Mobile Apps Performance Testing as a Service for Parallel Test Execution and Automatic Test Result Analysis

Amira Ali, Huda Amin Maghawry, Nagwa Badr
Information Systems Department-Faculty of Computer and Information Science, Ain Shams University Cairo, Egypt

***Abstract*—Now-a-days, numerous mobile apps are developed daily that influence the lives of people worldwide. Mobile apps are implemented within a limited time and budget. This is to keep up with the rapid business growth and to gain a competitive advantage in the market. Performance testing is a crucial activity that evaluates the behavior of the application under test (AUT) under various workloads. Performance testing in the domain of mobile app development is still a manual and time-consuming activity. As a negative consequence, performance testing is ignored during the development of many mobile apps. Thus, mobile apps may suffer from weak performance that badly affects the user experience. Therefore, cloud technology is introduced as a solution that emerges in the domain of software testing. Based on this technology, software testing is provided as a service (TaaS) that leverages cloud-based resources. This overcomes the testing issues and achieves high test quality. In this paper, a cloud-based testing as a service architecture is proposed for performance testing of mobile apps. The proposed performance testing as a service (P-TaaS) adopts efficient approaches for automating the entire process. Efficient approaches for test case generation, parallel test execution, and test results analysis are introduced. The proposed test case generation approach applies model-based testing (MBT) technique that generates test cases automatically from the AUT's specification models and artifacts. The proposed P-TaaS lessens the testing time and satisfies the fast time-to-release constraint of mobile apps. Additionally, the proposed P-TaaS maximizes resource utilization, and allows continuous resource monitoring.**

Keywords*—*Performance testing; mobile apps testing; mobile apps performance testing; automated testing; cloud computing; TaaS; model-based testing

I. Introduction

The evolution of wireless technology and the development of an immense number of smartphones led to the prosperity of the mobile app development industry [1]. An enormous number of mobile apps are developed and uploaded to different app stores (e.g., Google Play Store) daily. Thus, mobile app testing becomes an urgent matter that must be performed to ensure the application under test (AUT)'s functionality, quality, and reliability before it is released for public use. However, mobile apps usually have a short development life cycle [2]. Thus, many mobile apps are not rigorously tested.

The performance of mobile apps is considered an important concern to users. The prosaic performance of mobile apps roughly affects the user experience [3]. Therefore, mobile app performance testing is considered an indispensable activity. Mobile app performance testing refers to the determination of the AUT behavior under various workloads of concurrent users [4]. This ensures the AUT's responsivity to the concurrent users' instructions as well as discovering AUT vulnerability under various workloads.

At present, cloud computing with its virtualization technologies has become a critical orientation in the information technology industry [5]. The integration of cloud technology with the software engineering domain led to an evolution in the field of software testing. Therefore, the expression of cloud testing appeared. The cloud-based testing frameworks provide an on-demand TaaS for testing any type of software app including mobile app testing [6]. TaaS is defined as a service model that automatically carries out the entire testing process in a cloud-based environment. Then, it submits the test results to the end user. Consequently, the TaaS architecture can be used to provide performance testing as a service (P-TaaS) using cloud-based resources.

However, the challenges found in the literature [7-10] related to performance testing and P-TaaS include the following:

- The majority of researchers focus on discussing functional testing in the context of cloud testing. However, performance testing was relatively rare. Thus, few researchers introduced a comprehensive architecture for automating the entire performance testing process and utilizing cloud-based resources.
- Most of the existing performance testing research focuses on performance test case generation and execution. Few works present systematic approaches to automatically analyze the performance test results.

Therefore, the main contributions of this paper include proposing the following:

- A performance testing as a service (P-TaaS) architecture that automates the whole performance testing process for hybrid mobile apps. Hybrid mobile apps are web apps that are downloaded from mobile app stores and need an internet connection to operate.
- An automated approach for performance test case generation. Thus, no need for skillful testers to design test cases that resemble real users' scenarios when using AUT under various workloads.

- An approach for simultaneous test case execution on multiple virtual nodes. This reduces the time required for the test execution process.
- An automated approach for performance test results analysis that can detect the performance bottleneck in the AUT.

The rest of the paper is organized as follows: Section II presents brief background information. Section III surveys the most relevant work related to performance testing and P-TaaS specifically. Section IV presents the proposed mobile app P-TaaS architecture and a detailed explanation of the functionality of each module. Section V shows the experimental results. Section VI mentions the limitations of the proposed P-TaaS and the differences between the proposed P-TaaS and other relevant tools. The paper is concluded in Section VII. Finally, the future work is presented in Section VIII.

II. Background

This section provides brief background information about concepts that are utilized in the proposed mobile app P-TaaS architecture (i.e. model-based testing and the OCL-based UML diagrams).

The Model-based Testing (MBT) [11] is known as an automatic testing technique that generates performance test scenarios from the AUT specifications that are represented by software models. Unified modeling language (UML) [12] is one of the most widely used methods to model software apps. MBT automatically generates test cases from the software models that represent the behavior and the requirements of the AUT. Then, the software models are converted into test models (e.g., Finite State Machine (FSM)). FSM graph is defined as a set of AUT states, where the inputs trigger each transition and convert AUT from one state to another. FSM graph is traversed to obtain paths. Each path represents a user behavior when using AUT. Thus, FSM graph allows the automation of test case generation.

OCL [13] stands for object constraint language. Generally, OCL is used as a formal language for adding user-defined constraints on the UML diagrams. The OCL as a formal language includes three types of constraints: (i) invariant, (ii) precondition, and (iii) post conditions. The invariant constraint added to any object means that this constraint must be true for the entire lifetime of that object. The precondition constraint added to an operation shall be true before the operation execution. The post condition added to an operation shall be true just after the operation execution.

III. Related Work

Many researchers were concerned with studying mobile app performance testing in a cloud-based environment from different perspectives. The benefits and challenges of mobile app performance testing using cloud-based resources are widely discussed in the literature [5], [6]. Ali et al. [14] reviewed the most recent studies and research gaps related to the performance testing using cloud-based resources. Section III A discusses the most recent mobile apps performance testing frameworks, as well as some of the performance testing tools and services widely used in the market. Section III B introduces relevant studies on the adoption of model-based testing (MBT) techniques in mobile app testing. Section III C discusses the performance test results analysis and interpretation techniques. Finally, Section III D introduces recent studies related to the resource utilization and scheduling approaches adopted in the TaaS domain.

A. P-TaaS Frameworks and Widely Used Performance Services in the Market

Mobile apps performance testing frameworks based on the cloud-based environment were presented in the literature. For instance, Prathibhan et al. [15] presented Android Testing as a Service framework known as (ATaaS). The framework depends on the emulators and Android Application Package (APK) file of the AUT as input to execute test cases. Performance test cases are generated manually by the testers, then test cases are recorded to be executed several times under various workloads. The author did not ensure the efficient utilization of cloud-based resource. The presented ATaaS framework focused only on test case execution and ignored the rest of the performance testing activities.

