiment carried out on the proposed platform show that multithreading on Ruby can speedup functional testing process applied on the WAUT (Web Application Under Test). The functional test speedup varies depending on the testing methods (sendKeys or Javascript) and functional features under the test.

Among browsers used during the testing, Internet Explorer is always much slower to excute the test, especially when the testing type is using sendKeys.

Javascript method does not always show positive speedup, especially in Firefox and Phantomjs.

In this research, the multithreading approach was implemented in a static way, that is just by using 4 (four) threads. At this point, there is no clue about the relationship between the number of thread used and the speed of testing. It will be a good chance to explore further such relationship to formulate the optimal number of thread for a testing. Besides, the browsers explored so far has not included some popular ones, such as Opera or Safari. Extending popular browsers to be included in the testing using the proposed platform is a challenge.

REFERENCES

[1] P. K. Suri and A. Pooja, "Study of Software Quality and Risk Estimation and Quality Cost Analysis using empirical study," *International Journal Of Engineering And Computer Science*, vol. 4, no. 7, 2015 [Online]. Available: https://www.ijecs.in/index.php/ijecs/article/download/3874/3610/

[2] Anonymous, "Testing Effort Estimation." [Online]. Available: http://www.geekinterview.com/question_details/75715

[3] R. Gupta and N. Bajpai, "A Keyword-Driven Tool for Testing Web Applications (KeyDriver)," *IEEE Potentials*, vol. 33, no. 5, pp. 35–42, Sep. 2014 [Online]. Available: http://ieeexplore.ieee.org/document/6894287/. [Accessed: 18-Jan-2018]

[4] Y.-F. Li, P. K. Das, and D. L. Dowe, "Two decades of Web application testing—A survey of recent advances," *Information Systems*, vol. 43, pp. 20–54, 2014 [Online]. Available: https://doi.org/10.1016/j.is.2014.02.001

[5] A. Holmes and M. Kellogg, "Automating Functional Tests Using Selenium," in *AGILE 2006 (AGILE'06)*, 2006, pp. 270–275 [Online]. Available: http://ieeexplore.ieee.org/document/1667589/. [Accessed: 20-Jan-2018]

[6] P. Montoto, A. Pan, J. Raposo, F. Bellas, and J. López, "Automated browsing in AJAX websites," *Data & Knowledge Engineering*, vol. 70, no. 3, pp. 269–283, 2011 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169023X10001503#

[7] T. J. Naidu, N. A. Basri, and S. Nagenthram, "SAHI vs. Selenium: A comparative analysis," in *Contemporary Computing and Informatics (IC3I), 2014 International Conference on*, 2014, pp. 967–970 [Online]. Available: http://ieeexplore.ieee.org/document/7019594/

[8] L. Christophe, R. Stevens, C. De Roover, and W. De Meuter, "Prevalence and maintenance of automated functional tests for web applications," in *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, 2014, pp. 141–150 [Online]. Available: http://ieeexplore.ieee.org/document/6976080/

[9] M. Leotta, D. Clerissi, F. Ricca, and P. Tonella, "Capture-replay vs. programmable web testing: An empirical assessment during test case evolution," in *Reverse Engineering (WCRE), 2013 20th Working Conference on*, 2013, pp. 272–281 [Online]. Available: http://ieeexplore.ieee.org/document/6671302/

[10] R. A. Razak and F. R. Fahrurazi, “Agile testing with Selenium,” in *Software Engineering (MySEC), 2011 5th Malaysian Conference in*, 2011, pp. 217–219 [Online]. Available: http://ieeexplore.ieee.org/document/6140672/

[11] A. M. F. V de Castro, G. A. Macedo, E. F. Collins, and A. C. Dias-Neto, “Extension of selenium RC tool to perform automated testing with databases in web applications,” in *Proceedings of the 8th International Workshop on Automation of Software Test*, 2013, pp. 125–131 [Online]. Available: http://ieeexplore.ieee.org/document/6595803/