Proceedings of the 34th European Modeling & Simulation Symposium (EMSS), 003
19th International Multidisciplinary Modeling & Simulation Multiconference

ISSN 2724-0029 © 2022 The Authors.
doi: 10.46354/i3m.2022.emss.003

Using HLPC for parallelization of autonomous tests of WEB applications from its GUI

Mario Rossainz-López[1] *, Jesús A. Islas-Fuentes, Ivo Pineda-Torres[1], Manuel Capel-Tuñón[2]

[1] Faculty of Computer Science, Autonomous University of Puebla, Av. San Claudio and 14 Sur Street, San Manuel, Puebla, México, C.P. 72570
[2]Software Engineering Department, College of Informatics and Telecommunications ETSIIT, University of Granada, Daniel Saucedo Aranda s/n, Granada 18071, Spain

*Corresponding author. Email address: mrossainzl@gmail.com

Abstract

A proposal for parallelization of autonomous tests of Web applications from its graphical user interface (GUI) is presented to reduce its execution time since these increases exponentially due to the number of combinations that are generated with the different states of the fields, of the forms on the web application pages by creating a tree-like structure. It is proposed to use the model of high-level parallel compositions or HLPC as a suitable model of semi-automatic parallelization that defines a Tree-like structure as a communication pattern between processes. The proposed HLPC, which we call HLPC-Tree, uses reinforcement learning that associates each node of the process tree as a slave object using the Q-Learning (QL) algorithm and achieves autonomous recognition of the fields of the forms and valid-invalid options to identify failures and display them with HTTP status codes. In addition, the Mechanize library is used to find the number of possible combinations through the states of the fields of the forms and to know how many nodes are generated at each level of the process tree in the HLPC-Tree when it grows in depth. Finally, the performance analysis of the proposed HLPC is shown with an analysis of the speedup and execution times in an 8-core machine to demonstrate good scalability in its accelerations compared to Amdahl's Law.

Keywords: HLPC, reinforcement learning, Q-Learning, Web Application, GUI, Autonomous tests

1. Introduction

Web applications are an important part of our daily lives. They range from entertainment websites to complex and critical web applications such as banking services and e-commerce. All of them demand a high quality to guarantee efficiency, reliability, and security in the use of data and information processing, otherwise, failures will occur that could cause economic losses both for the company that developed the application and for the end-user. To guarantee the correct functioning of a system and particularly of a web application, software tests are carried out that consist of applying methods of verifying expected requirements and identifying and correcting bugs. Software tests are carried out at different levels, with different methodologies and manually or automatically, however, they are limited due to the complexity and time required to carry out a complete verification, since its design and execution require considerable time and it is at this point where the use of parallelism can help speed up the testing process. Web applications have particular characteristics, and these affect the testing process. The proposal to include parallelism in web application testing from its GUI aims to reduce the testing time in a Web application and contribute to the quality of the

development of these applications.

In this work we show a proposal for parallelization of autonomous tests of standard Web Applications, multi-page from its GUI, using Structured Parallel Programming through High-Level Parallel Compositions (HLPC), to reduce the execution time of tests and achieve good performance to get the application's native HTTP errors on machines with multi-core processors.

This paper, therefore, proposes a model of Structured Parallel Programming and Parallel Objects to parallelize software testing methods, which allows acceleration of test execution times, as well as a good performance concerning their sequential execution and helps developers to reduce work times in testing web applications to find bugs and correct them.

2. Literature review

Humanoid is a tool proposed by (Li, Y., et al, 2019) that is based on Deep Learning for automatic testing of Android applications. The tool learns from human interactions, that is, it can generate artificial human-like interactions based on a graphical user interface to test the application. In this proposal, a deep neural network was implemented so that the tool could learn how users interact with the application, and based on this learning, a model was built that generates new test cases. With this tool it is possible to carry out automated tests for Android applications, with greater coverage and speed than other generators of traditional test cases, however, this work considers limitations such as, for example, it presents a low coverage of less than 10%concerningo other proposals.

