WHITE PAPER

THE BENEFITS OF PARALLEL TESTING

Running Selenium tests can be very slow. But the typical response to this slowness is counter to modern software delivery. Teams will attempt to make up for this slowness by reducing test coverage, testing on fewer system and browser configurations, and doing their best not to update their test suite for fear of failed tests and re-runs. The best way to address performance is to change how you test. And parallel testing is the key. With parallel testing you not only decrease the amount of time a test suite takes to run, but also increase coverage, and allow teams to test continuously with more confidence. In this white paper you will learn about the cost of serial testing, what parallel testing is, and the ways you can leverage it.

SAUCELABS

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Today, Development operations are more scalable than ever. Yet there is one area that continues to constrain the ability of organizations to scale: functional testing.

The reason why is that most organizations still rely on sequential tests to deliver quality assurance. No matter how rapidly a business can produce great products, its ability to bring them to consumers quickly is limited by its dependency on a slow, tedious testing process that does not adapt well to changing development needs or ever-faster business processes.

That, at least, is the way things have traditionally been. But there's a replacement for sequential testing. It's called parallel testing, and it can revolutionize your organization's ability to scale its IT operations.

FUNCTIONAL TESTING: WHY IT'S SLOW, AND HOW YOU CAN DO IT FASTER

Sequential testing involves running a long series of tests, one after another, usually on local hardware. In most data centers – even ones that automate operations heavily — the efficiency of sequential testing is limited by two factors.

The first is that testing is often done locally in integration environments. That means the number of tests you can perform at a given time is limited by the size of your physical infrastructure and how comprehensive your testing grid is. Most organizations know this all too well, and are forced to run their tests only in the evening when everyone is off the CI environment, and when it has plenty of time to run.

The second limitation is the time it takes to test thoroughly. You can automate tests quickly using only CPU time. But the time required to run a sequential test suite for the entire application can be slow — very slow.

In other words, local testing environments are bogged down by large maintenance overhead, the time wasted on analyzing failed and broken test runs, and limited ability to run multiple tests at a time. And the more comprehensive the test cases and suite, the slower they run.

You could try to work around this problem by doing less rigorous tests with less coverage, or not running your entire test suite before putting the application into production. But that would mean cutting corners, and increasing the risk that your users encounter bugs, or weak functionality. Or, you could just slow down your release processes, which is the antithesis of modern delivery chains.

Luckily for QA, however, the cloud-centric, software-defined-everything age has ushered in a solution to this testing problem. It's called parallel testing, and it lets you leverage the cloud and virtualization to perform more tests at faster speeds than your physical infrastructure or homegrown testing grid could support on its own.

The benefits of parallel testing extend beyond speed. Traditionally, the greater the test coverage, and the more tests and configurations in play, the slower tests run. But extensive parallel testing environments not only run tests in parallel, they also run on multiple browsers and operating systems at the same time, increasing coverage as well as speed.

THE COST OF ON-PREMISES SEQUENTIAL TESTING

Before delving deeper into the details of parallel testing, it's worth elaborating on why traditional sequential testing is inefficient and risky.

Even when tests are scripted, sequential testing is still a manual and inefficient process. The test engineer installs the software on a test system, then runs a series of tests, one after another. This is the obvious (and often only) way to test when testing is being done by hand, and it takes a lot of time. Even if the builds are automated, the test runs still have to be kicked off one at a time, on each of the required configurations.

When slow testing is the bottleneck for a release — or worse, the cause for a delay — no one is happy. That's especially true if the delay is related to the test environment, rather than a real programming issue like a newly discovered bug.

Consider also that in the DevOps processes of Continuous Integration and Delivery and Deployment, testing needs to happen more often. Most companies with older test environments revert to smaller, less comprehensive tests. That results in even faster accumulation of technical debt, not to mention unhappy users.

So the development teams suffer from two problems when they rely on local sequential testing. The first is that it wastes time and slows release cycles. The second is that it increases the risk of poor product quality. Both of these issues can have insurmountably negative impacts on a company's brand.

WHAT IS PARALLEL TESTING?

Parallel testing is the process of running multiple test cases on multiple combinations of operating systems and browsers at the same time. The process is automated, and it often runs on virtual machines. For the QA professional, no additional effort is required once the appropriate test scripts

have been developed. All the testing solution needs is the test suite, and a pool of resources.

WHAT ARE THE ADVANTAGES OF PARALLEL TESTING?

You can test many more configurations.

Because parallel testing is much faster, it allows you to test a wider array of configurations in the same amount of time. If you're releasing a new app and want to test the UI functionality, you don't have to limit yourself to testing against only the most popular types of OS environments and configurations — you can test each one.