There are cloud-based performance testing tools and services that are adopted for both small and large businesses with various pricing structures. SOASTA CloudTest [16], LoadStorm [17], and Xamarin Test Cloud [18] were selected from the list of the top 10 extremely used cloud-based performance testing tools in 2020 [19]. SOASTA CloudTest depends on manual test case generation and test results analysis. LoadStorm generates a performance test results analytics report that represents the performance metrics, such as response time and error rate. LoadStorm does not support automatic test case generation. It requires testers to record test scenarios manually. Xamarin Test Cloud allows cross-platform mobile app testing. It supports multiple tenets to execute tests over thousands of devices. Xamarin Test Cloud has limited access to open-source libraries.

B. Model-Based Testing Techniques for Mobile Apps

Model-based testing (MBT) techniques are widely adopted in mobile app testing. Researchers introduced the MBT approach for mobile app test case generation. For instance, Usman et al. [12] adopted UML class diagrams and state machine models in their proposed performance testing approach. The proposed approach generates abstract test cases automatically using UML class diagrams, state machine models, and OCL constraints. The proposed approach was experimented on two different Android apps. The results showed that the proposed approach successfully estimates the performance of apps under test. However, the author only focused on conducting the performance testing of the code related to the business logic code. Additionally, he ignored testing the performance of the GUI.

C. Performance Test Results Analysis and Interpretation

Techniques for performance test results analysis and interpretation were studied in the literature in the field of P-TaaS. Researchers presented frameworks for the automatic analysis of performance test results. Liu et al. [20] proposed a framework for analyzing test results and detecting performance bottlenecks. The proposed framework increased the workload iteratively and monitored the AUT performance metrics simultaneously. The proposed framework was based on cloud infrastructure. However, the author did not discuss issues related to conducting the performance test efficiently using cloud resources. Additionally, the author ignored mentioning the used approaches and tools in test case generation, test execution, and workload generation.

D. Scheduling and Resource Utlization Techniques

There are many researchers presented scheduling approaches to enhance resource utilization, especially in the TaaS field. For instance, the fuzzy sets theory was applied to schedule test cases in the TaaS platform by Lampe et al. [21]. The author applied the fuzzy sets theory as a solution to address the difficulty of predicting the duration of test case execution in advance. This is called uncertainty task scheduling issue. Two algorithms were proposed by the author to handle the uncertainty task scheduling issue. The proposed algorithms are based on simulated annealing (SA). The author used the Q-recent estimate for each test case to estimate the average durations from the history of executions of a given test case.

The metaheuristic methodologies were proposed by Rudy [22]. The author applied metaheuristic methodologies to schedule the parallel test case execution in the context of TaaS. Genetic algorithm (GA) [23] is an example of the metaheuristics methodologies that was used by the author. The presented metaheuristic methodologies assume that the test case execution time is unknown before the test execution. The proposed metaheuristic methodologies have the following drawbacks (i) it needs a long computation time; (ii) the metaheuristics can operate on a limited number of test cases at once (i.e., 300 for SA and 100 for TS and GA).These limitations badly influence the quality of the solution.

Therefore, it observed from analyzing the previous related work that there is no comprehensive framework that conducts the whole performance testing process automatically and leverages the cloud-based environment efficiently.

IV. The Proposed Mobile App P-TaaS Architecture

This paper proposes a P-TaaS architecture for mobile apps. Tester submits a request to the proposed P-TaaS where the entire performance testing process will be automatically processed. The overall architecture of the proposed mobile apps P-TaaS is shown in Fig. 1. The five main layers of the proposed architecture are as follows (1) user interface layer; (2) performance testing layer, (3) service management layer, (4) infrastructure as a service (IaaS) layer, and (5) data repository.

1) User interface layer: The user interface layer is the top web-based layer of the proposed architecture. It is where the testers can interact with the proposed P-TaaS architecture. The tester submits the test input files to accomplish the performance test. Then, the tester receives the test results report through it.

2) Performance testing layer: The performance testing layer is responsible for automatically accomplishing all the performance test activities including test case generation, parallel test case execution, test result analysis and interpretation, and finally the test report generating. The proposed approaches that are applied to each of these activities are discussed in the consequent subsections IV A.

3) Service management layer: The service management layer is concerned with managing, monitoring, and scheduling the test execution tasks among the available resources. The main modules of the service management layer are the scheduler, runtime monitor, and resource allocator. A detailed explanation of each module will be introduced in Section IV B.

4) IaaS layer: The infrastructure as a service (IaaS) layer includes virtual machines (VMs) where the testing process physically occurs. The virtualization technology is applied to provide all needed resources in the proposed P-TaaS architecture.

5) Data repository: The data repository is where the proposed mobile app P-TaaS architecture stores all generated data during its operation. These data include the following: (1) The test cases generated from the test case generation module, (2) The performance measurements generated from the test case execution module, (3) Information produced from the test results analysis module, (4) The test reports obtained from test report generation module, and (5) VMs status (i.e., on, off, or idle) detected by the monitor module.

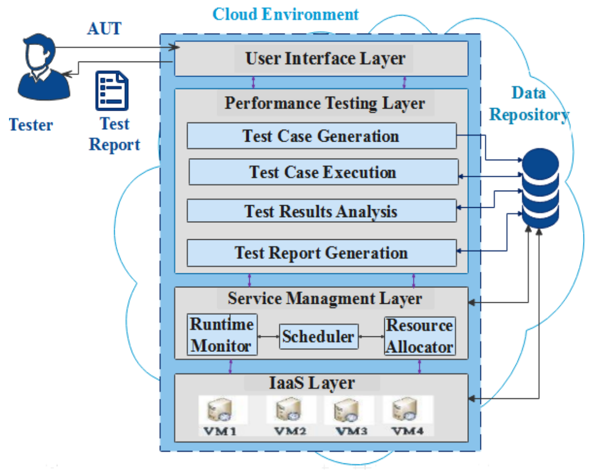

Fig. 1. Mobile apps P-TaaS architecture.

A. The Performance Testing Layer

The performance testing layer is the core layer of the proposed mobile app P-TaaS architecture. In this section, the proposed approaches employed in each module of this layer will be briefly explained.

1) The test case generation module: The test case generation module is responsible for the automatic generation of AUT's performance test cases. Generating test cases is an essential activity of the entire testing process [12]. Performance testing requires a set of appropriate test cases that can evaluate the responsiveness of the AUT under various workloads of concurrent users' accesses [3]. Generally, a performance test case composes of a consecutive set of actions exerted on the mobile app's GUI widgets. Performance test cases should resemble real user scenarios when using AUT. Besides, they should achieve full coverage of all the AUT's GUI actions and activities [4]. Automating the process of mobile app performance test case generation lowers the cost and raises the efficiency of testing. Additionally, it can improve the accuracy of test results.

In this paper, the proposed test case generation approach is based on the MBT methodology. The proposed MBT methodology depends on the OCL activity diagrams. Activity diagrams are used to model AUT's workflows in terms of stepwise activities and actions. The OCL defines the performance requirements formally. The OCL-based activity diagrams become a good candidate for modeling user behavior which allows automated mobile app performance testing. OCL-based activity diagrams are used to represent the AUT flow of actions as well as the performance requirements associated with every GUI action. The proposed approach targets interactive hybrid mobile apps. Interactive hybrid mobile apps [24] perform the requests of users who interact with the apps through the GUI through an internet connection.