In (Harries, L., 2019) a framework called DRIFT is proposed that performs automatic software tests based on Q-Learning reinforcement learning through Batch-RL whose algorithms operate under a symbolic representation of the graphical interface and model the state value function through a GNN (Graph Neural Network). This framework can execute the desired functions in an automated manner and performs simple or combined tasks, proving to be efficient for software testing with a wide range of objectives.

In (Nguyen, D.P., 2020) a framework is proposed that allows the automation of functional tests without code in a web application. To do this, it uses ML (Machine Learning) and SVM (Support Vector Machines) to detect and adapt to change and generate efficient test cases. The framework uses Selenium, which is a suite of testing tools for web applications (see Gross, P. and Wang J.T., 2022), its results showed that the system is efficient to carry out automatic tests on most standard websites using generic test cases.

Finally, (Eskonen, J., 2019) deep reinforcement learning is used to perform automatic software exploration and testing, specifically in web applications where functional tests are carried out and communication errors with the backend are analyzed through JavaScript. It was found that it is possible to represent the states of the user interface as a vector or image to operate the reinforcement learning algorithms. Unlike other techniques based on supervised learning, it does not have the problem of requiring large volumes of data to achieve the training of an efficient test model. In this approach, the testing time for a real application can take whole days and it is not known when to stop the training to reach optimal efficiency.

As can be read, most of the proposals cited here, although they are automated, require human assistance on the one hand and, on the other hand, the execution of the analysis of tests that they carry out on web applications takes a lot of time. Our proposal to parallelize the autonomous tests through HLPCs proposes a novel alternative to reduce the execution time of said test analysis and provide good performance in computers with multicore processors.

3. Tests in WEB applications

Testing web applications consists of executing the application in its different states using different combinations of data inputs to verify its response to each of these combinations and rule out defects in the application. Depending on the type of fault found, certain errors can be attributed to the runtime environment where the application is running. The tests of a web application can be divided into 2 large groups, functional requirements tests, and non-functional requirements tests. The first ones involve the verification of the services and specific functionality of the application (Lucca, F.A.D., 2006), that is, the behavior directly related to the business logic for which it was developed, while the second ones involve tests that have nothing to do with the services that the application provides but with the level of quality with which the application responds in different circumstances. The standard process for testing software systems is to design the test, run it, identify problems, errors, or bugs, and fix them.

3.1. Non-Functional Test (Lucca, F.A.D., 2006)

- Load tests: Evaluates if the performance of the application is as expected under certain conditions and a certain number of users.
- Performance test: Evaluates the performance of the application using parameters such as response time and service availability.
- Volume tests: Evaluates the performance and behavior of the application with a large volume of data.
- Stress tests: Evaluates the behavior of the application under conditions of use that exceed

the specifications for which it was designed.

- Reliability test: Evaluates if the application is reliable.
- Usability tests: Evaluate if the application is easy to learn and use for end-users.
- Compatibility tests: Evaluates if the application behaves correctly when running in different hardware and software environments.
- Security tests: Evaluates the ability of the application to defend against unauthorized access attempts and the ability to maintain the integrity of the application.

3.2. Functional Test (Lucca, F.A.D., 2006)

The functional tests verify that the services and functionality provided by the application do not present faults and behave according to the requirements under which it was developed, for example, user registration and authentication, consultation, deletion and writing in the database, operations with data, upload and download of files, generation of dynamic views, etc.

3.3. Test Level (Umar, M.A., 2019)

- Unit Tests: Individual components that have specific tasks within the application are tested on both the server and client sides and validated that they work as expected.
- Integration tests: The integration of the different unit components of the application is tested to validate their correct operation when they interact with each other.
- System Tests: The application is tested when all its unit components, both on the server and client-side, are fully integrated.
- Acceptance tests: the tests are intended to determine whether the system is accepted for release and production based on the requirements and quality level requested.

3.4. Strategies

- Black box testing: It does not require knowing how the application to be tested is built, the system is treated as a black box to which inputs are provided and its outputs are analyzed to determine if it is the expected one based on the software requirements.
- White box tests: It is required to know the internal structure of the application and have access to the system code. Tests are performed for security holes, broken data streams, cycle integrity, etc.
- Gray box testing: Combines black box and white box testing to perform unit verification of the application.

