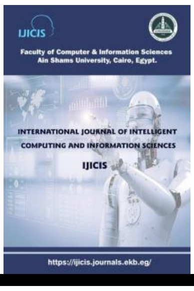
IJICIS, Vol.22, No.4, 35-50
DOI: 10.21608/ijicis.2022.161846.1219

International Journal of Intelligent Computing and Information Sciences

https://ijicis.journals.ekb.eg/


IJICIS
Faculty of Computer & Information Sciences Ain Shams University, Cairo, Egypt.
INTERNATIONAL JOURNAL OF INTELLIGENT COMPUTING AND INFORMATION SCIENCES
IJICIS
https://ijicis.journals.ekb.eg/

AUTOMATION OF PERFORMANCE TESTING: A REVIEW

Amira Ali* Huda Amin Maghawry Nagwa Badr

Information Systems Department, Faculty of Computer and Information Science, Ain Shams University, Cairo, Egypt
Amiraaly@cis.asu.edu.eg

Information Systems Department, Faculty of Computer and Information Science, Ain Shams University, Cairo, Egypt
Huda_amin@cis.asu.edu.eg

Information Systems Department, Faculty of Computer and Information Science, Ain Shams University, Cairo, Egypt
Nagwabadr@cis.asu.edu.eg

Received 2022-09-09; Revised 2022-10-10; Accepted 2022-10-12

Abstract: *Quality assurance of software applications becomes crucial for achieving a competitive advantage in the market. Automating the testing process reduces the required cost and human effort. However, automating the entire testing process is still challenging for both academia and industry. It is essential to verify the robustness of the application under test (AUT) by conducting performance testing. Performance testing is a non-functional form of software testing which examines the performance features of the AUT when exposed to various workloads. The performance behavior can be measured by throughput, response time, and resource utilization of the AUT under a certain workload. Besides, performance testing finds the performance breaking points and bottlenecks during the operation of the AUT. Due to the necessity of conducting performance testing before releasing applications to the market, this paper surveys the previous related work to performance testing since 2009. Recent studies related to performance testing for testing both mobile and web-based applications are discussed. The strengths and weaknesses of these studies are discussed. Besides, a comparison between the previous studies related to performance testing is held from different perspectives.*

Keywords: *Performance Testing, Performance Testing Automation, Mobile Apps, web-based Apps, Performance Testing Tools.*

1. Introduction

Performance testing is a form of non-functional software testing. It tests the applications' behavior when exposed to a certain workload [1]. A workload refers to the number of concurrent users' access to the application under test (AUT). Performance testing evaluates the reliability, scalability, and responsiveness of the AUT under different workloads. Additionally, performance testing aims to identify the functional

***Corresponding Author**: Amira Ali

Information Systems Department, Faculty of Computer and Information Science, Ain Shams University, Cairo, Egypt

Email address: Amiraaly@cis.asu.edu.eg

issues that may occur in the AUT under heavy workloads. Thus, performance testing is an essential step that reveals diagnostic information required to detect bottlenecks in the AUT. Microsoft Performance Guide mentions multiple types of performance testing [2], such as (i) load testing; (ii) stress testing; (iii) Spike testing; (iv) Endurance testing; (v) Volume testing; (vi) Scalability testing. Load testing refers to testing the AUT when exposed to a normal workload. Stress testing measures the reliability of the AUT when exposed to a heavy workload, beyond the normal workload. Spike testing is a sub-type of stress testing that measures the performance of the AUT during exposure to a temporarily heavy workload that increases quickly and repeatedly for short amounts of time. Endurance testing evaluates the performance of the AUT under an ordinary workload for a prolonged amount of time. Endurance testing aims to examine the system problems such as memory leaks, which affect the system's performance or may cause its failure. Scalability testing involves increasing the workload gradually. Also, the workload may remain steady during the variation in the resources such as CPUs and memory. Volume testing measures the efficiency of the AUT when exposed to large amounts of data.

The process of performance testing includes the following steps [3], [4], [5]:

- Step 1: Determine the required performance test environment.
- Step 2: Determine the performance acceptance Criteria for the AUT.
- Step 3: Design performance test cases.
- Step 4: Configure the performance testing environment.
- Step 5: Implement the designed performance test cases.
- Step 6: Execute performance test cases.
- Step 7: Analyze the performance test results.
- Step 8: Generate a performance test report.

