Linköping University | IDA
Bachelor, 16hp | Innovative programming
Spring term 2016 | LIU-IDA/LITH-EX-G--16/041--SE

Selenium-Testing as a Service

André Andersson

Tutor, Anders Fröberg
Examinator, Erik Berglund

Selenium-Testing as a Service

André Andersson
Linköpings university
Linköping, Sweden

1. ABSTRACT

Selenium has been a method to test web applications for over a decade, it is interacting directly with the browser and has gained support from both browsers and the community. With the growing amount of browsers, mobile devices and operating systems which a web application is expected to work with, services providing these systems for testing web applications against has gained interest. These services provide testing as a service (TaaS), and runs Selenium-tests in the cloud. This research tried to compare the different services with each other in regard to flexibility, cost, simplicity and reliability. I have also tried to see differences between running the tests locally and using these services. The results showed that there are some differences between the services, and the one best suited might depend on the web application.

Keywords
Selenium, Testing as a Service, Testing, Testing on Demand, Selenium-testing, TaaS

2. INTRODUCTION

The concept of Testing as a Service (also known as testing on demand) was created in Denmark around 2009 [1] and since then a lot of services has appeared for testing. There has been a lot of research on the future of TaaS [2, 8], and several different services focusing on testing has emerged. While all the services tries to make life easier for the developers, they have their different strengths and weaknesses and is best suited for different people. For companies and developers it might be hard to know which of the services are best suited for their applications and needs. At the same time, most researchers and developers do agree that it is beneficial to use these services [3, 7, 8].

The web has changed a lot in the last decade, and a web application should now look perfect in all platforms and browsers. A good web application could be one of the most important parts for marketing, both for a start-up company or a big business. In the same time the growth of hardware and software have made it almost impossible to test a web application on all devices and all browsers locally. The explosion of the smartphone-market have not helped, and it is now even more difficult to fully test a web application, especially for start-up companies. Testing locally in-house is both a costly and time consuming task when one think of updating all the browsers, operating systems and devices.

Software practices tries to push features and code to production faster, using among others different types of agile development and test-driven development. Both methods focus a lot on testing of code, by start writing tests and write code which works for the tests. For web development you do however need to run your tests on such a high variety of hardware and software, that it is impractical to have them all in-house and keeping them updated.

To test the user interface of websites, you usually have a set of Selenium-tests to test different parts of the web application. Selenium-tests is one of the latest types of tests which has migrated to testing-services in the cloud. There are several different services focused solely on Selenium-testing as a service, running the tests on a high variety of different hardware and software. This type of services have been popular both for researchers and developers, due to the cost and worktime otherwise required to keep everything up to date.

Services in the cloud does however give the developer less control over the environment. It might take time before they update a specific browser or they do not have a specific device needed. It could also be software-related, such as issues with one platform. This could impact the results, costs and time consumption compared to running the tests locally.

Objective

The objective of this research is to test and compare some of the best testing-services for Selenium-testing in the cloud. While analysing and evaluating these different services, the following should be kept in mind:

- Simplicity to implement
- Flexibility
- Performance
- Client support
- Time consumption
- Cost
- Security
- Reliability

Research Question

Since the main focus of this research will be to compare different Selenium-services for testing in the cloud, a research question has been defined as:

1. What is the differences between the top Selenium-TaaS in regard to simplicity, flexibility, client support and time consumption?

The services will also be compared with running tests locally. For this, a second research question has been defined:

2. What is the main differences between running Selenium-tests locally and in the cloud?

Limitations

Time restraints have been a limitation to this research, and therefore both performance and security have been excluded. Performance could be measured in several ways, such as the amount of concurrent machines, local tunnels or the amount of users. Instead the focus has been on simplicity to implement, flexibility, client support and the time consumption of running these tests.

Another limitation is the amount of services tested. I have limited myself to the services which focuses mainly on Selenium-testing. There are others whom focus on several different kinds of TaaS (load testing for example), which has been excluded.

Some of the pricier plans for the services have also been excluded. The amount of testing time needed for the web application is not considered that big, and a cheaper price plan should be enough. Using other price plans from the services could yield another result.

3. TESTING

Test-driven development and agile development is widely used techniques for software development [15, 16] and relies heavily on writing tests before writing the actual code and to test the code from a user perspective [15]. When the code has successfully passed all tests, that code should be considered done. Microsoft have observed that test-driven development has increased the code quality significantly [16]. The development towards service-oriented-architecture and software-as-a-service has greatly affected the development of software [7], which has required tools for testing web applications.