The proposed test case generation approach depends on representing each functionality in the AUT by a separate OCL-based activity diagram. Each AUT's functionality will be modeled as an activity diagram. Besides, each activity diagram will have OCL constraints associated with each action in the activity diagram. Thus, the prerequisite inputs of the proposed mobile apps performance test approach are acquired from the analysis and design team. These inputs include: (a) an activity diagram for each functionality in AUT; (b) OCL constraints added to each activity diagram.

For each OCL-based activity diagram, which are submitted by the tester through the user interface layer of the proposed P-TaaS architecture, the steps of the proposed test case generation approach go as follows:

- OCL Based Activity Diagram is parsed into an XM Based Activity Diagram. This is considered the primary step to automate the test case generation process.
- All details included in each XML Based Activity Diagram are extracted and inserted into a Predicate Table. Each XML Based Activity Diagram is converted to its corresponding Predicate Table. Each element included in the XM Based Activity Diagram is shown as a row in the Predicate Table.
- The following information about each element of its corresponding XML Based Activity Diagram are stored in the Predicate Table:
    - Element name.
    - Element type (i.e., action, decision, initial, and final nodes).
    - Performance constraints on this element (i.e. represented with OCL constraints)
    - List of all edges to this element.
    - List of all edges out from this element.

Edges are used to represent the dependency between elements. The dependency between elements is proved by the existence of an edge out from an element which is the same as the edge to the current element.

- A Finite State Machine (FSM) Graph [25] is automatically created from each Predicate Table.
- The Depth First Search (DFS) [26] algorithm is applied to the FSM Graph to find all available Independent Path. The Independent Path refers to any path from the start node to the terminal node of the FSM and has at least one new edge that has not been traversed before.
- The Longest Common Subsequence (LCS) [27] algorithm is applied to filter the previously generated independent paths. LCS is an algorithm that eliminates test paths that are included as a sub-path in another path, to obtain basic paths. Applying the LCS algorithm decreases the number of test paths and ensures high test coverage. Additionally, it guarantees the existence of each GUI component at least once in the obtained basic paths. Thus, the LCS algorithm is used as a test case selection and reduction method.
- Finally, the obtained basic paths as an output of the proposed test case generation approach are considered abstract test cases. Each abstract test case consists of a set of GUI actions that resemble an actual user scenario, as well as an analogy of an entire path inside the AUT.

2) The test case execution module: Test case execution module is responsible for the automatic execution of mobile app performance test cases in a cloud-based environment. In the proposed mobile app P-TaaS architecture, test cases are executed simultaneously on multiple VMs. This leads to a reduction in the overall testing time, cost, and effort. The input to the test case execution module includes: (i) abstract test cases; (ii) Android Application Package (APK) file of the AUT. The tester submits these two inputs to the proposed P-TaaS through the user interface layer. APK file is a file format used by the Android operating system [28]. It assists in the easy distribution and installation of Android apps. The proposed P-TaaS architecture is based on using the APK file of the AUT. Hence, there is no necessity for the AUT's source code.

The automatically generated abstract test cases are converted to test scripts by the tester. The tester records these abstract test cases using the capture and reply test methodology [29]. Then, these test scripts will be executed to measure the AUT performance characteristics and responsivity toward the heavy load of concurrent users' access. During the test case execution, performance response time and error rate are measured. Response time is defined as the time taken by the AUT to respond to a certain action. Thus, response time is the total time between sending the request and receiving the response [3]. The error rate is defined as the ratio of failed requests with respect to the total number of requests [3].

Generally, there are several mobile app performance testing tools for test case execution. Apache JMeter [30] is an open-source tool that is first used to test the performance of web applications. Eventually, Apache JMeter expanded to allow mobile app performance tests as well. Proxy is used by the JMeter tool to record requests to mobile apps. The proxy can be configured on a mobile device. Then, the request will be captured by JMeter. Additionally, the JMeter tool allows the scalability during the performance testing process by providing any number of the virtual workload of concurrent users. Therefore, the JMeter tool is selected in the proposed P-TaaS to simulate different workloads and test the AUT under heavy workloads to detect the performance bottlenecks in the AUT.

The entire process of test case execution is shown in Fig. 2. The steps of the test case execution process are as follows:

a) Firstly, the test case execution module retrieves the automatically generated abstract test cases from the data repository.

b) The tester records the steps included in every abstract test case using the capture and reply test methodology.

c) Test scripts are recorded by the JMeter recording proxy. The recording proxy of the JMeter can record the HTTP requests executed by the tester on mobile devices. This is used for test script generation.

d) Test scripts are stored in the data repository.

e) The test case execution module submits the captured test scripts and APK of the AUT to the assigned VMs to execute test scripts in parallel on multiple VMs.

f) APK file will be installed on the assigned VMs.

g) Then, each test script is executed with a different workload to test the responsivity of the AUT under various workloads and reveal the AUT bottlenecks.

h) Finally, test results are stored in the data repository for further analysis and interpretation.

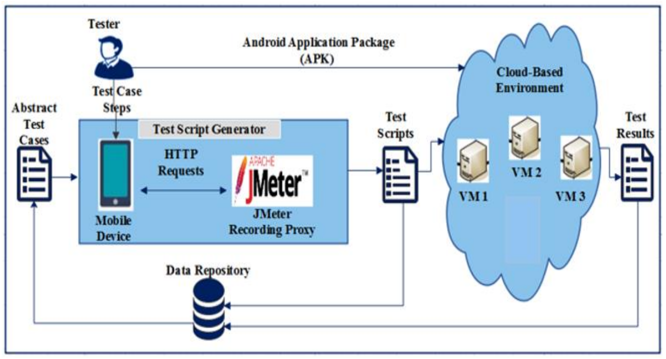

Fig. 2. Test case execution.

3) Test result analysis module: The test result analysis module is responsible for the automatic analysis of test execution results. In this paper, an approach for performance test results analysis is proposed. The proposed approach collects the performance metrics values (i.e. response time and error rate) under different workloads. Then, it interprets these data to determine the performance critical turning point and the heavy workload value. Performance critical turning point refers to a point during the AUT execution under various workloads, where the error rate suddenly increases and the response time of the AUT increases exponentially [20]. This indicates that the performance of the AUT descends sharply and the AUT may crash. When the AUT reaches the performance critical turning point, this implies that a performance bottleneck occurs under this workload. Heavy workload means the value of workload is more than or equal to the workload where the performance critical turning point occurs.

The required input to the proposed performance test results analysis approach includes the minimum and the maximum number of concurrent users defined by the tester. The tester defines the minimum and maximum number of concurrent users that simulate the expected minimum and maximum workloads exposed to the AUT during its production. The test shall consider the expected workloads during daily operations, peak hours on the AUT, and the most popular days the AUT will be used. The performance metrics are measured starting from the minimum number of concurrent users' accesses to the AUT, until the maximum number of users' accesses. During test execution, the response time and error rate are measured at each workload from the minimum to the maximum workload values. Additionally, the Error_Threshold refers to the maximum error rate value that is accepted by the tester. The AUT is considered in an unstable state when its error rate value reaches the Error_Threshold during the test execution. The Error_Threshold is defined by the tester during the proposed performance test results analysis approach.