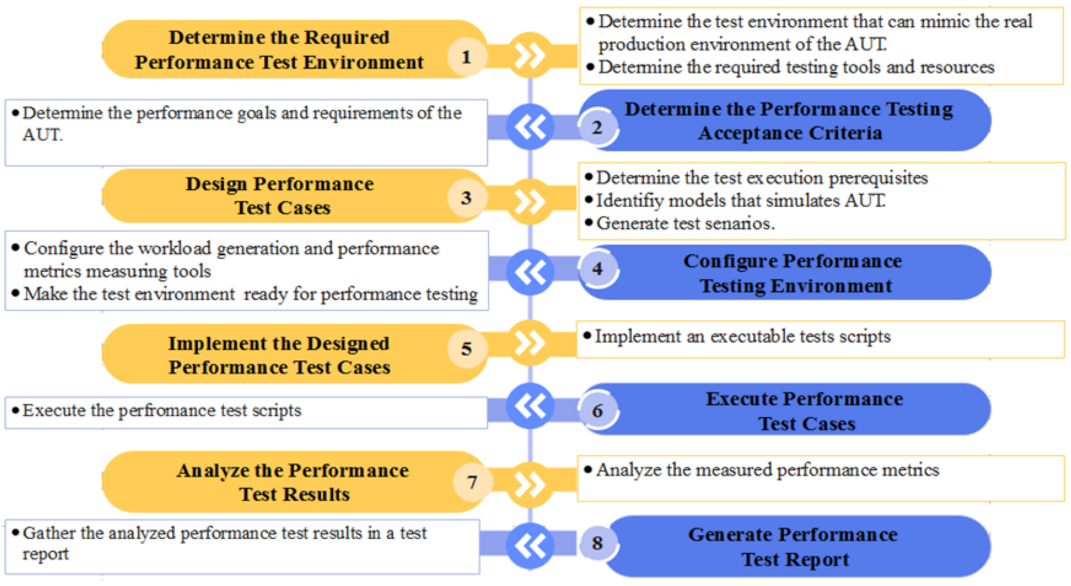

Determine the Required Performance Test Environment
1
• Determine the test environment that can mimic the real production environment of the AUT.
• Determine the required testing tools and resources
• Determine the performance goals and requirements of the AUT.
2
Determine the Performance Testing Acceptance Criteria
Design Performance Test Cases
3
• Determine the test execution prerequisites
• Identifiy models that simulates AUT.
• Generate test senarios.
• Configure the workload generation and performance metrics measuring tools
• Make the test environment ready for performance testing
4
Configure Performance Testing Environment
Implement the Designed Performance Test Cases
5
• Implement an executable tests scripts
• Execute the perfromance test scripts
6
Execute Performance Test Cases
Analyze the Performance Test Results
7
• Analyze the measured performance metrics
• Gather the analyzed performance test results in a test report
8
Generate Performance Test Report

Figure. 1: The overall cycle of the performance testing process

Figure 1 shows the overall cycle of the performance testing process. This is considered the best practice for the process of performance testing [3].The purpose of performance testing is to measure the performance metrics of the AUT, then detect performance issues that exist in the AUT. Performance metrics reflect the quality and responsivity, as well as the performance behavior of the AUT. The widely used performance metrics include the following [3], [6]:

- Response time: refers to the time spent between sending a request to the server and receiving the response. It is measured in kilobytes per second.
- Throughput: refers to the number of requests/transactions processed in a certain amount of time during the test. It shows the amount of the required capacity that the AUT can handle. Throughput depends on the number of concurrent users.
- Wait time: It is called the average latency. It refers to the time taken until the developer receives the first byte after sending a request.
- Average load time: refers to the average amount of time taken to receive each request. It reflects the quality and the responsivity of the AUT from the user's perspective.
- Error rate: refers to the ratio between the failed requests and all requests. The ratio is calculated in percentage. The failed requests always occur when the load exceeds the capacity of the AUT.
- CPU utilization: refers to the time spent by the CPU to process a request.
- Memory utilization: refers to the amount of memory needed to process a request.

Performance is considered an important quality attribute for all software applications. The presence of performance issues in any application will lead to negative consequences on the user experience [7]. the response time of any application whether mobile or web-based application is critical from the user experience (UX) point of view [8]. However, performance testing is always left until the end of the testing process, when no excess time remains due to business pressure. Thus, performance testing is ignored during the development of many applications [9]. Additionally, the process of performance testing is considered an expensive process due to the need to prepare the following: an appropriate test environment, tools to simulate the actual simultaneous user access to the AUT, and adequate test cases which imitate real users' transactions. Recently, performance testing became a challenge for all types of software applications (e.g., Android apps, iOS apps, web services) to find performance breaking points and improve AUT robustness [10].

This paper provides a brief review of the previous work which is relevant to the automation of the performance testing process. A comprehensive review of the previous research is presented. Multiple studies related to automating the performance testing process for different types of applications are discussed to show their features and drawbacks. Besides, excessively utilized commercial performance testing tools are presented with a comparison of their features.

The remainder of the paper goes as follows: Section 2 surveys the most relevant work related to the automation of the performance testing process for both mobile and web-based applications. Section 3 compares widely used performance testing tools in the market. Section 4 compares the previous work related to the automation of the performance testing process and discusses their strengths and weaknesses. Finally, section 5 includes the conclusion.

2. The Automation of Performance Testing

This paper focuses on discussing the techniques for automating the performance testing process. In this section, the previous work in the field of performance testing will be surveyed. The process of automating performance testing for mobile applications is different from the case of testing web-based applications. The architecture of mobile application development is not the same as that of web-based applications. In the literature, many researchers focused on testing a particular type of application whether web or mobile. Hence, we categorize the past relevant research into two categories according to the type of AUT whether mobile, web-based, or cloud application. Sections 2.1, 2.2, and 2.3 review the previous studies related to performance testing for mobile, web-based applications, and cloud applications, respectively.