Selenium is an open-source tool developed by Thought Works, with the goal to help developers test the user interface of web applications [4, 5]. Since Selenium interacts directly with the web application, it becomes easy to test advanced AJAX-calls directly from a user perspective. Selenium is supported by most of the major browsers, operating systems, programming languages and testing frameworks [6, 9]. Since a web application, and especially the back-end, can be served by any programming language, the support for the multitude of languages is one of Selenium's great advantages. Each programming language has its own set of Client Drivers, which will communicate with either a Browser Driver or a Remote WebDriver. The Browser Drivers are implemented by the browser manufactures, which will control the browser programmatically. Selenium has proven to be great with agile development and test driven development, some researchers have even asked whether other kind of testing is required for web applications [6]. Part of agile testing is to test applications and websites from a user perspective, something which Selenium does [5].

There are some other tools besides Selenium for testing the user interface and web applications. Among others QTP, WinRunner and Rational Robot can be used for this, however they have proven to be both costly and had a mixed result. Therefore developers have turned to Selenium for help with automatic testing of user interface [4].

While writing and maintaining Selenium-tests different methods have been approached to keep them both easy to read and maintain. One method to do this are by using Page Object Patterns [11], which essentially abstracts a page in the web application. One example of such a Page Object could be a class *LoginPage* with a method *logIn* which returns a *MainPage*. This design makes the tests both easier to read and maintain [11]. Leotta, Maurizio *et al.* found that while the code base grew slightly, it became both faster and easier to update a test when it failed (because of the test-code). They believe this is a great advantage for new projects, where the web application is updated often to live up to what the users want. When realigning their tests after an update they managed to reduce the amount of source lines of code (SLOC) with 87.77% in their case, and a 65.32% reduction in time, when using Page Object Patterns compared to not using [11].

Selenium is locating elements in the web application by using different kinds of Locators. Locators can be based on the ID, class names, tag name, xPath, CSS-selectors or the link text. Leotta, Maruizo *et al.* also conducted research on the different kind of locators [17], and found that ID-based locators is far better than xPaths. The ID-based locators were better both in time and amount of modified SLOC to realign a test after an update to the web application.

Testing as a Service

Testing as a service is defined as "a model of software testing where an application is tested as a service provided to customers across the Internet." [12]. It is also a method of testing which is increasing in popularity [3]. Researches have studied different types of testing as a service to determine current practices [3, 5, 7, 10, 13]. They have concluded that there are several different types of testing-services [13]:

- Service function testing
- Integration testing API and connectivity testing
- Performance and scalability testing
- Security testing
- Interoperability and compatibility testing
- Regression testing

One strength of the testing-services are that developers do not have to focus on keeping the environment up to date with all browsers, operating systems and devices [10]. That will instead be handled by the service, and the developer can focus on writing tests and develop, and use this environment and service for the tests. A weakness is that the developer will no longer be in control of the testing-environment, and the developer have to hope that the service includes the hardware and software wanted for the testing. Since the tests is running in cloud-environments, the application and tests have to be optimized for this [8], and specific hardware might not be available.

The RemoteWebDriver in Selenium uses DesiredCapabilities to control which browser, version, operating system and additional data which should be started by the Selenium-server, which will start the WebDriver best matched to those capabilities. The services use this to send additional data, such as screen resolution, debug-data, the name of the test running and other information. Exactly how they should be used depends on the service, and how they choose to implement them. It is also possible to send RequiredCapabilities in the same way, something which is required to be able to run the tests. While these are implemented in a similar way, most of the services do not use, or at least not market them, directly as a feature.

Related Work

Antawan Holmes and Marc Kellogg [4] did some research about Selenium and testing web applications. They concluded that it "handles many of the problems very well and doesn't add significant new ones" and mentioned a growing community and need for Selenium.

More research has been done on how to write good Selenium-tests, where Leotta, Maurizio *et al.* have concluded on how to locate elements [17] and use Page Object Patterns [11]. They saw that their tests became both faster and easier to maintain by using these methods.

Researches have already concluded that "cloud testing is becoming a hot research topic" [13], with a lot of different angles on the research. Some researches have found that it is time to migrate tests to the cloud [8] even if it can be costly depending on the project in question. Others have tried to find current practices [3, 7] and have found that while there is issues with testing as a service, more applications will start using them.