4. High Level Parallel Composition - HLPC

Using an OO-programming environment, the idea is to implement any type of parallel communication patterns between the processes of an application or distributed/parallel algorithm. An HLPC comes from the composition of a setoff three object types: an object manager (Figure 1) that represents the HLPC itself and makes an encapsulated abstraction out of it that hides the internal structure (McCool, et al, 2012; Rossainz, M. and Capel, M., 2017). The object manager controls a set of objects references, which address the object collector and several stage objects and represent the HLPC components whose parallel execution is coordinated by the object manager. The objects stage are objects of a specific purpose, in charge of encapsulating a client-server type interface that settles down between the manager and the slave objects. And a collector object, we can see an object in charge of storing the results received from the stage objects to which is connected, in parallel with other objects of HLPC composition. During a service request, the control flow within the stages of an HLPC depends on the implemented communication pattern. Manager, collector, and stages are included in the definition of a PO (Corradi, A. and Leonardi, L., 1991; Rossainz, M. and Capel, M., 2017). POs are active objects, which have intrinsic execution capability. Applications that deploy the PO pattern can exploit the inter-object parallelism as much as the intra-object parallelism. A PO-instance object has a similar structure to that of an object in C++, and additionally defines a scheduling policy that specifies which one or more operations carried out by the instance synchronize. The communication modes used are Synchronous communication, asynchronous communication, and the asynchronous future. The Synchronization policies are expressed in terms of restrictions; for instance, mutual exclusion in reader/writer processes or the maximum parallelism allowed for writer processes.

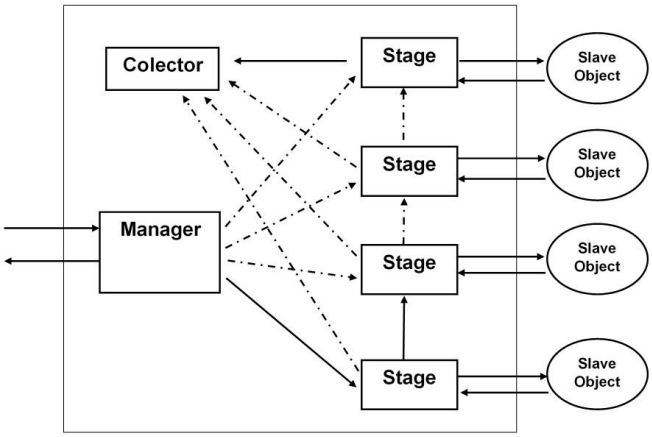

Figure 1. Abstract model of an HPLC

4.1. Representation of Tree as communication pattern between processes as HLPC

The representation of the patron tree that defines the technique of it Divide and Conquer as HLPC has their model represented in Figure 2. This parallel solution offers the prospect of traversing several parts of the tree simultaneously in the HLPC Tree. Once a division is made into two parts, both parts can be processed simultaneously executing the sequential algorithm contained in the slave object associated with the nodes of the tree. Though a recursive parallel solution could be formulated. One could simply assign one process o thread to each node in the tree (Danelutto, M., Torquati, M. 2014)

This model can be easily extended using object-oriented properties (such as inheritance, polymorphism, and abstraction) to an HLPC that represents N-arity trees as communication patterns between processes.