2.1. Performance Testing for Mobile Applications

The widespread of smartphones leads to the growth of mobile applications development [11]. Mobile applications need to be tested for their functional requirements satisfaction as well as their efficient performance [12]. Mobile applications rely on processing speed, memory usage, battery level, and network type. Many authors proposed approaches for automating the process of mobile application testing. However, the majority of the proposed approaches only addressed the functional aspects of mobile applications. A limited number of authors are concerned with the non-Functional Testing perspective, especially Performance testing for mobile applications [13], [14], [15].

Online mobile applications are widely used in different aspects of our daily life such as online shopping, ticket booking, and E-commerce). Testing the performance of this type of application is introduced in the literature. For example, an online mobile applications testing framework called (Test My APP) was proposed by Rajan et al [16]. The proposed framework measured the response time of online mobile applications. The response time values were recorded iteratively under various device and network conditions during the testing process. Then, the author applied the chi-square test distribution [17] to determine the overall performance evaluation of the AUT accurately. The average response time value was not accurate. However, the author depended on a set of recorded test cases, which were stored in a database. Hence, the proposed approach depended on the testers' participation to generate and store test cases in a database. The author was not concerned with the automatic generation of efficient test cases. The author only focused on calculating AUT response time including network delay and hardware delay iteratively. Then, the author got the Chi-Square value that represented the overall performance evaluation measure of AUT. The author ignored the other performance metrics such as resource utilization, and throughput. The proposed framework needs the Android device to be connected to the test machine using a USB connector or through a wireless connection. Additionally, the proposed framework depended on the tester to set up the required test environment manually.

Mobile applications are usually characterized by a stumpy development lifecycle to earn a competitive advantage. Some organizations conduct performance testing at the system test level. Sometimes, system-level performance testing is imperfect due to the limitations of the mobile applications development environment. Therefore, there are studies in the literature that consider the early accomplishment of performance testing at the unit testing level of mobile applications, rather than focusing such tests in the final stage of development. Unit-level performance testing processes can be conducted in a similar manner to conducting system-level testing. Besides, unit-level performance testing suits organizations that work according to the agile working model [18].

Several studies presented frameworks for unit-level performance testing, such as A method for performance testing based on Test-Driven Development (TDD) proposed by Kim et al [19], [20]. The proposed method relied on a Mobile Performance Benchmark Database (MObilePBDB) and PJUnit tool. The MObilePBDB was considered the basic building block in the proposed performance unit test method. The author used the MObilePBDB to gather performance data generated from the benchmark testing conducted on real mobile devices. Then, the author utilized these data during the performance unit testing conducted in an emulator-based test environment. Besides, the author implemented an Eclipse plug-in called PJUnit. PJUnit created and executed performance test scripts. This diminishes the time and cost requisite for performance testing and helps to find errors in the AUT. The author generated a test case based on the source code of AUT as an input. However, AUT’s source code maybe not be available in some cases. Additionally, the author focused on the test cases execution, but the author did not matter about improving the test execution by applying a parallel test execution method. Additionally, there was no test results analysis method mentioned by the author.

Some performance testing frameworks presented in the literature did not depend on the source code of the AUT. For example, AppSpeedXray is an open-source mobile applications performance measurement and analysis framework. AppSpeedXray was proposed by Mun et al [21]. AppSpeedXray provides a unified view of mobile app performance metrics on your Android smartphone without source codes or compilation. The proposed AppSpeedXray tool gatherers mobile app performance data such as CPU utilization, packet trace, XML log, and execution video file. These data are collected from the binary file of the AUT. Then, the proposed AppSpeedXray framework evaluates the performance metrics. The proposed tool includes the following components: (i) a mobile app crawler to collect Android APK (Android Application Package) files; (ii) a UI (User Interface) to allow the user inputs needed for testing; (iii) a performance analyzer that figures out the performance metrics speed index, mobile app performance metrics, and traffic statistics. The author implemented the performance analyzer component in Python to include a scene detector, snapshot generator, and similarity calculator modules. However, the author did not mention specifically how the performance analyzer works and how it analyzed the collected performance data. Additionally, the author did not discuss details about the test cases used in the proposed framework, which were required to collect the performance data. The author did not mention the performance testing tools used for test case execution.

On the other hand, there are types of performance testing frameworks presented in the literature depending on the source code of the AUT. For example, Zeng et al [22] presented a framework to measure the page load time of Android applications. The presented framework is integrated with Appium [23] and Maven package management tools [24]. Appium is an open-source mobile application testing tool used for executing test cases. The proposed framework mainly focuses on testing whether the page is loaded or not (i.e. elements of a certain page appeared or not). Firstly, the proposed framework records the screen. Then, the author calculates the time spent loading the page frame by frame. The proposed framework implemented the following methods: (i) screen recorder method; (ii) video-to-picture conversion method, (iii) first and last image comparison methods; (iv) final load time calculation method. Then, the recorded video file is analyzed to measure the page load time.

2.2. Performance Testing for Web-based Applications

Recently, web-based applications dominate most of the activities accomplished during the day (e.g., E-Commerce, E-learning, social media). There are different types of web-based applications such as web services, and web applications. Users spend a lot of time using these apps. The behavior of web-based applications should be ensured before releasing them for public use. Performance testing evaluates the quality of web applications. Hence, different types of web-based applications need to thoroughly test their performance under heavy workloads.