Issues and needs with testing as a service have also been explored [13], which includes problem with the environment and the need for security of user data among others. The problems mentioned in that research might not apply any more due to solutions and technical advances. Researchers have also found general information about software testing as a service [12].

Very few of the studies have however compared different testing-services online, and the only one which I have found focused on all types of testing-services [3]. Their goal was to determine practices and future research in the area, but I will focus on the differences between the testing-services and specifically on Selenium.

4. METHOD

The web application used was fully functional from the beginning, however the Selenium-tests for the web application did not exist and was a part of this project to create. Therefore a test-driven development was not possible in this project. The work has been divided into three different parts – the pilot study, creating the tests and then testing the services using the tests created.

The back-end for the web application was built in Java, while the front-end used Angular and HTML5. The framework for testing was JUnit.

Pilot Study

To determine which services should be used and tested Selenium's own home page [14] and earlier research [3, 13], was used as the main information-sources. The process of selecting services was done together with the client, but was based upon a few criteria. The services needs to be able to run towards a non-public web application (usually done with tunnelling), the amount of testing time (automatic testing primarily) and the cost.

Table 1 displays the data gathered during this phase. Each service did provide several different "price plans" where each plan had a different price, different amount of testing time for automatic testing, amount of

concurrent machines running at the same time and users. Several plans from all services were excluded here, mainly since the amount of time needed for automatic testing for this web application is not expected to exceed those levels.

Service Name	Price per Month	Testing Time	Devices
BrowserStack	59 dollar	500 min	1161
BrowserStack	99 dollar	unlimited	1161
Sauce Labs	99 dollar	1000 min	723
Sauce Labs	199 dollar	2000 min	723
TestingBot	20 dollar	400 min	546
TestingBot	30 dollar	1000 min	546

Table 1 containing pricing, amount of automatic testing time and amount of devices.

The price was a bit higher if one choose to be billed monthly instead of annually. There were also different price plans depending on the amount of concurrent machines, in that case the cheapest method was selected. All services had a free trial, but with some differences. TestingBot allowed one to use 100 free minutes to either automatic or manual testing or a maximum of two weeks, BrowserStack had 100 free automatic testing minutes and 30 free manual minutes for a maximum of two weeks while Sauce Labs had 90 hours free automatic testing for a maximum of two weeks.

Writing the Tests

The tests were written in Java, integrated with Maven and done in JUnit 4. They were also built so several tests could run in parallel, in order to utilize all features of the services. While writing the tests, a local Selenium Grid was created, running Chrome and Firefox. While writing the tests I have tried to hold a high standard using both Page Object Pattern [11] and finding elements in a good way [17]. In total five different tests were built for the web application:

- Basic test
- Settings test
- Map test
- Flash test

The basic test is meant to test basic functionality, log in and see that all objects are visible. The settings test was meant to change the settings for a user, and see that the settings are applied correctly. The map test was done to see that the behaviour of the map was correctly, zooming and objects on the map and so on. The flash test tested that a lightning animation in the web application was displayed correctly.

The tests should be developed to work in as many as the web drivers as possible, even if this means different methods for different drivers. The reason for this is that the tests should run in parallel and run in different browsers at the same time, when the tests needs to be able to run on the same code base for all browsers without specific modifications.

During the development some other tests were created too, however these tests have not been used in the measurements while testing the services. The reason they were excluded was because the features were either not stable enough, had ongoing development during the testing phase or did not match up between different services.

After the tests were completely done, they were adapted to work with the services. While doing this, all changes were tracked and counted. I also took notes of all problems detected during this time.

Testing the Services

The services have been tested both in a qualitative and quantitative way. The services have been tested one at a time, and for the measurements the code base has been reverted to a pre-determined state between each service. While setting up a service notes have been taken on all changes done. When the service was working and considered stable, a pre-determined set of browsers and devices have been used to run the tests. After the measurements were done, special features included in each service were looked at to see what they offer.

When making changes to the code, the service own documentation was the primary source of information. If they recommended one way to implement something, that method was used. Each service provided a client to set up a tunnel, in order to be able to run tests on a non-public web application. Since all the tests were written to be running local, that code base have been used as the start point for each service.