Fig. 3 shows the pseudocode of the proposed performance test results analysis approach. The steps of the proposed performance the proposed performance test results analysis approach go as follows:

a) Firstly, loop on the number of concurrent users starting from the minimum number to the maximum number. For each iteration, the workload value will be increased by 50.

b) For every workload value:

- Performance metrics (i.e., response time and error rate) are measured.
- The measured performance metrics values are added to Response_Time_Measurements, and Error_Rate lists.
- H_Respone_Time refers to the length of the perpendicular line on the line connecting the first and last value of response time at the minimum and the maximum number of concurrent users, respectively. Fig. 4 shows H_Respone_Time at a certain workload value [20]. Equation 1 shows how to calculate H_Respone_Time at a certain workload x.

$$\text{H\_Response\_Time}[x] = A\_RT[x]Sin(Cos(A\_RT[x]^2 + C\_RT^2 + B\_RT[x]^2)/2 * A\_RT[x] * C\_RT)$$

(1)

- *A_RT* refers to the length of the line between the following two points: the coordinates of the first point are (minimum workload, value of the measured response time at minimum workload) and the coordinates of the second point are (certain workload x, value of the measured response time at this workload x).
- *B_RT* refers to the length of the line between the following two points: the coordinates of the first point are (maximum workload, value of the measured response time at maximum workload) and the coordinates of the second point are (Certain workload x, the value of the measured response time at this workload x).
- *C_RT* refers to the length of the line between the following two points: the coordinates of the first point are (minimum workload, value of the measured response time at minimum workload) and the coordinates of the second point are (maximum workload, and value of the measured response time at maximum workload).
- Sin and Cos refer to the trigonometrical sine rule and cosine rule, respectively.
- Equation 1 is used to calculate the length of the perpendicular line H_Response_Time.
- The perpendicular line H_Response_Time which corresponds to the workload value x. The performance critical turning point [20] is the point with lonest perpendicular line.

c) Secondly, the error rate value that exceeds the Error_Threshold value is determined.

d) Thirdly, the workload value where the error rate exceeds the defined threshold value is captured. The captured workload value is called the Unstable_Workload. It is where AUT starts to crash, and its behavior becomes unstable.

e) Fourthly, RT_Critical_Turning_Point is determined. It refers to the maximum value of response time in the H_Response_Time list. It has a workload value less than the captured Unstable_Workload.

*f) RT*_Critical_Turning_Workload is captured. It refers to the workload value at which the RT_Critical_Turning_Point occurs.

g) Finally, the value of the RT_Critical_Turning_Workload is obtained. This workload value represents the Performance_Critical_Turning_Workload.

*h) Heavy*_Load refers to any workload value more than the value of the Performance_Critical_Turning_Workload.

```
Input: Min_of_Concurrent_Users, Max_of_Concurrent_Users,
Error_Threshold
Output: Performance_Critical_Turning_Workload, Heavy_Load,
Response_Time_Measurements, Error_Rate _Measurements

Start

For (x= Min_of_Concurrent_Users; x < Max_of_Concurrent_Users; x+=50)
  // Loop to execute the test cases on different workloads
 {
   Response_Time_Measurements [x] = Get_Response_Time (x)
 // calculate the response time at workload x

   Error_Rate_ Measurements [x] = Get_Error_Rate (x)
 // calculate the error rate at workload x

   $\mathrm{H\_Response\_Time}[x] = \mathrm{A\_RT}[x]\mathrm{Sin}(\mathrm{Cos}(\mathrm{A\_RT}[x]^2 + \mathrm{C\_RT}^2 + \mathrm{B\_RT}[x]^2)/2 * \mathrm{A\_RT}[x] * \mathrm{C}\_RT$
  }
For (i=0; I < Error_Rate_ Measuremenst.Lenght; i++)
 // Loop to determine the unstable workload where the AUT starts to crash
 {
 If (Error_Rate_ Measurements.Lenght >= Error_Threshold)
          {
          Unstable_Workload = i;
          Break;
          }
 }
For (i= Min_of_Concurrent_Users; i< Unstable_Workload; i++)
 {
 RT_Critical_Turning_Point = Max (H_Response_Time [i]);
 RT_Critical_Turning_Workload = i;
 }
 Performance_Critical_Turning_Workload =
 RT_Critical_Turning_Workload;

 Heavy_Load is more than or equal to the RT_Critical_Turning_Workload

End
```

Fig. 3. Performance test results analysis approach.

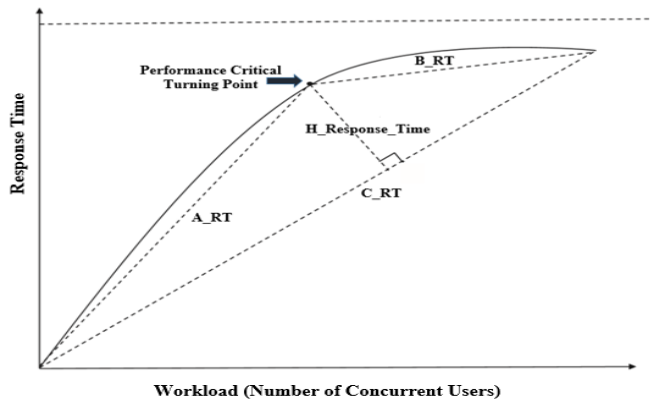
Response Time
Performance Critical Turning Point
B_RT
H_Response_Time
C_RT
A_RT
Workload (Number of Concurrent Users)

Fig. 4. Performance critical turning point.

4) Test report generation module: The final activity of the performance testing process is to generate the test report and submit it to the tester through the user interface. The test report gathers test results collected from multiple virtual nodes. The test report contains: (i) the automatically generated test cases; (ii) the measured performance metrics (i.e., response time measurements and error rate measurements); (iii) the information produced from test results analysis and interpretation (i.e., performance critical turning point, and heavy workload value). This can be valuable for testers to determine the performance deviations and bottlenecks in the AUT. Consequently, testers utilize this information to resolve performance issues within the AUT.

B. Service Management Layer

The service management layer is responsible for managing and optimizing the cloud-based resources and infrastructure. The aim of this layer includes the following: applying a suitable scheduling approach for simultaneous test case execution; reducing the overall testing time; handling uncertainty issues in scheduling the test case execution; improving the resource utilization; monitoring the runtime status of resources.

The service management layer includes three modules: (1) runtime monitor, (2) scheduler, and (3) resource allocator. An insightful explanation of the role of each module will be mentioned in this section.

1) Runtime monitor module: The runtime monitor module guarantees a high-reliability level. The monitor module is implemented as a local service for tracking the runtime status of all virtual nodes. Then, it stores a list of the available VMs. Additionally, it determines the state of each virtual node (i.e., idle, busy, and fail). The list of available VMs is sent to the scheduler and resource allocator modules, in order to assign tasks to available VMs.

2) Scheduler module: The scheduler module is developed as a local service for sorting and prioritizing test tasks submitted to the proposed P-TaaS architecture [31]. This module aims to achieve efficient utilization of the resources.

Test task duration is hard to predicate before the actual test task execution. Additionally, test task duration varies from one task to another. This is considered an uncertainty scheduling issue [22]. The proposed approach considers the uncertainty scheduling issue. The test task scheduling approach depends on task waiting time and task deadline. Task waiting time is the amount of time between task submission and the current time. The task deadline is the time defined by the tester when the task is submitted to the user interface of the proposed P-TaaS architecture. The task deadline is the predefined time at which the task must be finished and delivered before it.