Many researchers study the performance testing of web-based applications from different perspectives. Some researchers propose self-adaptive approaches for performance testing, which neither depend on the source code nor performance/system models for the AUT. For instance, self-adaptive reinforcement learning-driven load testing approach called RELOAD was proposed by Moghadam et al [25]. The proposed RELOAD approach generates an effective workload that can be utilized in further relevant testing scenarios (e.g., performance regression testing). the RELOAD learns from the continuous changes in both the AUT and the execution environment. The author depended on the participation of testers to determine the action list that resembled the test scenarios. This means that the test cases generated by the proposed approach are created partially manually. The proposed RELOAD approach focuses on searching for the optimal workload that leads to the best performance. However, the author ignored identifying the workload that leads to the performance bottleneck. JMeter tool [26] is used to generate the required workload that simulates the real users' accesses.

Performance unit testing permits developers to recognize the performance behavior of web applications. Besides, it detects performance issues continuously during the development of web applications [27]. Therefore, the authors adopt a web application performance testing approach at the unit level. The adopted approach supports the widespread usage of an agile method [28] and test-driven development (TDD). For example, a unit-level performance testing approach for web applications testing called PerfMock was introduced by Chatley et al [29]. The author focused on presenting an approach for executing the performance testing at the unit level, continuously in a test-driven manner. The PerfMock prolongs a well-established mocking framework for Java called jMock2 [30], which allows these tests. Mock objects are used to represent the real cooperation of objects under test. Mock objects are configured to imitate specific test scenarios. The introduced performance unit testing approach led to rapid turnaround times. The proposed approach works completely in virtual time (i.e., the performance estimated values are produced without having to stay for the passage of real-time). However, the introduced approach has some weaknesses, such as (i) The author did not mention the method used to increase the workload submitted to AUT. The workload was generated using Apache benchmark [31], instead; (ii) The author was not concerned with the automatic generation of test cases and executable test scripts. Test suit was written using the Junit tool [32], instead; (iii) the author ignored the test results analysis steps.

Test case generation is the core step in the performance testing process [33]. Performance test case generation approaches are introduced in the literature. For example, Zhou et al [34] introduced a workload model for representing and generating synthetic web workloads and test cases for web applications load testing. The author depended on the context-based sequential model for defining user behavior. The proposed approach requires accurate modeling of the session. Thus, the proposed approach requires testers to have good knowledge of session, actions, and request concepts. However, the context-based sequential model is not produced during the development and design of the AUT. As a result, the proposed model requires extra time and cost for modeling.

Many authors integrate automated user interface (UI) testing along with load testing. This is considered a way to allow a complete quality measurement of the user's experience of AUT. For instance, Whiting et al [35] introduced an approach for integrating load testing with automated UI testing. The author aimed to give an overall quality calculation to the user's perception of the AUT. the proposed approach implements a holistic method for evaluating the performance of the AUT. The holistic method used in the proposed approach considers different types of users. This lets testers measure the system quality from the perspective of multiple users. Thus, the proposed approach is based on the user research team that resembled various user groups. The user research team is responsible for picking out the most significant attributes in the AUT, then assigning appropriate weights to these attributes. An effective test oracle is derived and tested according to these weights.

Generally, a test oracle is used to determine whether the AUT has passed or failed for a certain test case. It compares the results of AUT for certain test case inputs and the result that AUT should provide. The proposed approach properly identifies these weights during the collaboration between the technical teams and users. The strengths of the proposed model include the improvement of the software engineering sub-discipline of requirements gathering and increasing the ability of technical and non-technical users to debate the non-functional requirements. On the contrary, the proposed approach relies on the stakeholders to allocate weights, which reflects the significance of the functionality and performance. These weight values may not be empirical, they only reflect a major idea of prioritization functionality and performance.

2.3. Performance Testing for Cloud Applications

Nowadays, cloud applications are widespread, due to their quick response to business needs. Cloud application refers to a software programming model that needs collaboration between cloud-based and local components. This software programming model depends on remote servers to process their logic. Cloud applications require an internet connection to access them through a web browser [36]. They are quickly updated, tested, and deployed. This provides organizations with agility experience and quick time to market [37]. Testing the performance of cloud applications is discussed in the literature. For example:

1. A cloud performance testing approach called PT4Cloud was presented by He et al [38]. The presented PT4Cloud approach targets the performance testing of IaaS clouds. The author aimed to reduce the number of test runs, as well as satisfy the high-test coverage. Test coverage is a metric that measures how much the test cases cover the AUT's code and functionalities. Stop conditions are used in the proposed PT4Cloud to cut off the repeated test runs and obtain accurate performance testing results. This led to a reduction in the performance testing cost. The author used non-parametric statistical approaches (i.e., likelihood theory and the bootstrap method) in the proposed approach. The author mentioned some threats to validity, such as (i) The performance results of the PT4Cloud remain valid unless the execution environment is not changed. This meant changes in the execution environment change the performance results of the proposed approach; (ii) Cloud uncertainty factors including data center location, VM types, and hardware variation, may affect the performance. The author did not consider these cloud factors in the proposed approach.