The browsers and devices selected for the services were pre-determined, based on which browsers and devices were available on all the services. The same version of a browser was used in all cases to ensure equality when measuring. This means the booting time for the machines should have been the same, however the services might have implemented some different software, something which could not be controlled. The tests did run on the latest versions of Windows and Mac, and Chrome, Firefox, Safari and Internet Explorer. All the tests did not succeed in all the browsers, however those whom failed did it at the same command and at all the services. They failed because of problems in the version of the web application used (and specific browsers), and were the same for all the services.

While the tests were written to be able to run in parallel in several different browsers at the same time, only one thread has been created at a time with one device and one browser. This was done to minimize the effect of the local hardware and network speeds, since traffic needs to go through the tunnel and local network before being served to the services. The same network and local machine was used to test all services, so the impact is believed to be the same. The services also provide a different amount of concurrent machines, which could have affected the result.

When all the tests were done in the automatic environment of each service, other features and functions were looked at. This included manual sessions, screenshots, uploading the result using their API, plugins to other testing services and other things which could be found on their websites. A direct comparison between the different features have not been done, since they do not necessarily try to provide the same features.

5. RESULT

This section will present the results gathered in this study from the pilot study, while writing the tests for the web application and while testing the services.

Pilot Study

Table 1 displays a summary of the results found for each service. Those numbers show a high difference between the services in regard to the amount of different devices. To research this further, I have divided the data into which services support the different browsers and operating systems.

Browser	BrowserStack	Sauce Labs	TestingBot
Chrome	Yes	Yes	Yes
Edge	Yes	Yes	Yes
Firefox	Yes	Yes	Yes
Internet Explorer	Yes	Yes	Yes
Opera	Yes	Yes	No
Safari	Yes	Yes	Yes
Yandex	Yes	No	No

Table 2 displays the services and their browser support.

Table 2 displays which browsers are supported by the different services. Which browser versions are supported have been removed from the data-set. Neither does it say which operating systems the browsers are available in. The only differences in the table is that Opera is not supported by TestingBot, but supported by both BrowserStack and SauceLabs and that Yandex is only supported by BrowserStack, but not Sauce Labs or TestingBot.

Operating system	BrowserStack	Sauce Labs	TestingBot
Android	Yes	Yes	Yes
IOS	Yes	Yes	Yes
Windows Phone	Yes	No	No
Windows 10	Yes	Yes	Yes
Windows 8.1	Yes	Yes	No
Windows 8	Yes	Yes	Yes
Windows 7	Yes	Yes	Yes
Windows XP	Yes	Yes	Yes
Linux	No	Yes	Yes
Mac	Yes	Yes	Yes

Table 3 displays the services and support for operating systems, including mobile systems.

Table 3 displays the different operating systems which are supported by the services. In regard to mobile operating systems, only BrowserStack support Windows Phone. However, BrowserStack is the only service which does not support Linux. TestingBot, as well as the others, do support Windows 8, but TestingBot does not support Windows 8.1, which both BrowserStack and Sauce Labs do. There is also a difference in the amount of supported mobile devices. Sauce Labs support 82 different mobile devices, while BrowserStack support 53 different mobile devices and TestingBot support 10 different mobile devices.

Writing the Tests

The tests were first written to run locally in two different browsers. Some differences were noted compared to running locally, like geolocation giving another position. This required an additional script which modified the position using HTML5. For some services there were also changes required to DesiredCapabilities, for example a flag that the test was being tunneled. However no changes were required for the code to run in the tunnel itself, as long as the tunnel-client was started before running the tests.

Figure 1 displays the result of the modifications and added SLOC. There were no modifications done to the Page Object Patterns, but all the changes were done in the actual test-code. Such as a script for setting geolocation, changes with the URL for the RemoteWebDriver and DesiredCapabilities. This were the minimum amount of SLOC which required changes, while most services had

more “extra” features which could be added (and therefore more changes had to be done).

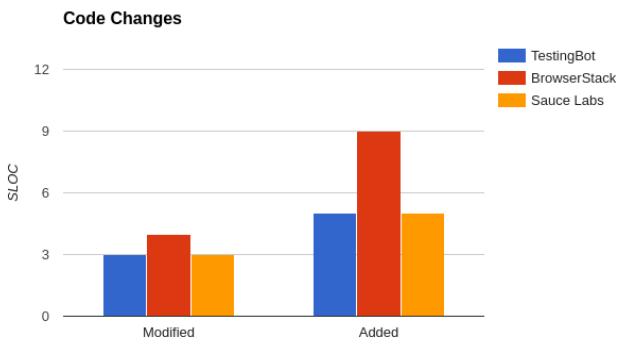

Figure 1 showing the amount of SLOC modified, added and removed for each service.