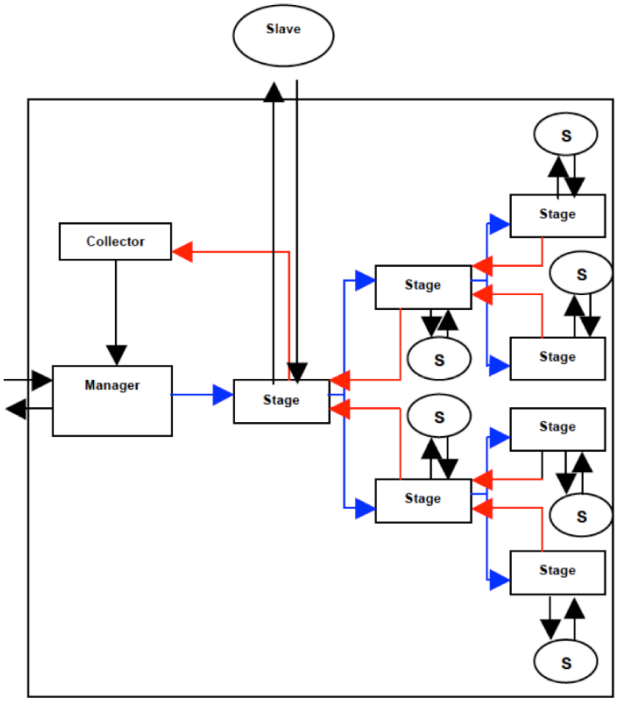

Figure 2. HLPC-Binary Tree Abstract Model

5. Test parallelization process

The characteristics of the type of web application that was worked on in this article are the following:

- Standard and traditional web application is written in HTML with a structure based on URLs.
- Dynamic Web application under the client-server model.
- Web Application MPA (Multi-Page Application) and/or WEB Application SPA (Single Page Application).
- Web application with client-side JavaScript code.

The tests considered in this work are restricted to integration tests through the GUI of the web application and comply with the following:

- The tests are carried out with the interaction of the graphical User Interface of the Web application.
- The web application directly contains the backend.
- The types of errors considered in the tests come from the backend of the application through the basic HTTP responses:
    - HTTP 2xx status codes, for example, 203 – Non-Authoritative Information.
    - HTTP 3xx status codes, for example, 307 – Temporary Redirect.

- HTTP 4xx status codes, for example, 404 -Not Found.
- HTTP 5xx status codes, for example, 500 – Internal Server Error.

· The black box testing strategy is used. It does not consider how the backend is built.

· It interacts with the GUI of the application.

We assume a web application with 3 pages, each one with a form (to show the interaction with the GUI) and each form with 3 input fields with different states, for example:

1. Web page 1: Form with 3 fields, a menu with 3 states (options), a checkbox with 2 states, and a text field with two states. Total possible combinations: 3X2X2=12.
2. Web Page 2: Form with 3 fields, a menu with 4 states, 2 checkboxes, each with two states. Total possible combinations: 4X2X2=16.
3. Web page 3: Form with 3 fields, two menus each with 3 states, and a checkbox with two states. Total combinations: 3X3X2=18.

The total number of combinations of the possible states of the web application is 12X16X18= 3456. Figure 3 shows the graphical representation of the application context using a site map.

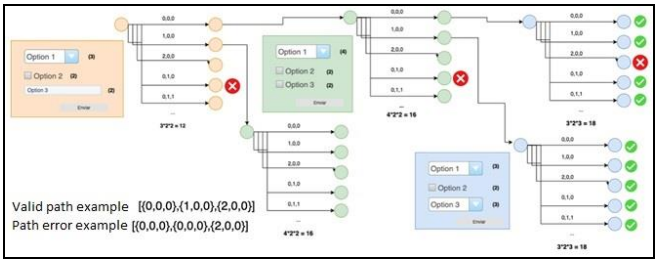

Figure 3. Contextual representation (site map) of a web application with 3 pages, each with a form

In this example, 10 HTTP 500 errors and 35 HTTP 404 errors were intentionally incorporated into the web page forms.

In Figure 3 the navigation of the web application through its different forms creates a tree where the root is the home page from which the tree grows in depth through the navigation of the pages (see Figure 4).

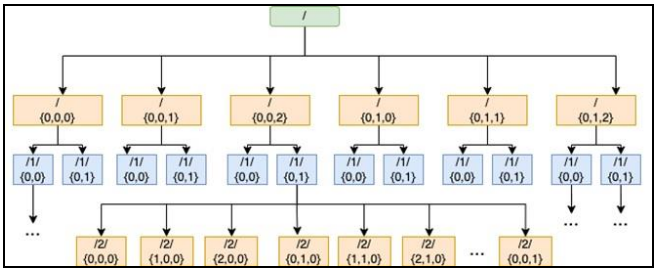

Figure 4. Tree generated by the navigation in the Web application of the example regarding the different states of the fields of the forms

The tests of the web application were carried out in an automated way traversing the tree, using the Mechanize opensource library for direct interaction with the HTML code through the HTTP protocol for filling out forms (see Kovid Goyal, 2017) and the use of reinforcement learning using the Q-Learning (QL) algorithm for the autonomous control of forms, regardless of the total number of fields, their order and the options available in each of them (Barto G., 2014: Jang B., 2017).