It can radically reduce the time required for testing.

Good software testing must be thorough and comprehensive. In a sequential testing environment, it can be (and usually is) a very time-consuming process. Parallel testing can divide invested time (roughly) by the number of test machines being used in parallel, so that overall testing time is a fraction of that required for sequential testing. For example, if you are running 10 concurrent tests, you can reduce your testing time by a factor of about 10. This can cut days or weeks from delivery time. This reduction in testing time also allows you to make more efficient use of your engineering team. It frees them from a set of necessary but time-eating and repetitive tasks, and allows them to concentrate on more forward-looking and innovative projects.

Because parallel testing is automated and runs in the cloud, the cost per test is considerably less.

Leasing test time on a set of virtual machines will generally be much less expensive than the purchase price plus maintenance costs of acquiring a single bare-metal testing environment. Even the common practice of using development hand-me-down computers as test machines has its costs and practical limits. (As used systems age, they require more maintenance, and they become less capable of duplicating real-world deployment conditions.) Cloud-based testing grids mean environments have high availability, and they are up-to-date.

Testing science: repeatable, measurable, accurate.

A less obvious benefit is the ability to focus on making tests repeatable, measurable, and accurate. By simply testing faster, you can test more, and testing more produces more actionable data to find bugs and also improve test cases. Automated parallel testing delivers strong data on test results with key indicators of where tests can be improved. The net result is a test process that is more scientific.

PUSHING THE TEAM FORWARD BY INCREASING TESTING MODULARITY

Parallel testing offers crucial organizational advantages, too.

That's because it makes it easy to break tests into many smaller, autonomous parts. This leads to more modularity for DevOps teams.

In modern testing, each test ideally consists of a single function or operation, plus any necessary dependencies. These tests are then divided across several test systems. For instance, if you have 20 test cases and are using five test machines, each machine would be assigned a set of four tests, and all five test machines would operate at once, cutting your testing time down to 20% of that required for sequential testing on one machine. To expand on this example, if you had 25 test machines, you could run five sets of identical parallel tests, with each set running in a different browser/operating system combination.

This also helps with aging test suites. Sequential tests start with a logical order, but as new tests are added (often by simply tacking them on at the end), and old, obsolete tests are removed (or more often, not removed, even though they should be), entropy sets in, and testing becomes a jumble of out-of-sequence steps. The process of revising such a test regime can become so time-consuming that it is put off until it is no longer possible to follow the test schedule.

Parallel testing eliminates this problem. With parallel testing, when you need to add a new test, you can break it down in the same way, and add it (or its component sub-tests) to the schedule as you see fit. Removing a no-longer-needed test becomes equally simple: You simply take it out of the schedule, and rearrange the remaining tests to produce a new, optimized schedule. The process of revising the test schedule goes from a major and often-postponed chore to a quick and fairly simple occasional task.

PARALLEL TESTING EXAMPLE: BRINGING SELENIUM GRID TO THE CLOUD

Parallel testing can pay big dividends in all sorts of contexts. But one task for which organizations are likely to find it particularly useful is the testing of how apps behave under different combinations of Web browser and OS environments.

If you test apps regularly, you are probably already familiar with Selenium Grid, a very handy tool that makes it easy to mimic different types of browser and OS environments for testing purposes. Selenium is powerful. But it also takes time to set up, and it is designed to run tests sequentially rather than in parallel.

By adding the cloud to the mix, you can get much more out of Selenium. If you run Selenium in the cloud using virtual machines, you can easily spin up as many

testing environments as you need and run the tests concurrently. This gives you all the power of Selenium, without the drawbacks of sequential testing. And thanks to parallel testing, test results can be complete in as little as one-tenth of the time.

Plus, Selenium running in the cloud simplifies the task of running tests not only across different browser environments on a single OS, but on all OSes. Because you can spin up as many virtual machines as you need to run your tests, you can quickly process whichever OS/browser combinations you require. And whenever your release plans change (if you decide to target a new type of OS market, for example), it's trivially easy to update your testing environment accordingly.

A cloud-based Selenium solution like that from Sauce Labs lets you run tests in parallel on over 700 browser and OS combinations, with no need to maintain complex local testing environments.