Testing the Services

To determine the speed of the services and the time it takes to use them, a set of tests was done on all the services. Figure 3 displays the result, with the time it takes for the tests locally, what time the service reported for the tests and how much time the service have charged for.

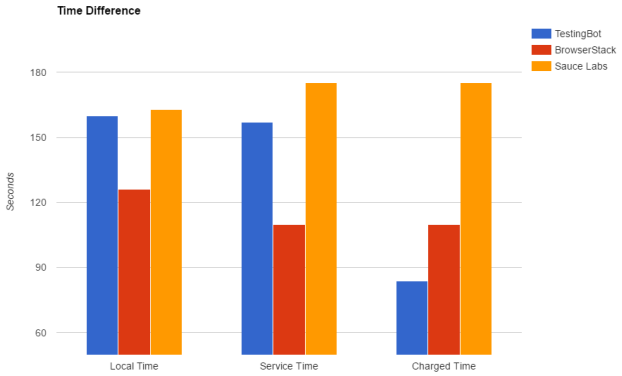

Figure 2 with the amount of time it takes to run the tests according to JUnit locally, the amount of time it takes according to the service provider and the how much time the service provider charged for.

All the services offered manual testing as well, where one could take control over a browser and test the web application manually. All the services also had an API to send the result after the automatic testing. This required additional changes to the code, not counted in figure 2.

They also had other features and issues, discussed further in the discussion.

6. DISCUSSION

This section will discuss the findings in the pilot study as well as adapting and running the tests at the services. There will also be a discussion about threats of validity of this research and future work to be done in the area.

Result

The services should be compared against each other, but they have been evaluated individually. The results have been divided into the pilot study, writing and adapting the tests, evaluating the services and a part with information about each service.

Pilot Study

Most of the services marketed themselves with amount of devices and browsers, reaching several hundreds of different combinations (table 1). They reached this number by using all the versions of all the browsers in all the operating systems. But that number did not really provide anything of value during this research. Instead it was more interesting to see which browsers and operating systems were supported. Also, since both Firefox and Chrome auto-updates itself, it is always up-to-date and it is not likely an older version is needed. Browser developers are also likely to at least try to aim for an equal experience between the operating systems. While the number had some differences in table 1, more similarities between the services was found in table 2 and 3. This might show that the number is more of a marketing trick than anything of real value, but there could be cases there the exact browser version is needed.

While all the services provided emulated or simulated devices for their mobile-section, both BrowserStack and Sauce Labs included real devices as well. These devices could be used both in manual and automatic testing. The amount of mobile devices used available included emulated and simulated devices. According to BrowserStack they should be exactly like real devices in over 99% of the cases. BrowserStack was also the only service to include some Windows Phone-devices.

They all offer a free trial, however the amount of testing allowed during that free trial is a bit different between the services. Sauce Labs allows you to use 90 hours of automatic testing, BrowserStack has 100 minutes of free automatic testing and TestingBot allows 100 minutes of free automatic or manual testing. There are a few other limitations as well, for example BrowserStack limits the manual testing of mobile devices to a few specific devices. Sauce Labs only allow manual sessions on 10 minutes each, including the start-up time for the virtual machine selected. All of these limitations are removed when a price is bought.

Writing and Adapting the Tests

There are several differences between running the tests locally and at a service in the cloud. Among them are that the tests probably will run on more devices and browsers, the geolocation will be different and there needs to be some way to verify which user is running the tests on the services. There could also be additional features that the service provide, which could require even more changes to the code.

When the tests have to be adapted for a service, it will be both harder to switch between the services and there will be more changes compared to running locally. All of the services have made it very easy to switch to them, but have added more features which could be used by the developer. This could be a method to "lock you in", so the developer will stay with their service.

There are also changes which have to be done due to the differences in running tests locally and at a service. This could include geolocation, which requires changes to get a position which can be used for testing. By running the tests locally, the position will always be the same as the developers location (or the controlled environment). Luckily you can use JavaScript to set the success-function from a get location-call and set the position to a pre-determined position. The different location could also mean that the web application shows another language or something similar.