The proposed scheduler approach goes as follows: firstly, the scheduler module sorts tasks in ascending order according to the task deadline. This means that tasks with an earlier deadline will be executed earlier. Secondly, tasks with the same deadline will be arranged according to their waiting time. For tasks with an equal deadline, the task with a higher waiting time will be executed earlier.

3) Resource allocator module: The resource allocator module aims to achieve a high level of resource utilization and load balance. The resource allocator module receives an ordered list of test tasks from the scheduler module. Additionally, the list of available virtual nodes is sent to the resource allocator from the monitor module. Consequently, the resource allocator module allocates test tasks to certain virtual nodes, in such a way that guarantees the load balance between virtual nodes.

The proposed resource allocation approach goes as follows:

- Each virtual node in the proposed P-TaaS has a waiting queue. It includes a list of tasks assigned to it.
- The length of the waiting queue of each virtual node is calculated.
- Virtual nodes are sorted in descending order according to the calculated waiting queue length.
- The virtual nodes with small waiting queue lengths will be assigned to high-priority test tasks.

V. Experimental Results

Experiments are carried out to assess the applicability of the proposed P-TaaS. In the experiments, three virtual machines are used to simulate a cloud-based environment and support the simultaneous execution of test cases. The three virtual machines are created using VMware workstation [32]. The VMware workstation allows the creation of multiple virtual machines on the same physical machine. The experiments are carried out on a machine with the following specs: processor Intel Core i5, memory 8 GB, and Windows 10 operating system. Thus, three virtual machines are created to suit the specs of this physical machine and allow the simulation of simultaneous test execution of the cloud-based environment. The JMeter performance testing tool [30] runs on virtual machines. MS SQL is used to develop data repositories that store the automatically generated test cases and test results (i.e., performance measurements, test results interpretation, and analysis information). The OCL-based activity diagrams are drawn using the Enterprise Architect tool [33]. In addition, the Enterprise Architect tool parses the drawn OCL-based activity diagrams to XML-based activity diagrams with their corresponding OCL constraints. The experimental environment is the same for executing each task using VMs with equivalent specifications.

Nowadays, Android dominates almost 88% of the mobile device market worldwide [34]. Therefore, our experiments depend on using Android apps as AUTs. The APK files of the android AUTs are installed on the virtual node before the test case execution starts. The objective of the proposed P-TaaS architectures is testing the performance of hybrid interactive mobile apps. Therefore, two hybrid mobile apps were chosen for experiments. They are the SpeedTest app [35] and GoodReads app [36]. GoodReads app looks like an online bookstore, where users can read, browse, recommend, and review books. It is a widely used app worldwide. It is the first ranked app in the category of (Science and Education, Libraries and Museums) in the United States. The total number of visits to the GoodReads apps reached 119.7M [37].

SpeedTest app is an Android app that measures the speed of the internet. It is the most used app for measuring the upload/download speed of the internet. It is used by more than 45 billion times unparalleled. Therefore, GoodReads and SpeedTest apps are selected as AUTs in the experiments.

A. *Experimental Results of the Proposed Performance Test Case Generation Approach*

The purpose of this experiment was to evaluate the proposed mobile app performance test case generation approach. The proposed test case generation approach is assessed in terms of action coverage, activity coverage, and performance requirements coverage. The action coverage refers to the percentage of actions included in the automatically generated abstract test cases to all actions included in the activity diagrams. Similarly, the activity coverage. Performance requirements coverage refers to the ratio between the performance requirements covered by test cases and all performance requirements in the activity diagrams. The proposed test case generation approach is based on the black box MBT methodology. The experimental results of applying the proposed test case generation approach on the SpeedTest app and GoodReads app are presented in Tables I and II, respectively.

As shown in Table I, the number of automatically generated abstract test cases was 17. They were used to test seven major functionalities of the SpeedTest app. The total time required to automatically generate test cases using the proposed approach was 3.055 seconds. It is observed from Table II that the number of automatically generated abstract test cases was 28. They were used to test 13 major functionalities of the GoodReads app. The total time required to automatically generate test cases using the proposed approach was 6.325 seconds. Additionally, it is observed from Tables I and II that the generated test cases for each functionality cover 100 % of both activities and actions exerted during each functionality. Therefore, it is considered faster than the manual approaches, which need a long time to design, write, and review the coverage of the test cases.

TABLE I. EXPERIMENTAL RESULTS OF THE PROPOSED PERFORMANCE TEST CASE GENERATION FOR SPEEDTEST APP

Functionality	**Number of GUI Activities**	**Number of GUI Actions**	**Number of Test Cases**	**Time Spend to Generate Test Cases**	**% of Covered GUI Activities**	**% of Covered GUI Actions**
1. Measure download/upload speed	2	2	1	0.211 sec	100 %	100 %
2. Show or delete previous results	3	4	3	0.617 sec	100 %	100 %
3. Adjust app settings	3	3	3	0.614 sec	100 %	100 %
4. SpeedTest support	5	6	3	0.540 sec	100 %	100 %
5. Privacy and terms	8	8	5	0.651 sec	100 %	100 %
6. Test the speed of the video	4	4	1	0.211 sec	100 %	100 %
7. Generate map data	2	2	1	0.211 sec	100 %	100 %
Total	27	29	17	3.055 sec	100 %	100 %

TABLE II. EXPERIMENTAL RESULTS OF THE PROPOSED PERFORMANCE TEST CASE GENERATION FOR GOODREADS APP

Functionality	**Number of GUI Activities**	**Number of GUI Actions**	**Number of Test Cases**	**Time Spend to Generate Test Cases**	**% of Covered GUI Activities**	**% of Covered GUI Actions**
1. Search by book name or author name	2	2	1	0.214 sec	100 %	100 %
2. Search by book genre	4	9	2	0.376 sec	100 %	100 %
3. Browse recommended books	4	4	2	0.361 sec	100 %	100 %
4. Browse best-selling books	2	5	1	0.261 sec	100 %	100 %
5. Browse “The Books That Everyone Should Read At Least Once” list	1	2	1	0.261 sec	100 %	100 %
6. Browse the featured list of books	4	4	10	2.59 sec	100 %	100 %
7. Adjust app settings	3	4	5	0.698 sec	100 %	100 %
8. Edit favorite genres	2	3	1	0.259 sec	100 %	100 %
9. Enter a reading challenge	2	4	1	0.266 sec	100 %	100 %
10. View past challenges	2	4	1	0.277 sec	100 %	100 %
11. Update reading progress	2	3	1	0.269 sec	100 %	100 %
12. Show the best books this year	2	3	1	0.255 sec	100 %	100 %
13. Add kindle notes and highlights	3	3	1	0.270 sec	100 %	100 %
Total	33	50	28	6.325 sec	100 %	100 %

B. *Experimental Results of the Proposed Performance Test Results Analysis Approach*

This experiment focuses on executing test cases, then analyzing the test results to detect the performance critical point and heavy load of the AUT. The previously generated test cases were executed. Then, the test results were analyzed and interpreted. JMeter tool is used for test execution. The value of the Thread Group variable (i.e., number of simulated concurrent users) starts from 100 users. Then, the Thread Group value increments iteratively by 50, until it reaches 1000. The minimum and the maximum number of concurrent users are defined as a tester input to the proposed P-TaaS architecture. The minimum and the maximum number of concurrent users shall be defined according to the estimation of the owner of the AUT. In the experiments, multiple trials were made to choose the appropriate minimum number of concurrent users. It was noticed that the error rate was 0 % and the response time is very small at workloads less than 100. However, the response time and error rate values began to upsurge starting from 100 concurrent users. Thus, the test results analysis experiments use 100 concurrent users as the minimum number. The maximum workload is defined in the experiments as the number of workloads where the AUT's behavior (i.e., fluctuation and instability) could be observed. Thus, the maximum workload in the experiments was 1000 concurrent users, as shown in Tables III and IV.