2. A performance testing approach for an application hosted on a serverless computing environment was introduced by Khatri et al [39]. Serverless computing refers to the function as a service (FaaS), where the service provider processes and executes the code sent by the clients. Virtual machines (VMs) are hired and managed by the service provider. The author conducted the performance test for certain business flows and simulated the expected users' transactions under the peak load (i.e. peak working hours). The author executed the test over many iterations and under high-stress loads to determine the application response time. The workload was generated using the JMeter tool, which creates multiple threads to simulate the virtual users. The proposed approach conducts the following: (i) conduct baseline performance testing to identify the initial issues in the AUT; (ii) generate performance test scripts and test data sets; (iii) conduct load and stress tests; (iv) fix the found issues; (v) repeat the tests until the exit criteria were met. However, the author did not discuss some important issues, such as (i) The test cases generation method; (ii) The test results analysis method that could be applied to determine the performance peaks; (iii) The test environment setup.

Microservices are software applications that consist of loosely coupled software components. They are single-function modules and have well-defined interfaces and operations. They provide an independent deployment facility and can be integrated continuously by the developers [40]. Recently, testing the scalability of microservices deployment by conducting load tests is presented in the literature. For example, an approach for the performance assessment of microservice deployment alternatives was presented by Avritzer et al [41], [42]. The high-level performance models were used in the presented approach. The author used the operational profile data to identify the probability of occurrence of each operational workload situation in production. Each operational situation had a test case. The author used the operational workload situations (e.g., arrival rates or a concurrent number of users) in conducting load testing. The threats to validity mentioned by the author included: (i) The generation of the experiment needs an estimation of the occurrence percentage of each performance test case. This threatens the accuracy of the estimated operational distribution; (ii) The author relied on the automated execution and analysis of the load test cases. This requires uninterrupted improvement using an identified approach and automated deployment.

3. Performance Testing Tools

In this section, various examples of performance testing tools that are excessively used in the market are discussed. These tools target performance testing for both web-based and mobile applications. The mentioned tools are chosen from the list of the top 10 frequently used performance testing tools in 2020 [43]. Table 1 presents a comparison between these selected commercial performance testing tools. Tools included in table 1 are (i) Apache JMeter [26]; (ii) HP Loadrunner [44]; (iii) Microsoft Visual Studio (TFS) [45] (iv) WebLOAD [46]; (v) NeoLoad [47]; (vi) LoadNinja [48]; (vii) LoadUI Pro [49]. These tools are compared according to the following criteria: (i) whether it is open source or needs licenses fees; (2) the type of the interface (i.e. Console, GUI); (iii) type of the AUT that the tool can test it (i.e. mobile application, web-based application); (iv) the required system specification to use the tool; (v) the scripting language the tool support it (e.g. javascript); (vi) the supported protocols.

4. Discussion and Research Gaps

Majority of the researchers who studied performance testing lack important issues. They did not care about providing effective testing approaches to automate the whole performance testing process. Tables 2, and 3 show a summary of the previously mentioned studies from the literature. Table 2 presents the comparison between performance testing approaches for mobile applications testing. Table 3 presents the comparison between performance testing approaches for web-based application testing. Table 4 presents the comparison between performance testing approaches for web-based application testing. The comparison in both tables 2, 3, and 4 is based on the following: (i) the proposed automatic test cases generation approach; (ii) the proposed automatic test execution approach; (iii) support the parallel test execution; (iv) the proposed automatic test result from analysis approach. Additionally, table 5 shows the comparison between the advantages and drawbacks of the top performance testing tools in the market.

Table 1: Comparison Between the Top Performance Testing Tools in The Market

Point of comparison	Apache JMeter	HP Loadrunner	Microsoft Visual Studio (TFS)	WebLOAD	NeoLoad	LoadNinja	LoadUI Pro
Open Source	Yes	No	No	No	No	No	No
Interface	GUI	GUI	GUI	Console	Graphical and Code-based	GUI	GUI
Type of Supported AUT	Web and Mobile apps	Web and Mobile apps	Web and Mobile apps	Web applications	Web and Mobile apps	Web applications	Web services
System Requirements	– Windows – MAC – UNIX	– Windows – LINUX	– Windows 7 – Windows Vista – Windows Server 2008 – Later Windows operating systems.	– Windows – LINUX	– Windows – Linux – Solaris.	None	– Windows – Linux – Mac OS
Scripting language	– Javascript – BeanShell	– Citrix – ANSI C – .Net – Java	– PowerShell – Perl	– Javascript	– YAML-based description format (human readable, implementation agnostic, and domain-specific to load testing) – Jenkis pipeline as Code	Scriptless (Capture and playback script using InstaPlay recorder)	– Java – JavaFX – Groofy
Supported Protocols	– HTTP – HTTPS – XML – SOAP – Java-based protocols – FTP	– Support all protocols	– Support all protocols	– HTTP – HTTPS – XML – Enterprise applications – Network Technology – Server Technology	– HTTP – HTTPS – SOAP – REST – Flex Push – AJAX Push	– HTTP – HTTPS – SAP GUI Web – Web Socket – Java-based protocol – Google Web Toolkit – Oracle form	– HTTP – REST – SOAP – JSON – API Blueprint – JSON Schema – XML Schema

Table 2: Comparison Between Performance Testing Approaches for Mobile Applications Testing