Figure 1 displays the amount of SLOC added and modified for each service, while no services required any SLOCs to be removed. The amount of added and modified SLOC is compared to running the same set of tests locally. The code base for running the tests locally had 82 SLOC, without the inclusion of any Page Object Pattern. The Page Object Pattern was not modified in any way to use the services. It was the minimal amount of SLOC I was able to get the services to run in the same browsers as it did locally. All services gave the possibility for more features, which then required more code changes.

In total the changes were relatively small, only requiring the addition of API-keys, change of the Selenium-server location, addition of geolocation-script and changes related to the DesiredCapabilities. The extra features included an API to send them the result after a test had finished, which allowed you to see and store the result on their services. However, that came with the cost of making huge changes to the code base (the code grew with up to 80%). I believe though, that most users would like to use a local server to store and handle the result. This allows for more modifications, for example the server could send email updates once the tests have completed or let the server tell the service to run the tests at specific times or events. This also has the cost of having to handle and maintain the server.

When the tests were ran locally they were only done in Chrome and Firefox, but when using one of the services several other browsers was used. This meant that the code had to be adapted to several browsers. This took a lot of time, since they all handle some things differently. It could be a render issue, and having one item rendered above another making it un-clickable, or loading issues. For example, Chrome told Selenium it was ready once the DOM was loaded, while Firefox waited until all images were loaded as well. Since the tests were running in parallel, the code base had to be able to handle all browsers. For example, Safari has not implemented the Interactions-API in the WebDriver-implementations, which meant advanced movements (such as MouseOver) were not available. The system could still be tested using JavaScript to execute if such a browser is detected, however that would also mean the interaction with the web application is not tested, only the underlying system.

Evaluating the Services

In figure 2 the difference between the service time and charged time for TestingBot is fairly big. While both Sauce Labs and BrowserStack are charging for the entire time the virtual machine is used, including the booting time of the virtual machine, TestingBot only charge for the time the tests are actually running. In this case the actual time it took to ran the tests were only 53% of the time taken from start to end. While this number will change heavily depending on which set of tests are running, it means that a minute of automatic testing time at TestingBot will last longer than a minute at either BrowserStack or Sauce Labs.

Also noticeable is the time it takes for BrowserStack to complete the tests compared with both Sauce Labs and TestingBot. There could be several reasons why BrowserStack is faster, they might have better hardware. But it could also be that the location of the server-hall is closer or they have optimised the virtual machines and booting time more than Sauce Labs and TestingBot. Another possibility is that virtual machines with popular set ups is already running in sleep mode, so they do not have to create new machines when a test is started.

The time the tests take locally is lower for both TestingBot and BrowserStack compared to the time it takes according to the service, however for Sauce Labs the time is actually longer. This could be because of several reasons, one of them might be that Sauce Labs continue charging the user until the virtual machine has shut down entirely.

TestingBot

TestingBot is the cheapest of the ones tested, but had the least amount of mobile devices. It was easy to set up the tunnel to run a local web application in their service, download it and change the remote web address to point to the tunnel instead. It was easy to interact manually with an automatic testing, and they "live-streamed" the entire process and allowed manual input into the stream on their website. Their documentation was easy to understand and follow, and some of the code could be copied and pasted directly. The documentation included authentication keys required (when logged into their website) to get the code running directly.

However, when running the tests several problems were detected. During the time of this research Microsoft Edge did throw an exception even if the test succeeded, making it hard to trust the results. It could perhaps be possible to work around this issue, by catching that specific exception in the end and mark it as a success, but that has not been tested. TestingBot also had problems with the recorded video when using Linux, something which was probably related to the their recording-software. This made debugging on those devices a bit harder, luckily TestingBot did provide logs about all actions during the testing phase.

BrowserStack

BrowserStack offered the most devices, but also had a higher price point to begin with. They there as well as TestingBot easy to understand and it was easy to create the tunnel to start running a local web application. The code examples they had also directly included authentication keys (once logged in on the service), so some examples could be set up just as they were without any additional changes at all. However, they required an additional DesiredCapability in order to be able to run a local web application, only specifying that the tunnel was used.

They also required more changes to the DesiredCapabilities, especially if logs were wanted or if the tests were ran on mobile devices. Compared to TestingBot it also took some time before BrowserStack was able to update to the latest versions of some browser, such as Chrome 50 and Firefox 46. Since both of these browsers updates automatically, it could be problematic if a new version introduces a bug to the web application. They do however promise updates to browsers within a week.