Based on this analysis, the proposed parallelization consists of taking the HLPC from section 4.1 as a base and extending it using inheritance, abstraction, and polymorphism to implement a new HLPC that generates the tree in figure 4.

The Manager object receives the URL of the Web application and creates a first Stage process as the root of the tree (home page). The tree is created at runtime, since we do not know the level of depth it will have, nor the number of nodes that will be created by levels, this depends on the combinations of the possible states of the fields of each form on each web page.

Each tree node represents a Stage process that runs in parallel with the others that are at the same tree level and a slave object is associated with them that contains the Q learning algorithm for the autonomous execution of forms in deep learning.

The possible states of the form fields are obtained with the Mechanize library and the possible combinations that define the number of tree nodes at a given level are generated.

When advancing through the tree with the divide and conquer technique, the tests of the current web page of the application are carried out and the errors found by the Stage objects are identified (not necessarily in all of them, only in those where a fault is found) and they send to the collector of the HLPC. The Collector process collects them and sends them to the Manager process, which in turn sends them as final output. The Manager process controls the parallel execution of stages in deep learning. The graphic model of the proposed HLPC is shown in Figure 5.

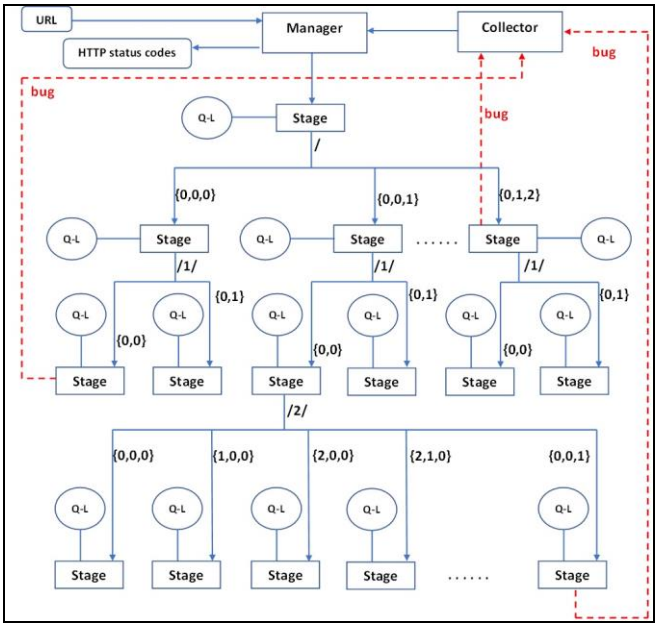

Figure 5. Model of the HLPC-Tree for autonomous tests of a multi-page WEB application

6. Performance

A particular web application of seven pages was created, each of them with a form with different fields and states. Similarly, a series of HTTP errors were intentionally created. The errors considered were those represented by status codes 2xx to 5xx (see Figure 6). The performance analysis of the HLPC-Tree was carried out in its execution for the autonomous tests of this web application.

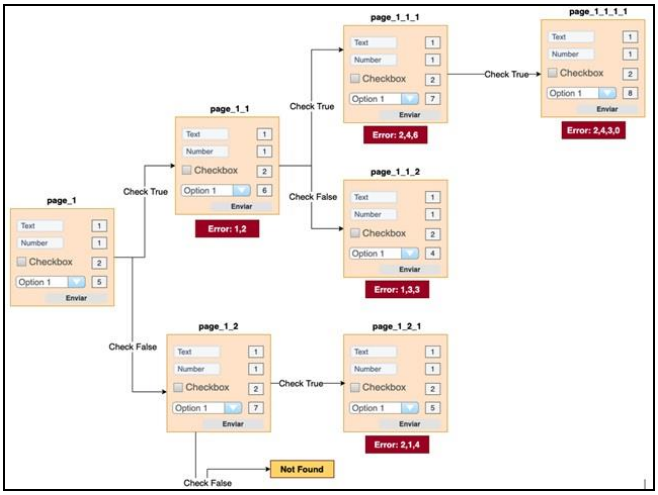

Figure 6. Contextual representation (site map) of the web application specifically for the performance analysis of the HLPC-Tree

Figure 6 shows that each page shows a form, each with 4 fields: A "text field" with a single state (mandatory), a "number field" with a single state (mandatory), a "checkbox" with two states possible (true or false) and the "select" field with different states on each page (from five to eight possible options). Table 1 shows in detail the possible states of each field in each form of the 7 pages of the application and the total number of combinations.