Approaches	Automatic Test Case Generation	Automatic Test Execution	Support Parallel Execution	Automatic Test Result Analysis
Rajan et al [16]	No	Yes	No	No
Kim et al [19]	Yes	Yes	No	No
Kim et al [20]	No	Yes	No	No
Mun et al [21]	No	Yes	No	Yes
Zeng et al [22]	No	Yes	No	No

Table 3: Comparison Between Performance Testing Approaches for Web-based Applications Testing

Approaches	Automatic Test Case Generation	Automatic Test Execution	Support Parallel Execution	Automatic Test Result Analysis
Moghadam et al [25]	No	Yes	No	No
Whiting et al [35]	Yes	No	No	No
Chatley et al [29]	No	Yes	No	No
Zhou et al [34]	Yes	Yes	No	No

Table 4: Comparison Between Performance Testing Approaches for cloud Applications Testing

Approaches	Automatic Test Case Generation	Automatic Test Execution	Support Parallel Execution	Automatic Test Result Analysis
Khatri et al [39]	No	Yes	No	No
Avritzer et al [41], [42]	Yes	Yes	No	No
He et al [38]	No	Yes	No	No

The features and drawbacks of the previously mentioned studies and tools are derived from examining the information included in tables 2, 3, 4, and 5. The concluded strengths are as follows:

- The performance testing for different types of web-based applications (e.g. cloud apps, web services, microservices, web apps) is discussed in the literature.
- Few performance test cases generation techniques are proposed.
- Techniques for the automatic analysis of performance test results are discussed.
- The commercial performance testing tools in the market mainly execute the test cases efficiently.
- The commercial performance testing tools in the market can generate a scalable virtual workload that imitates the real concurrent users' access to the AUT. These tools submit workloads to the AUT, then measure the performance metrics (e.g., response time, throughput) of the AUT.
- LoadNinja and Microsoft Visual Studio (TFS) tools provide analytics and reporting features.
- LoadUI Pro allows parallel load testing.

Table 5: Comparison Between Advantages and Drawbacks of the Top Performance Testing Tools in the Market

Performance Testing Tools	Advantages	Drawbacks
Apache JMeter [26]	– Run different tests simultaneously – Provides accurate test results – Provides data analysis and visualization – Requires low scripting efforts – Available free of charge – Can be installed easily – Allows test recording for native and browser applications	– Can't support Javascript, so can't support AJAX requests – High memory consumption, especially in the GUI mode.
HP Loadrunner [44]	– Allows testing of different types of applications – Allows scalability in the simulated workload – Checks network and server resources. – Supports the automatic client/server performance tracing while testing	– High license cost – License cost is based on the number of virtual users – Sometimes has compatibility issues. – Firewall causes installation issues
Microsoft Visual Studio (TFS) [45]	– Easy to use – Visualizes test reports graphically – Has high scalability in the simulated workload	– High license cost – Support MS Windows operating system only
WebLOAD [46]	– Easy to use – Generates a huge number of virtual users locally and on the cloud – Defines the features of the test easily as DOM-based recording/playback, automatic correlation – Supports many technologies (e.g., web protocols, enterprise applications) – Integrates with open-source software (i.e., Selenium, Jenkins), mobile testing (Perfecto Mobile)	– Long user documentation – Complex setup – High License cost
NeoLoad [47]	– Can be used easily – Allows graphical and code-based approaches – Provides realistic simulation of user behavior – Continuously schedules manage, and shares test resources and results across the organization. – Has a low-cost alternative to the LoadRunner tool – Automates test design, maintenance, and analysis for Agile and DevOps teams	– Limited analysis and reporting capabilities. – Building Test Plans takes a long time
LoadNinja [48]	– lessen the time needed for testing by 60% – Used real browsers instead of load emulators – Support the scriptless load test creation and replay with the InstaPlay recorder – Generates Load with 1000s of real browsers – Real-time debug tests – Real-time virtual user activity management. – Can be hosted on the cloud, no server machine & upkeep required – Has analytics and reporting features	– High license cost
LoadUI Pro [49]	– Provides cloud-based load tests – Allows the parallel load testing – Provides server monitoring mechanism. – Reuses existing functional tests – Has distributed load generators – Drags and drops load tests on distribution agents on the cloud	– High license cost

The concluded weaknesses that reflect the current research gaps are as follows:

- There is no comprehensive testing framework that can automate the whole performance testing process.
- The majority of authors were interested in executing the performance test cases without caring about the rest of the performance testing process (i.e., test case generation, test result analysis, and test report generation).
- Commercial tools in the market neither generate test cases nor analyze test results automatically. They rely on the manually generated test cases that are given as input to be executed.
- Some of the commercial tools rely on capture and replay techniques for test case generation. Although, capture and replay need testers' collaboration and do not ensure the complete coverage of the AUT.
- Some commercial testing tools in the market have high license fees, are difficult to use, operate only on certain operating systems, and need complex setups.