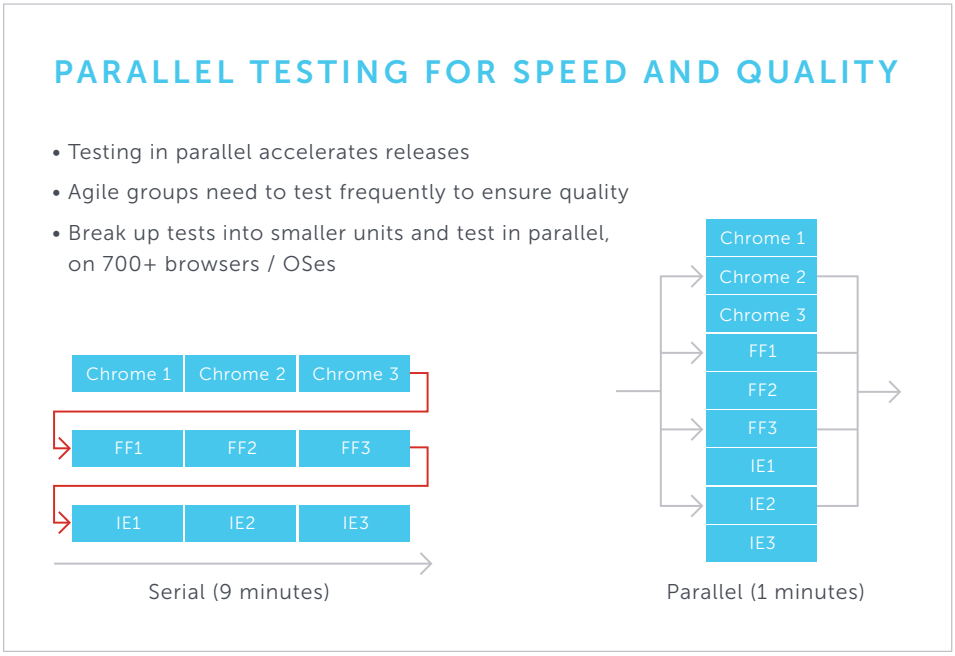

Image provided by Sauce Labs

PARALLEL TESTING MYTHS DEBUNKED

New does not always equal better. But it is hard to make an argument against parallel testing. If you leverage a robust testing cloud at all, parallel testing is the obvious option in most cases, whether or not it's asked for.

Still, skeptics might ask questions like these:

Is parallel testing as accurate and comprehensive as sequential testing?
If anything, parallel testing is even more accurate and comprehensive. Sequential testing has been the standard in software and a very broad range of other fields — not because of any kind of superiority over other kinds of

testing, but simply because it was and in many ways still is the easiest and most practical way to organize and conduct tests.

Is there a relationship between parallel testing and distributed testing?
Distributed testing and parallel testing both require several computers running simultaneously, but they are organized differently, and have somewhat different goals.

Distributed testing is typically used to test distributed software — systems with components which run on different machines, and interact with each other, such as a client-server system, or an Internet-based application with major functional units running in different locations. The goal of distributed testing is to find out how each of the different components of the system interact. While multiple computers will be running simultaneously during the test, each one will be running a different component of the application, and will be playing a different role.

When using parallel testing, on the other hand, each test system will be running the same application, and the goal will be to perform multiple tests that reveal the way that application functions on a single system. In contrast to distributed testing, each test system will be playing the same basic role.

This does not mean, however, that parallel and distributed testing are incompatible. It is possible to run parallel tests of a distributed system. To do this, you would set up multiple distributed test installations, treating each one as a single test unit in a parallel test. Distributed tests can be done on virtual machines, but they are more likely than other kinds of tests to be hardware-based, since they generally involve systems in which the location of components on physically separate machines is an important factor. This kind of parallel-distributed testing is feasible when you work with a testing service such as Sauce Labs, where multiple real test systems are readily available.

What about tests with built-in sequences or dependencies?
Parallel tests do not prevent a sequence of events or conditions from being executed. The modular aspects of tests allow you to daisy-chain the order of tests, building a more comprehensive test. Tests of this sort are sequential in the sense that the sequence of events in the test is important, but events can be parallelized just as easily as any other test. They are simply multi-step tests in which the test system is not set back to its original state after each step. If tests require a common database, and if each test must have sole access to that database when it is running, the opportunities for parallelization would appear, at first glance, to be very limited. If each test is not dependent on the

outcome of the previous test, however, you can split the tests into several groups, and run each group of tests on a separate VM with its own copy of the database. This will allow you to run all of the groups in parallel.

YOU CONTROL IT

An important benefit to keep in mind is that parallel testing does not require you to change the kinds of tests you perform. Additionally, it does not mean you cannot perform a sequential test if desired.