BrowserStack also included methods to take interactive control directly from the web browser over a specific automatic test. They also had some other features, such as screenshot and responsive design testing, where an URL is provided and several devices are selected and a screenshot is generated for each of those devices.

Sauce Labs

Sauce Labs included most time for the free trial, with 90 hours of free trial before a price plan had to be chosen. Sauce Labs had more mobile devices (including emulators and simulators) than any of the other services.

Noticeable was the time it took to set up Sauce Labs. While it was easy to download the tunnel it did take some time before I was able to find the authentication keys on their website, and without the authentication keys neither the tests or the tunnel could be started. It was hidden within an account page, and it took some time before it could be found. Sauce Labs had a wiki with information, however it did not contain details about how to find authentication keys. It was also somewhat hard to navigate around their website, and finding the exact DesiredCapabilities for different browsers and devices. The other services had a link to such pages. However they did have some good DesiredCapabilities, for example auto accept all alerts which popped up.

The browsers were unevenly updated. Chrome to Windows 10 was updated to version 50, while Chrome to Linux was still at 48 (which was released several months ago) at the end of this research. Sauce Labs also develops Appium, an open-source framework to test mobile devices. While not used in this research, I believe it to be easy to implement with Sauce Labs. Sauce Labs also have plugins to JIRA, a program to help developers track issues, sprints and so on.

Threats of Validity

The local computer which everything was developed on was a Linux-machine, and therefore the amount of browsers was limited to Chrome and Firefox locally. The limitation was because of the amount of hardware available, and another system could not be used. To ensure that the tests were working before making the measurements, TestingBot was used. The changes done when adapting the tests were done before any measurements. Most of the changes adapting the tests were done in the Page Object Patterns, however some small changes were also done in the actual testing code. To make sure the code was not adapted for TestingBot, I switched and ran the tests locally during the development of the tests.

Another threat is that the web application was developed in the same time as the writing of the tests was done. To ensure that all measurements were done correctly, an old version of the web application was used during the measurements. However newer versions wrtr used as well, both to test new functionality in the web application and other aspects of the services. Due to this, the newer versions have not been used when comparing the services against each other. They have instead been used only to compare when running tests locally.

Since the tests were running parallel in different browsers at the same time, the time measurements might not have been entirely correct depending on local hardware and the amount of concurrent machines available at the service. To ensure the measurements were done in the same way, only one thread with one browser was created at a time. This ensured that no differences between the services, which offer different amount of concurrent machines, were made.

The testing was mainly done using the services free trial. The cheaper price plans have however been considered when selecting. This limits the amount of time to use the services, but it does not affect the performance in any other way and should not have made any differences in result. Sauce Labs and BrowserStack had some differences with manual testing, like which devices could be used, however those limitations was not in the automatic testing used in this research. Sauce Labs also limited manual sessions to 10 minutes each. The only other limitations was in the amount of free minutes which could be used for automatic and manual testing, as well as the amount of concurrent machines running.

Future Work

The result in this research have looked at some aspects of the services, however one could look at more aspects. It might be possible to create a service similar to the ones studied in this project, using the cloud for all the devices and install the software to run the Selenium-tests yourself. This might be costly to maintain or set up. The gains of this would be a controlled environment. Just like running local the developer would know exactly what software has been installed and requested.

Another aspect could be on how to improve the services. Improvements could be in several forms including speed, stability or security. One method which could improve the speed would be to have virtual machines with the operating systems running at the services, but in sleep mode. This way you only have to wake up the virtual machine before starting the browser and the testing. To ensure good security you would still have to start a new machine once the testing was done.

While Selenium 2 was built with WebDrivers and a RemoteWebDriver, perhaps other aspects of the framework could be changed to improve the services or Selenium-testing in general. Currently Selenium depends heavily on the browser manufacturers to implement the Browser Drivers, and if the implementation is not working the tests will not work either.

It is also likely that the services will continue to be improved and modified, which could require more research in the future. This research could however be used by both the services to improve their current service and by developers to see what matters most for them and see what they might need.

7. CONCLUSION

Several different aspects of the services are show in the result of this research. There is also some differences between running tests locally and in the cloud. The research leaves a lot of space for future work, especially testing with more web applications and figuring out which service is best suited for a type of web application, if such conclusions can be drawn. Doing this could help both developers and companies to select services best suited for their needs.