Generally, there are dependent and independent variables in any experiment [38]. The goal of the experiment is to monitor the effect of changing the value of the independent variable on the dependent variable. In our experiments, the independent variable is the workload value (i.e., number of simulated concurrent users). The dependent variables are response time and error rate. The value of the H_Respone_Time is a dependent variable on the response time and error rate.

Table III presents the detailed measurements of test execution and result analysis for the first functionality of the GoodReads App (i.e., Search by book name or author name functionality). The first column in Table III includes the number of concurrent users' access to the AUT. The second column includes the measured response time. The third column includes the calculated H_Respone_Time value for the corresponding workload and response time. The fourth column includes the measured error rate value of the AUT under a certain workload. Error rate value indicates the percent of requests submitted to the AUT with error. The last column includes the time spent on executing the test case within the corresponding workload. It is observed from Table III, that at the workload of 1000 concurrent users, response time increases sharply and H_Respone_Time has the highest value. During the test execution, it was noticed that the error rate became 52.7% at the workload of 1000 concurrent users, which exceeds the threshold error value. This implies that the AUT becomes unstable and may crash at any workload of more than 1000 concurrent users. Thus, it is interpreted that the performance critical turning point occurs at a workload of 1000 concurrent users, a workload of more than 1000 will be considered a heavy load. This leads to an exponential increase in the error rate. Besides, the AUT becomes unstable and may crash. Fig. 5 shows the response time measurement for a different number of concurrent users' accesses. It is observed that when the number of concurrent users reaches 1000, the response time increases sharply and reaches its peak.

TABLE III. EXPERIMENTAL RESULTS OF TEST EXECUTION AND ANALYSIS FOR GOODREADS APP

Workload	**Response Time (sec)**	**H_Respone_Time**	**Error Rate (%)**	**Test Execution Time (sec)**
0	0	0	0	0
100	1,657	1599.1158	0	102
150	1,931	1866.810935	0.333	102
200	2,142	2075.136315	0.25	103
250	1,991	1940.209504	0.2	103
300	9,541	9177.330651	0.17	129
350	19,226	18482.24371	3.86	182
400	9,246	8899.692469	0.25	175
450	25,340	24358.72278	36.33	188
500	4,510	4379.667444	0.4	160
550	9,592	9243.143812	0.64	129
600	26,134	25125.38432	12	205
650	10,818	10426.99425	9.3	143
700	9,632	9296.884107	8.86	172
750	13,546	13050.01291	23.4	152
800	9,256	8950.281099	4.62	141
850	11,409	11014.08098	14.88	189
900	13,766	13275.02628	14.67	228
950	15,958	15379.26698	18.79	113
1000	78,602	71702.83435	52.7	610

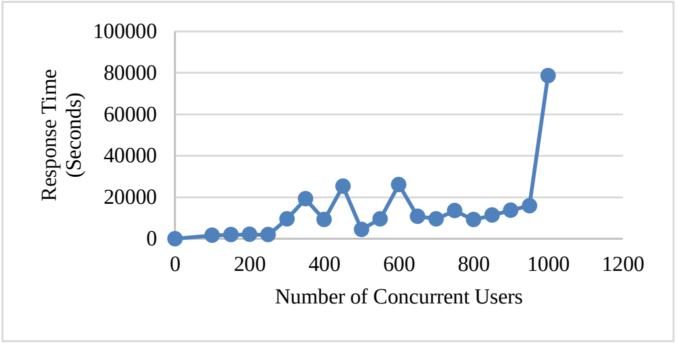

Fig. 5. Response time measurement for several numbers of concurrent users' accesses to the goodreads app.

Similarly, Table IV presents the results for the first functionality of the SpeedTest App (i.e., Show upload/download speed). It is observed from Table IV, that at the workload of 850 concurrent users, the response time increases sharply and the H_Respone_Time has the highest value. During the test execution, it was noticed that the error rate became 56.12 % at the workload of 850 concurrent users, which is the highest error rate that occurred during the experiment. Thus, it is interpreted that the performance critical turning point occurs at the workload of 850 concurrent users. A workload of more than 850 will be considered a heavy load. Fig. 6 shows the response time measurement for a different number of concurrent users' accesses. It is observed that when the number of concurrent users increases, the response time increases sharply and reaches its peak.

TABLE IV. EXPERIMENTAL RESULTS OF TEST EXECUTION AND ANALYSIS FOR SPEEDTEST APP

Workload	Response Time (sec)	H_Respone_Time	Error Rate (%)	Test Execution Time (sec)
0	0	0	0	0
100	663	647.3549	0	99
150	685	681.4788	0	100
200	717	728.3722	0	100
250	609	653.1596	0	100
300	723	776.9281	0	101
350	675	759.3621	0	100
400	706	811.3094	0	101
450	767	889.2544	0	101
500	1,015	1128.796	0	105
550	1,184	1300.434	0	108
600	2,646	2639.006	0	116
650	3,749	3680.732	0	116
700	1,106	589.3899	0	106
750	13,978	13453.63	23.9	236
800	11,976	11538.09	27.12	190
850	20,647	19859.62	56.12	748
900	46,544	44749.01	78.37	258
950	17,977	17301.74	67.3	469
1000	23,779	22873.01	52.4	623

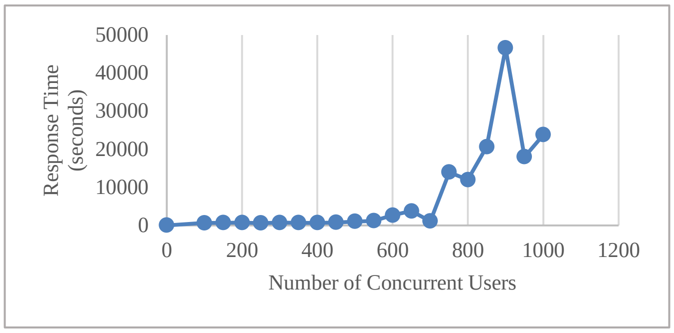

Fig. 6. Response time measurement for several numbers of concurrent users' accesses to the speedtest app.