Parallel testing simply offers a different framework for testing, which can easily accommodate most or all of your existing test regime. The transition from sequential to parallel testing does not force you to sacrifice test targets, strategic testing goals, or test quality. It increases the range and depth of your test program, allowing you to better meet your test goals, and it improves the quality of your testing. When you make the move to parallel testing, you are not abandoning traditional software testing. You are simply applying modern development practices and technology to it.

MAKING THE MOVE TO PARALLEL TESTING

The shift to parallel testing might seem overwhelming, but it does not have to be. And it can happen faster than you think.

Adopting parallel testing does not mean starting over or re-factoring all your tests at once. All you need as a starting point is to move the running of your tests on your current environments to an environment that supports parallel testing, like a virtual data center. From there, follow these steps to complete the process:

Map out how each of your tests can be broken down into atomic units, and make an approximate count of the results.
This number, along with the number of unique system configurations that you require, will help you determine how many virtual machines you will need to use. For example, if you have 750 tests and each executes for 2 minutes and you need to test across 2 browser / OS combinations, your tests would run for 50 hours in sequence or in about 1 hour if run in parallel on 50 concurrent instances. You can also apply a basic load-balancing technique to economize time and VM use, as long as you know approximately how long each test is likely to take. By combining short and long tests, you can arrange it so that each set of tests ends at roughly the same time on all test machines, limiting the amount of time any test system sits idle. This gives you the start of a rough roadmap for your transition to parallel testing. Along with a plan (and a schedule) for breaking down larger tests and arranging them in

efficiently timed sets, this roadmap should include automation of any tests that are not currently automated.

There are a variety of automated testing tools available, of course, and while most are not specifically designed for parallel testing, many of them (Cucumber and Selenium, for example) lend themselves well to a parallel testing regime.

Look at your existing testing regime and decide which tests you want to include in the initial move from sequential to parallel.
Tests that are good candidates for parallelization are those that can be broken down into simple actions or short sequences of actions. These include tests that may require a large number of repetitions or iterations, such as stress tests. Many tests in a traditional testing regime already consist of simple actions, but even complex tests can be broken down into self-contained atomic units.

User interface tests and API tests are generally very good candidates for parallelization. Web-based applications need to be tested across multiple browsers, and APIs can subject the program to a wide variety of conditions that may only partly be under the developer's control.

You do not need to make the switch to parallel testing all in one move.
You can transition in steps, starting with the tests which are most easily converted to parallel, then adapting some or all of the remaining tests after your team has had time to adjust to the parallel testing environment. Once you have started to use parallel testing and have integrated it into your development process, it is easy to expand the quantity and scope of parallel tests as the need arises.

A very common (and generally quite easy) way to begin a parallel testing program is by running a full set of parallel tests on a single platform (for example, on a browser, or at most a handful of browsers, running on a single operating system), followed by the same set of parallel tests on another platform, with another platform after that, and so on — in effect, handling single-platform blocks of parallel tests in sequence.

Once your testing team is sufficiently familiar with the procedures for setting up the tests for each platform, you can begin to expand to a more advanced parallel testing regime in which you use greater parallelization to test multiple browsers in multiple operating systems in parallel. You can use this kind of expansion to increase the number of browser/operating system combinations that you test, and do so at a pace that suits both the requirements and practical capabilities of your operation.

A FASTER TEST RUN IS A PENNY EARNED

The faster you run a test, and the more tests you run, the more frequent the releases and the more defects you can find. The math is simple. Parallel testing decreases the time cost of maintaining test infrastructure, the time cost of test failures, the financial cost of delayed releases, the cost of technical debt, and finally, the branding cost of unhappy customers due to poor application quality.

Even without changing the way you test, a shift to a parallel testing environment will provide an immediate speed enhancement — which means every test that still runs on existing environments is a cost that can be avoided. And every test run in parallel is an opportunity to test your application better and faster at the same time.

ABOUT SAUCE LABS

Sauce Labs ensures the world's leading apps and websites work flawlessly on every browser, OS and device. Its award-winning Continuous Testing Cloud provides development and quality teams with instant access to the test coverage, scalability, and analytics they need to deliver a flawless digital experience. Sauce Labs is a privately held company funded by Toba Capital, Salesforce Ventures, Centerview Capital Technology, IVP and Adams Street Partners. For more information, please visit saucelabs.com.

SAUCELABS

SAUCE LABS INC. - HQ
116 NEW MONTGOMERY STREET, 3RD FL
SAN FRANCISCO, CA 94105 USA

SAUCE LABS EUROPE GMBH
NEUENDORFSTR. 18B
16761 HENNIGSDORF GERMANY

SAUCE LABS INC. - CANADA
134 ABBOTT ST #501
VANCOUVER, BC V6B 2K4 CANADA