What is the differences between the top Selenium-TaaS in regard to simplicity, flexibility, client support and time consumption?

While the services have differences in the amount of devices, browsers and operating systems, they do support the same systems with only a few exceptions. The code changes required is almost the same for all of the tested services, making the difference in regard to flexibility low for the testing of this web application. It is possible that this is only for this web application, and another set up could give another result. Therefore more research is required in this area.

While the time consumption did show a difference between the services, where BrowserStack was the fastest and TestingBot did charge for least amount of time, the result was for one type of web application. The area needs to be further analysed in order to draw any conclusions, with more testing using more web applications.

The services were fairly simple to implement, in more or less all regards. However, there could be differences in other web applications with other set up, which could yield other results.

What is the main differences between running tests locally Selenium-tests locally and in the cloud?

There could be some unforeseen problems with running tests locally compared to running them in the cloud. While the environment will be fully known in advance by running the tests locally, they will not be known when using a service. This could be problematic if the web application is using special technologies, such as geolocation, which is different when running locally and in the cloud.

Running the tests locally limited the amount of browsers available, as well as the amount of real devices. These numbers grew when using a service instead.

How much of a difference it is between running locally and in the cloud depends on the service, the tests written and the web application. There is not known

whether this could have a negative or positive effect, and more research is required.

8. ACKNOWLEDGMENTS

I would like to thank Peter Halvarsson for the feedback, suggestions and help with the timing issues for different browsers related to the Selenium-tests.

9. REFERENCES

1. Gao, Jerry, et al. "Testing as a service (taas) on clouds." Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on. IEEE, 2013.
2. Candea, George, Stefan Bucur, and Cristian Zamfir. "Automated software testing as a service." *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 2010.
3. Riungu-Kalliosaari, Leah, Ossi Taipale, and Kari Smolander. "Testing in the cloud: Exploring the practice." *Software, IEEE* 29.2 (2012): 46-51.
4. Holmes, Antawan, and Marc Kellogg. "Automating functional tests using selenium." *Agile Conference, 2006*. IEEE, 2006.
5. Bruns, Andreas, Andreas Kornstädt, and Dennis Wichmann. "Web application tests with selenium." *Software, IEEE* 26.5 (2009): 88-91.
6. Razak, Rosnisa Abdull, and Fairul Rizal Fahrurazi. "Agile testing with Selenium." *Software Engineering (MySEC), 2011 5th Malaysian Conference in*. IEEE, 2011.
7. Riungu, Leah Muthoni, Ossi Taipale, and Kari Smolander. "Software testing as an online service: Observations from practice." *Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on*. IEEE, 2010.
8. Parveen, Tauhida, and Scott Tilley. "When to migrate software testing to the cloud?." *Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on*. IEEE, 2010.
9. *"Platforms Supported by Selenium", Web. 15 February 2016* http://www.seleniumhq.org/about/platforms.jsp
10. Kaalra, Bhavnesh, and K. Gowthaman. "Cross Browser Testing Using Automated Test Tools." *International Journal of Advanced Studies in Computers, Science and Engineering* 3.10 (2014): 7.
11. Leotta, Maurizio, Diego Clerissi, Filippo Ricca, and Cristiano Spadaro. "Improving test suites maintainability with the page object pattern: An industrial case study." In *Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on*, pp. 108-113. IEEE, 2013.
12. van der Aalst, Leo. "Software testing as a service (staas)." *Sogeti Whitepaper. Available at www. sogeti. com/staas* (2009).
13. Gao, Jerry, Xiaoying Bai, and Wei-Tek Tsai. "Cloud testing-issues, challenges, needs and practice." *Software Engineering: An International Journal* 1.1 (2011): 9-23.
14. *"Selenium Ecosystem", Web. 1 March 2016* http://www.seleniumhq.org/ecosystem/
15. Maximilien, E. Michael, and Laurie Williams. "Assessing test-driven development at IBM." *Software Engineering, 2003. Proceedings. 25th International Conference on*. IEEE, 2003.
16. Bhat, Thirumalesh, and Nachiappan Nagappan. "Evaluating the efficacy of test-driven development: industrial case studies." *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*. ACM, 2006.
17. Leotta, Maurizio, Diego Clerissi, Filippo Ricca, and Cristiano Spadaro. "Repairing Selenium Test Cases: An Industrial Case Study about Web Page Element Localization." In *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on*, pp. 487-488. IEEE, 2013.