Table 1. Possible states of the forms of the 7 web pages of the application

Form	Possible States
Page_1: TextField= 1 NumberField=1 Checkbox= 2 Select= 5	1*1*2*5 = 10 Subtotal= 10
Page_1_1: TextField= 1 NumberField=1 Checkbox= 2 Select= 5	1*1*2*6 = 12 Subtotal= 10*12=120
Page_1_1_1: TextField= 1 NumberField=1 Checkbox= 2 Select= 5	1*1*2*7 = 14 Subtotal= 120*14=1680
Page_1_1_1_1: TextField= 1 NumberField=1 Checkbox= 2 Select= 5	1*1*2*8 = 16 Subtotal= 1680*16= **26880**
Page_1_1_2: TextField= 1 NumberField=1 Checkbox= 2 Select= 5	1*1*2*4 = 8 Subtotal= 10*12*8=**960**
Page_1_2: TextField= 1 NumberField=1 Checkbox= 2 Select= 5	1*1*2*7 = 14 Subtotal= 10*14= 140
Page_1_2_1: TextField= 1 NumberField=1 Checkbox= 2 Select= 5	1*1*2*5 = 10 Subtotal= 140*10=**1400**

In total, the web application has 26880+960+1400= 29240 possible states in its navigation that are explored by the HLPC-Tree to detect HTTP errors. The GUIs of the web application pages were created with the Django framework (for details see Django, 2021), HTML, CSS, and Javascript code, which allowed easy handling of error mapping and testing of the web application under different scenarios.

The execution of the HLPC-Tree (see Figure 5) was carried out on an 8-core computer with 16 Gb of main memory and memory shared by the stages or nodes of

the tree and other parallel objects (manager and collector). Speedup and maximum acceleration measurements were made using Amdahl's Law and these measurements were compared with its sequential execution, the results of which are shown in Table 2 and Figure 7.

Table 2. Results of the HLPC-Tree acceleration analysis in autonomous tests of the Web application in Figure 6.

CPAN-Tree	SEQ	2 CORES	4 CORES	6 CORES	8 CORES
Runtime Secs	1380	1020	780	720	540
Speedup	1	1.35	1.77	1.92	2.56
Amdahl	1	1.67	2.50	3.00	3.33

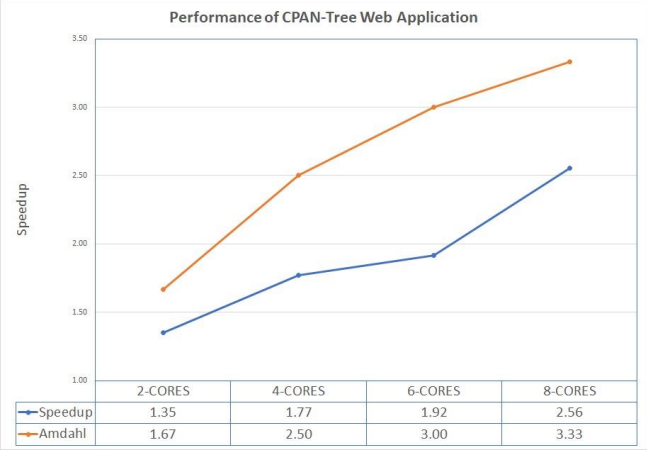

Fig. 7. Speedup scalability found for HLPC-Tree in autonomous tests of WEB application of Figure 6.

Under these execution conditions, the workload in the cores is considered sufficient to show a good performance of the HLPC-Tree.