The experimental results presented in Tables III and IV, and Fig. 5 and Fig. 6 were closely examined. The observations revealed are as follows:

- There is a fluctuation in the response time, especially with workloads lower than the detected heavy load. The fluctuation means that the response time of the AUT is faster than the preceding run. In the experiments, the fluctuation appears more obviously in Fig. 5 than in Fig. 6. For instance, in Fig. 5 the response time was 9,541 seconds at 300 workloads, the response time was 19,226 seconds at 350 workloads, then the response time decreased to 9,246 seconds at 400 workloads. This fluctuation in the response time is expected and is not considered as a problem during the performance test execution. This fluctuation indicates that the AUT does not clean up its resources. The memory usage metric is used to confirm this. If the memory usage remains high after the test is completed, then this implies that the resources are not cleaned-up at the AUT's web server. Thus, there are many other performance metrics (e.g., CPU usage, Memory usage, DB response time) that affect the responsivity of the AUT.

- The response time sometimes decreases under high workloads. However, the error rate values increase exponentially. This occurs despite the expectations of high response time value due to the heavy workload. For instance, in Table IV the response time was 46,544 seconds at 900 workloads and the error rate was 78.37%. However, the response time decreased to 17,977 seconds at 950 workloads and the measured error rate was 67.3%. The reason is that 900 workloads were defined as the heavy workload during the experiment where the AUT became unstable. Thus, the high error rate implies that a considerable percentage of the requests returned failed immediately and the average response time calculated by the JMeter decreased.

C. *Experimental Results of the Effect of Utlizing the Simulated Cloud-based Environment in the Proposed P-TaaS*

An experiment was conducted to evaluate the effect of utilizing the simulated cloud-based environment in the proposed mobile app P-TaaS. This experiment measures the time spent executing the automatically generated test cases sequentially on one virtual machine. Then, it measures the time spent executing the automatically generated test cases simultaneously on multiple virtual nodes. The experimental results for sequential and simultaneous test execution for the GoodReads app and SpeedTest app test cases are shown in Table V. The sequential execution for GoodReads app test cases takes 25 hours and 52 minutes, but the simultaneous execution only takes 8 hours and 37 minutes. The sequential execution for SpeedTest app test cases takes 18 hours and 18 minutes, but the simultaneous execution only takes 6 hours and 6 minutes. From the results, applying the proposed mobile app P-TaaS shows a vast reduction of time in case of simultaneous test execution.

TABLE V. EXPERIMENTAL RESULTS FOR SEQUENTIAL AND SIMULTANEOUS TEST EXECUTION FOR SPEEDTEST AND GOODREADS APPS TEST CASES

Test Case Execution Approach	SpeedTest	GoodReads
	Total Time	Total Time
Sequential Execution	18 hours and 18 minutes	25 hours and 52 minutes
Simultaneous Execution	6 hours and 6 minutes	8 hours and 37 minutes

VI. Discussion and Limitations of the Proposed P-TaaS Architecture

This section discusses the difference between the proposed P-TaaS architecture and other frameworks presented in literature and are mentioned in Section III. These comparisons show that the proposed P-TaaS architecture fulfill the contributions and handle challenges mentioned in Section I. Additionally, this section discusses the limitations of the proposed mobile app P-TaaS architecture.

The proposed P-TaaS architecture is compared with similar cloud-based performance testing frameworks discussed in literature as well as widely used tools in the market. The comparison includes the following criteria: automatic test case generation with high test coverage, simultaneous test execution on multiple virtual nodes in the cloud-based environment, automatic test result analysis, automatic test report generation that includes the performance metrics (i.e., response time and error rate) collected during the test execution at different workloads, and efficient scheduling and resource utilization. The comparison is shown in Table VI. The first five criteria are related to automating the whole performance testing process. The last criterion in Table VI is related to the efficiency of utilizing cloud-based resources. From the comparison, the proposed P-TaaS is considered a comprehensive framework that conducts the whole performance testing process automatically. Moreover, the proposed P-TaaS leverages the cloud-based environment efficiently.

Table VII shows that the proposed scheduling approach achieves efficient resource utilization. Besides, it guarantees the uncertainty of test execution, load balance between resources, and low complexity. Although, the fuzzy sets theory-based approach proposed by Lampe et al. [21] did not address the issue of uncertainty of test execution time. Lampe et al. [21] depend on knowing the test execution time in advance before starting the test, which is difficult to predict before the test started. Metaheuristic methodologies proposed by Rudy et al. [22] are very complex to implement. Besides, both approaches proposed by Rudy et al. [22] and Lampe et al. [21], have high computation time. The three approaches presented in Table VII balance the load between the available resources. The load balance between resources avoids the downtime and reduces the possibility of losing task productivity.

However, the limitations of the proposed mobile app P-TaaS architecture include: (i) not experimenting with other platforms such as iOS; (ii) not conducting the experiment on a cluster of distributed VMs; (iii) the experiments are conducted on VMs with the same capabilities and configurations using the same performance test tool (i.e., JMeter). However, performance testing is environment dependent. The test environment of the AUT shall mimic the real deployment environment of the application. Changing any factor (e.g., test execution tool, VM capabilities) during the test execution may change the test results.

TABLE VI. Comparison Between the Proposed P-TaaS and Performance Testing Tools in Literature and in the Market

Performance Testing Framework	Automatic Test Case Generation	Simultaneous Test Execution	Automatic Test Result Analysis	Automatic Test Report Generation	Efficient Scheduling and Resource Utilization
The Proposed Mobile Apps P-TaaS	Yes	Yes	Yes	Yes	Yes
ATaaS Framework by Prathibhan et al. [15]	No	Yes	No	No	No
SOASTA CloudTest [16]	No	Yes	No	No	o
LoadStorm [17]	No	Yes	Yes	Yes	Yes
Xamarin Test Cloud [18]	No	Yes	No	Yes	Yes

TABLE VII. Comparison Between the Proposed Scheduling Approach and other Approaches in the Literature

Approach	Uncertainty of Test Execution Time	Load Balance Between Resources	Computation Time
The Proposed P-TaaS Schedule Approach	Exists	Exists	Simple and Low Computation Time
Fuzzy Sets Theory-Based Approach Proposed by Lampe et al. [21]	Not Exists	Exists	Long Computation Time
Metaheuristic Methodologies Proposed by Rudy et al. [22]	Exists	Exists	Complex and Long Computation Time

VII. Conclusion

The performance of mobile apps is significantly important. Performance testing assesses and guarantees the reliability and stability of mobile apps when exposed to different workloads of concurrent users' accesses. The lack of performance testing may lead to degradation in mobile apps. Therefore, more attention from the industrial and academic communities is directed to performance testing as a service (P-TaaS). This paper introduced mobile app performance testing based on the TaaS architecture. It accomplishes the entire performance testing process automatically. The proposed P-TaaS adopts efficient approaches for test case generation, simultaneous test execution, test results analysis, and scheduling of test tasks. The experimental results on two different mobile apps prove the effectiveness of the proposed P-TaaS.

VIII. Future Work

In future work, the proposed mobile apps P-TaaS can be extended to include many directions such as: (i) allowing the performance testing of different types of mobile applications running on different platforms, (ii) including more types of testing as: security testing and regression testing., (iii) using a real cloud-based environment for simultaneous test execution on many VMs (e.g., Amazon EC2).

REFERENCES

[1] Iqbal, Muhammad Waseem, Nadeem Ahmad, Syed Khuram Shahzad, Irum Feroz, and Natash Ali Mian. "Towards adaptive user interfaces for mobile-phone in smart world." International Journal of Advanced Computer Science and Applications 9, no. 11 (2018).