5. Conclusion

Recently, users focused on the quality of software applications. The developed software applications should be tested to validate that they can fulfill their required functionality as well as their performance and reliability requirements. Thus, Performance testing becomes critical. It ensures that the AUT satisfies its functional requirements under multiple workloads of simultaneous user accesses. This evaluates the stability and reliability of the AUT when exposed to various workloads. Performance testing demands efficient testing approaches, experienced testers, adequate test cases, and precise execution tools. Thus, it gains interest from researchers who studied it from different perspectives. This paper provides a brief review of the current state-of-the-art in the performance testing field. The paper introduces the recent research related to performance testing for both mobile and web-based applications. Besides, an analysis of the applied approaches is conducted to pinpoint their strengths and weaknesses. A comparison between the discussed studies is provided. Additionally, issues involved in performance testing are identified from the previous studies in the field of performance testing. These issues still await proper solutions. As a result, new opportunities and challenges appear in the field of performance testing, which challenges researchers to provide solutions.

References

1. Hakeem, Mohd Abdul, Mohammed Abdul Razack Maniyar, and Mohd Khalid Mubashir Uz Zafar. "Performance testing framework for software mobile applications.", 2022.
2. Microsoft Performance Guide, https://docs.microsoft.com/en-us/previous-versions/msp-n-p/bb924375(v=pandp.10)?redirectedfrom=MSDN
3. Arif, Muhammad Moiz, Weiyi Shang, and Emad Shihab. "Empirical study on the discrepancy between performance testing results from virtual and physical environments." Empirical Software Engineering 23, no. 3 (2018): 1490-1518.

4. Mazuera-Rozo, Alejandro, Catia Trubiani, Mario Linares-Vásquez, and Gabriele Bavota. "Investigating types and survivability of performance bugs in mobile apps." Empirical Software Engineering 25, no. 3 (2020): 1644-1686.
5. Hao, Dan, Yinghui Chen, Fan Tang, and Feng Qi. "Distributed agent-based performance testing framework on Web Services." In 2010 IEEE International Conference on Software Engineering and Service Sciences, pp. 90-94. IEEE, 2010.
6. Goetz, Jozef, and M. Ruvacaba. "Mobile application performance for different platforms." In Proceedings of International Research Conference on Engineering and Technology, Kitakyushu, Japan, pp. 62-71. 2016.
7. Moghadam, Mahshid Helali, Mehrdad Saadatmand, Markus Borg, Markus Bohlin, and Björn Lisper. "Learning-Based self-adaptive assurance of timing properties in a real-time embedded system." In 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 77-80. IEEE, 2018.
8. Samir, Amira, Huda Amin, and Nagwa Badr. "A survey on automated user interface testing for mobile applications." International Journal of Intelligent Computing and Information Sciences (2022): 1-11.
9. Moghadam, Mahshid Helali. "Machine learning-assisted performance testing." In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1187-1189. 2019.
10. Moghadam, Mahshid Helali, Mehrdad Saadatmand, Markus Borg, Markus Bohlin, and Björn Lisper. "Machine learning to guide performance testing: An autonomous test framework." In 2019 IEEE international conference on software testing, verification and validation workshops (ICSTW), pp. 164-167. IEEE, 2019.
11. Li, Deguang, Bing Guo, Yan Shen, Junke Li, and Yanhui Huang. "The evolution of open-source mobile applications: An empirical study." Journal of software: Evolution and process 29, no. 7 (2017): e1855.
12. Jilani, Atif Aftab, Muhammad Uzair Khan, Muhammad Zohaib Iqbal, and Muhammad Usman. "An automated search-based test model generation approach for structural testing of model transformations." Journal of Software: Evolution and Process (2022): e2461.
13. Park, Joonseok, Taejun Kang, and Keunhyuk Yeom. "Mobile situation-aware framework for developing smart mobile software." Journal of Software: Evolution and Process 26, no. 12 (2014): 1213-1232.
14. Ali, Amira, Huda Amin Maghawry, and Nagwa Badr. "Automated parallel GUI testing as a service for mobile applications." Journal of Software: Evolution and Process 30, no. 10 (2018): e1963.
15. Az-zahra, Hanifah Muslimah, Nafilah Fauzi, and Agi Putra Kharisma. "Evaluating E-marketplace mobile application based on people at the center of mobile application development (PACMAD) usability model." In 2019 International Conference on Sustainable Information Engineering and Technology (SIET), pp. 72-77. IEEE, 2019.
16. Rajan, VS Sundara, A. Malini, and K. Sundarakantham. "Performance evaluation of online mobile application using Test My App." In 2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies, pp. 1148-1152. IEEE, 2014.