The execution was carried out in 2, 4, 6, and 8 exclusive cores whose times were measured in seconds. In all of them, a much shorter time is obtained than the sequential execution time, which is almost 24 minutes, achieving a decrease of 17, 13, 12, and 9 minutes, respectively (see Table 2). The speedup values are shown in acceleration when increasing the number of cores, and always below the maximum acceleration level (Amdahl's Law), obtaining good performance of the proposal.

The stages represent the leaves of the tree that is proposed as the HLPC-Tree model. Each stage is a thread of execution that is executed in parallel with the rest of the stages that are created at the same level of the tree and concurrently with respect to the parallel computer architecture that is used. We are in the presence of nested stages as the tree increase by levels.

The tests carried out on the web page created expressly for our work were done on a multi-core computer with 8 cores. With this we have greater control of the cores by managing them exclusively so that they attend only to the execution of our proposal, avoiding attention to other jobs as much as possible. On a server we can hardly have that control. That is one of the reasons for the speedup results obtained.

7. Conclusions

In this work, a structured parallel programming proposal has been presented to improve the execution times of the autonomous tests that are carried out in web applications from their GUI. The proposal consisted of using the model of high-level parallel compositions or HLPC to adapt it to a particular model that uses a tree appropriate to the problem posed as a communication pattern between the processes. We worked with a multi-page web application created for this purpose. In total there were 7 pages, each with a form and each form with 4 input fields: a text field, a number field, a checkbox, and a multi-state select. The number of possible navigation states in the application was 29,240. Reinforcement learning was used, associating the Q-Learning (QL) algorithm to each node (stage) of the HLPC tree to achieve autonomous recognition of the form fields and valid and invalid options and identify failures and display them with HTTP status codes 2xx to 5xx (for details see Barto G., 2014: Jang B., 2017). To find the number of possible combinations of the states of the form fields and for the HLPC to know how many nodes to generate at each level of the tree, the Mechanize library was used. Finally, the performance analysis of the HLPC-Tree is shown, which proved to be good. The performance analysis shows the speedups found and their execution times (CPU usage) which demonstrates the good performance of the HLPC on an 8-core machine and the good scalability of the speedups compared to Amdahl's Law.

References

Barto G., (2014). Reinforcement Learning: An Introduction. Stanford University.

Corradi A. and Leonardi L. (1991). PO Constraints as tools to synchronize active objects. Journal Object Oriented Programming 10:42-53.

Danelutto, M., Torquati, M. (2014). Loop parallelism: a new skeleton perspective on data parallel patterns. Proc. of Intl. Euromicro PDP, Parallel Distributed and Network-based Processing, Torino, Italy.

Django, (2021). The web framework for perfectionists with deadlines. https://www.djangoproject.com

Eskonen, J., (2019). Deep Reinforcement Learning in Automated User Interface Testing. Alto University,

9-30.

Gross, P., Wang J.T. (2022). Selenium automates browsers. https://www.selenium.dev/

Harries, L., (2019). DRIFT: Deep Reinforcement Learning for Functional Software Testing. 33rd Deep Reinforcement Learning Workshop (NeurIPS 2019), 1-8.

Jang B., (2017). Q-learning Algorithms: A Comprehensive Classification and Applications. Department of Computer Science, Sangmyung University.

Kovid Goyal (2017). Mechanize. https://mechanize.readthedocs.io/en/latest/

Li, Y., Yang, Z., Guo Y. and Chen X. (2019). Humanoid: A Deep Learning-Based Approach to Automated Black-box Android App Testing. 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), 1070-1073.

Lucca, F.A.D., (2006). Web Application Testing. Web Engineering. 219-260.

McCool M., Robison A.D. and Reinders J. (2012). Structured Parallel Programming. Patterns for Efficient Computation. Morgan Kaufmann Publishers Elsevier. USA.

Nguyen, D.P., (2020). Codeless web testing using Selenium and machine learning. 15th International Conference on Software Technologies, 2-7.

Rossainz M. and Capel M. (2017). Design and implementation of communication patterns using parallel objects. Especial edition, Int. J. Simulation and Process Modelling, 12:1.

Umar, M.A., (2019). A comprehensive study of software testing: Categories, levels, techniques, and types. International Journal of Advanced Research, Ideas, and Innovations in Technology.