[2] Arif, Khawaja Sarmad, and Usman Ali. "Mobile Application testing tools and their challenges: A comparative study." In 2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET), pp. 1-6. IEEE, 2019.

[3] Torres-Sanchez, Elisa Marlen, Jania Astrid Saucedo-Martinez, Jose Antonio Marmolejo-Saucedo, and Roman Rodriguez-Aguilar. "Multi-criteria Decision-Making for Supplier Selection Using Performance Metrics and AHP Software. A Literature Review." In The International Conference on Artificial Intelligence and Applied Mathematics in Engineering, pp. 735-743. Springer, Cham, 2023

[4] Fernandes, Thiago Soares, Álvaro Freitas Moreira, and Érika Cota. "EPE-Mobile—A framework for early performance estimation of mobile applications." Software: Practice and Experience 48, no. 1 (2018): 85-104.

[5] Khan, Habib Ullah, Farhad Ali, and Shah Nazir. "Systematic analysis of software development in cloud computing perceptions." Journal of Software: Evolution and Process (2022): e2485.

[6] Shaqrah, Amin. "Cloud CRM: State-of-the-Art and Security Challenges." International Journal of Advanced Computer Science and Applications 7, no. 4 (2016).

[7] Ya'u, Badamasi Imam, Norsaremah Salleh, Azlin Nordin, Norbik Bashah Idris, Hafiza Abas, and Ali Amer Alwan. "A systematic mapping study on cloud-based mobile application testing." Journal of Information and Communication Technology 18, no. 4 (2019): 485-527.

[8] Ya'u, Badamasi Imam, Norsaremah Salleh, Azlin Nordin, Norbik Bashah Idris, Hafiza Abas, and Ali Amer Alwan. "A systematic mapping study on cloud-based mobile application testing." Journal of Information and Communication Technology 18, no. 4 (2019): 485-527.

[9] Bertolino, Antonia, Guglielmo De Angelis, Micael Gallego, Boni García, Francisco Gortázar, Francesca Lonetti, and Eda Marchetti. "A systematic review on cloud testing." ACM Computing Surveys (CSUR) 52, no. 5 (2019): 1-42.

[10] Kumar, Pawan, and Rakesh Kumar. "Issues and challenges of load balancing techniques in cloud computing: A survey." ACM Computing Surveys (CSUR) 51, no. 6 (2019): 1-35.

[11] Kocatas, Alper Tolga, and Ali Hikmet Dogru. "Enhancing UML Connectors with Behavioral ALF Specifications for Exogenous Coordination of Software Components." Applied Sciences 13, no. 1 (2023): 643.

[12] Usman, Muhammad, Muhammad Zohaib Iqbal, and Muhammad Uzair Khan. "An automated model-based approach for unit-level performance test generation of mobile applications." Journal of Software: Evolution and Process 32, no. 1 (2020): e2215.

[13] Maschotta, R., N. Silatsa, T. Jungebloud, M. Hammer, and A. Zimmermann. "An OCL Implementation for Model-Driven Engineering of C++." In International Conference on Software Engineering Research and Applications, pp. 151-168. Springer, Cham, 2022.

[14] Ali, Amira, Huda Amin Maghawry, and Nagwa Badr. "Performance testing as a service using cloud computing environment: A survey." Journal of Software: Evolution and Process 34, no. 12 (2022): e2492

[15] Prathibhan, C. Mano, A. Malini, N. Venkatesh, and K. Sundarakantham. "An automated testing framework for testing android mobile applications in the cloud." In 2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies, pp. 1216-1219. IEEE, 2014.

[16] SOASTA CloudTest:
https://www.akamai.com/us/en/products/performance/cloudtest.jsp

[17] LoadStorm: https://loadstorm.com/performance-testing-tool/

[18] Xamarin Test Cloud: https://testcloud.xamarin.com/

[19] https://www.softwaretestinghelp.com/cloud-testing-tools/

[20] Liu, Xiaolong, Ruey-Kai Sheu, Win-Tsung Lo, and Shyan-Ming Yuan. "Automatic cloud service testing and bottleneck detection system with scaling recommendation." Concurrency and Computation: Practice and Experience 32, no. 1 (2020): e5161.

[21] Lampe, Paweł. "Fuzzy job scheduling for testing as a service platform." In Smart Innovations in Engineering and Technology, pp. 25-33. Springer, Cham, 2017.

[22] Rudy, Jarosław. "Online multi-criteria scheduling for testing as a service cloud platform." In Smart Innovations in Engineering and Technology, pp. 34-52. Springer, Cham, 2017

[23] Saad, Mohamed, Ali El-Moursy, Oruba Alfawaz, Khawla Alnajjar, and Saeed Abdallah. "Wireless link scheduling via parallel genetic algorithm." Concurrency and Computation: Practice and Experience 34, no. 6 (2022): e6783.

[24] Ali, Amira, Huda Amin Maghawry, and Nagwa Badr. "Model-Based Test Case Generation Approach for Mobile Applications Load Testing using OCL Enhanced Activity Diagrams." In 2021 Tenth International Conference on Intelligent Computing and Information Systems (ICICIS), pp. 493-499. IEEE, 2021.

[25] Ural, Hasan, and Hüsnü Yenigün. "Regression test suite selection using dependence analysis." Journal of Software: Evolution and Process 25, no. 7 (2013): 681-709.

[26] Enriquez, Nathanaël, Gabriel Faraud, and Laurent Ménard. "Limiting shape of the depth first search tree in an Erdős‐Rényi graph." Random Structures & Algorithms 56, no. 2 (2020): 501-516.

[27] Deorowicz, Sebastian. "Solving longest common subsequence and related problems on graphical processing units." Software: Practice and Experience 40, no. 8 (2010): 673-700.

[28] Talha, Kabakus Abdullah, Dogru Ibrahim Alper, and Cetin Aydin. "APK Auditor: Permission-based Android malware detection system." Digital Investigation 13 (2015): 1-14.

[29] Armaly, Ameer, and Collin McMillan. "Pragmatic source code reuse via execution record and replay." Journal of Software: Evolution and Process 28, no. 8 (2016): 642-664.

[30] Apache JMeter: https://jmeter.apache.org/download_jmeter.cgi

[31] 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.

[32] VMware workstation:
https://www.vmware.com/mena/products/workstation-pro.html

[33] Enterprise Architect Tool.
https://sparxsystems.com/products/ea/downloads.html.

[34] Toffalini, Flavio, Jun Sun, and Martín Ochoa. "Practical static analysis of context leaks in Android applications." Software: Practice and Experience 49, no. 2 (2019): 233-251.

[35] SpeedTest app: https://www.speedtest.net/apps/android.

[36] GoodReads app: https://www.goodreads.com/blog/show/1307-introducing-the-all-new-faster-goodreads-android-app-includes-rereads.

[37] https://www.similarweb.com/website/goodreads.com/#overview.

[38] Andreas Jedlitschka, Marcus Ciolkowski, Dietmar Pfahl: Reporting Experiments in Software Engineering. Guide to Advanced Empirical Software Engineering 2008: 201-22.