17. Shi, Dexin, Christine DiStefano, Heather L. McDaniel, and Zhehan Jiang. "Examining chi-square test statistics under conditions of large model size and ordinal data." Structural Equation Modeling: A Multidisciplinary Journal 25, no. 6 (2018): 924-945.
18. Mishra, Deepti, and Alok Mishra. "Complex software project development: agile methods adoption." Journal of Software Maintenance and Evolution: Research and Practice 23, no. 8 (2011): 549-564.
19. Kim, Heejin, Byoungju Choi, and W. Eric Wong. "Performance testing of mobile applications at the unit test level." In 2009 Third IEEE International Conference on Secure Software Integration and Reliability Improvement, pp. 171-180. IEEE, 2009.
20. Kim, Heejin, Byoungju Choi, and Seokjin Yoon. "Performance testing based on test-driven development for mobile applications." In Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication, pp. 612-617. 2009.
21. Mun, Hyunsu, and Youngseok Lee. "Appspeedxray: A mobile application performance measurement tool." In Proceedings of the 35th Annual ACM Symposium on Applied Computing, pp. 1010-1012. 2020.
22. Zeng, Wentao, Xiangyu Bai, and Kexin Zhou. "Automation Test Tool for the Page Load Time of Mobile Applications." In 2021 International Conference on Information and Communication Technologies for Disaster Management (ICT-DM), pp. 181-186. IEEE, 2021.
23. Appium, https://appium.io/
24. Maven package management tools, https://maven.apache.org/
25. Moghadam, Mahshid Helali, Golrokh Hamidi, Markus Borg, Mehrdad Saadatmand, Markus Bohlin, Björn Lisper, and Pasqualina Potena. "Performance testing using a smart reinforcement learning-driven test agent." In 2021 IEEE Congress on Evolutionary Computation (CEC), pp. 2385-2394. IEEE, 2021.
26. JMeter tool, https://jmeter.apache.org/
27. Ghafari, Mohammad, Konstantin Rubinov, and Mohammad Mehdi Pourhashem K. "Mining unit test cases to synthesize API usage examples." Journal of software: evolution and process 29, no. 12 (2017): e1841.
28. Poth, Alexander, Susumu Sasabe, Antònia Mas, and Antoni-Lluís Mesquida. "Lean and agile software process improvement in traditional and agile environments." Journal of Software: Evolution and Process 31, no. 1 (2019): e1986.
29. Baltas, Nikos, and Tony Field. "Continuous performance testing in virtual time." In 2012 Ninth International Conference on Quantitative Evaluation of Systems, pp. 13-22. IEEE, 2012.
30. jMock2, http://jmock.org/
31. Apache benchmark, https://ubiq.co/tech-blog/how-to-use-apache-bench-for-load-testing/
32. Junit tool, https://junit.org/junit5/
33. Singh, Rajvir, Rajesh Bhatia, and Anita Singhrova. "Demand based test case generation for object oriented system." IET Software 13, no. 5 (2019): 403-413.
34. Zhou, Junzan, Bo Zhou, and Shanping Li. "Ltf: a model-based load testing framework for web applications." In 2014 14th International Conference on Quality Software, pp. 154-163. IEEE, 2014.
35. Whiting, Erik. "Granular Modeling of User Experience in Load Testing with Automated UI Tests." In 2021 International Conference on Data and Software Engineering (ICoDSE), pp. 1-5. IEEE, 2021.

36. Pahl, Claus, Pooyan Jamshidi, and Danny Weyns. "Cloud architecture continuity: Change models and change rules for sustainable cloud software architectures." Journal of Software: Evolution and Process 29, no. 2 (2017): e1849.
37. Akbar, Muhammad Azeem, Arif Ali Khan, Sajjad Mahmood, Ahmed Alsanad, and Abdu Gumaei. "A robust framework for cloud-based software development outsourcing factors using analytical hierarchy process." Journal of Software: Evolution and Process 33, no. 2 (2021): e2275.
38. He, Sen, Glenna Manns, John Saunders, Wei Wang, Lori Pollock, and Mary Lou Soffa. "A statistics-based performance testing methodology for cloud applications." In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 188-199. 2019.
39. Khatri, Deepak, Sunil Kumar Khatri, and Deepti Mishra. "Performace Testing Approach for Enterprise Application comprising Serverless Component." In 2021 International Conference on Intelligent Technologies (CONIT), pp. 1-4. IEEE, 2021.
40. Balalaie, Armin, Abbas Heydarnoori, Pooyan Jamshidi, Damian A. Tamburri, and Theo Lynn. "Microservices migration patterns." Software: Practice and Experience 48, no. 11 (2018): 2019-2042.
41. Avritzer, Alberto, Daniel Menasché, Vilc Rufino, Barbara Russo, Andrea Janes, Vincenzo Ferme, André van Hoorn, and Henning Schulz. "PPTAM: production and performance testing based application monitoring." In Companion of the 2019 ACM/SPEC International Conference on Performance Engineering, pp. 39-40. 2019.
42. Avritzer, Alberto, Vincenzo Ferme, Andrea Janes, Barbara Russo, Henning Schulz, and André van Hoorn. "A quantitative approach for the assessment of microservice architecture deployment alternatives by automated performance testing." In European Conference on Software Architecture, pp. 159-174. Springer, Cham, 2018.
43. https://www.edureka.co/blog/performance-testing-tools/
44. HP Loadrunner, https://www.microfocus.com/en-us/products/loadrunner-professional/overview
45. Microsoft Visual Studio (TFS), https://docs.microsoft.com/en-us/azure/devops/server/tfs-is-now-azure-devops-server?view=azure-devops
46. WebLOAD, https://www.radview.com/webload-download/
47. NeoLoad,https://www.tricentis.com/software-testing-tool-trial-demo/neoload trial/?utm_source=referral&utm_medium=redirect&utm_campaign=neotys
48. LoadNinja, https://loadninja.com/
49. LoadUI Pro, https://download.cnet.com/LoadUI-64-bit/3000-2383_4-75915516.html