**TestNG Parallel Execution**

[***cross-browser testing in TestNG***](https://www.toolsqa.com/testng/cross-browser-testing-using-testng/) using Selenium, there was a noticeable event. When the test suite executes, the browsers open in a serialized manner, i.e., one after the other. If you are unaware or do not remember, I would recommend going through the tutorial as it will make a lot of sense to the existence of this topic later in the chapter. Serialization has been a pain for engineers for a long time which gave birth to the idea of ***TestNG Parallel Execution***.

The unnecessary time it took to execute the jobs generated by CPU gave birth to the idea of "***cores***" in CPUs whose work would be to run the jobs in parallel. Imagine FIFO case where a job is executing with a batch time of ten ns and a small job of batch time one ns is just waiting for its turn. With the same intentions of parallelizing things, we will introduce parallel test execution in TestNG using Selenium in this tutorial along with the following key topics:

* ***What is Parallel Testing, and why is it important?***
  + ***Advantages of Parallel Testing***
  + ***Disadvantages of Parallel Testing***
* ***Where can we apply Parallel Test Execution in TestNG?***
* ***How to perform Parallel Execution in TestNG?***
  + ***Running test methods parallelly in TestNG using Selenium.***
  + ***Introduction to Threads in TestNG***
  + ***Performance comparison between serialized and parallelized test execution in TestNG***
  + ***Running test Classes parallelly in TestNG using Selenium.***
  + ***Running test Suites parallelly in TestNG using Selenium.***
* ***Configuring the test methods to run parallelly in TestNG***
* ***Parallel test execution using DataProviders in TestNG***

## What is Parallel Testing and Why is it important?

***Parallel testing or parallel execution, as the name suggests, is a process of running the test case parallelly rather than one after the other***. In parallel testing, the program's multiple parts (or modules) execute together, saving the testers a lot of time and effort. The operating system's functionalities do this, but as a user, we need to trigger parallel execution through [***TestNG***](https://www.toolsqa.com/testng/what-is-testng/). As an example, you can think of having software with two different versions and running them in parallel with the help of TestNG. Parallel execution would give us the correct idea of the stability and performance of the software much faster than running serially.

Parallel testing is used heavily with [***Selenium***](https://www.toolsqa.com/selenium-webdriver/selenium-tutorial/) because of the importance of cross-browser testing in the market today. If we have so many browsers with a different version, we can just create a browser matrix and run the tests parallelly, saving us a ton of resources such as time.

### ***Advantages of Parallel Testing***

If we look at the bigger picture, parallel testing has the following advantages:

* ***Reduces Time***: Running the tests in parallel reduces the overall execution time.
* ***Allow Multi-Threaded Tests***: Using the parallel execution in TestNG, we can allow multiple threads to run simultaneously on the test case providing independence in the execution of different components of the software.

### ***Disadvantages of Parallel Testing***

As the two sides of the coin, parallel testing in TestNG also offers some disadvantages given as follows:

* ***Fails On Dependent Modules***: Parallel testing allows independent running of modules simultaneously. Due to this, we cannot go ahead with modules that are dependent on each other, and this occurs quite frequently while testing. So, either we run serially or dissolve independence, which takes extra time and effort.
* ***Knowledge Of Program Flow***: The tester should be well-versed with the flow of the program to create parallel testing modules. A slight interdependency can bring down the whole test case execution. The tester should also know which modules to run in multiple threads and which ones to run in the same threads etc.

Parallel testing in TestNG using Selenium helps us to deliver the projects at a faster rate in this agile and continuous delivery working environment, which is challenging in its way. Before looking at the process of how to perform parallel execution of tests in TestNG using selenium webdriver, let's see what the different areas inside the code that we can parallelize in TestNG are.

## Where can we apply Parallel Test execution in TestNG?

Before starting with the code, let's answer a genuine question, "***Where can we apply parallel testing in TestNG?***". TestNG has its rules too. Although it is reasonably evident that parallel testing must be used with the test case methods to run them in parallel TestNG offers three more areas where we can go ahead with parallel testing, combining these four areas, parallel testing accepts the following keywords (values) in TestNG:

* ***Methods***: This will run the parallel tests on all *@Test* methods in TestNG.
* ***Tests***: All the test cases present inside the <test> tag will run with this value.
* ***Classes***: All the test cases present inside the classes that exist in the XML will run in parallel.
* ***Instances***: This value will run all the test cases parallelly inside the same instance.

Alright, we are now all set to run our first test case parallelly in TestNG using Selenium. (Refer Cross-Browser testing In TestNG before moving forward)

## How to perform Parallel Execution in TestNG?

Parallel test execution in TestNG triggers with the help of keyword "***parallel***". In this variable, we can assign any of the four values that are discussed above. We will look into them one by one in this section.

### ***Running test methods parallelly in TestNG using Selenium***

In the following section, we will be running the test methods in parallel using TestNG. So, all the methods that come under the annotation *@Test* will run parallel when we execute the test suite. The following code will initialize the drivers of two different browsers (Chrome and Firefox) in parallel.

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.openqa.selenium.chrome.\*;

import org.testng.annotations.Test;

public class ParallelTest {

public WebDriver driver;

@Test

public void FirefoxTest() {

//Initializing the firefox driver (Gecko)

driver = new FirefoxDriver();

driver.get("https://www.demoqa.com");

driver.findElement(By.xpath("//\*[@id=\"app\"]/div/div/div[2]/div/div[1]/div/div[1]")).click();

driver.quit();

}

@Test

public void ChromeTest()

{

//Initialize the chrome driver

driver = new ChromeDriver();

driver.get("https://www.demoqa.com");

driver.findElement(By.xpath("//\*[@id=\"app\"]/div/div/div[2]/div/div[1]/div/div[1]")).click();

driver.quit();

}

}

If we just execute the above code as TestNG Test, the tests will execute serially. To run them parallelly, head over to the testng.xml file, and write the following code:

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >

<suite name = "Parallel Testing Suite">

<test name = "Parallel Tests" parallel = "methods">

<classes>

<class name = "ParallelTest" />

</classes>

</test>

</suite>

***Note***: The "***methods***" value is assigned to the parallel attribute since we are aiming towards the execution of the parallel methods.

Run the XML file as [***TestNG Suite***](https://www.toolsqa.com/testng/testng-test-suite/) and notice that both the drivers must have opened together, proving we are on the correct path. Using parallelism in TestNG, we can save a lot of time and perform other testing queries instead.

### ***Introduction to Threads in TestNG***

With this, I would like to introduce the concept of threads while performing parallel execution in TestNG. Threads in parallel testing refer to different parts in which the test execution will be divided and run parallelly. So if there are two threads and two methods, they will take one method each and run them parallelly (if we are running the methods parallelly). But if there are three methods and two threads, one will have to wait until one thread is free and takes up that method for execution. Therefore, we need to specify the number of threads we want to run while performing parallel testing in TestNG.

But, if you see in the above code, we did not specify any thread value, and still, the methods ran parallelly. How could that happen? If you are thinking what I am thinking, you are right. ***The TestNG has a default value of thread = 5 for parallel testing***, and since the methods were just two, there was no problem.

The operating system assigns a thread ID to every thread that we create for a process. From now on into the chapter, to demonstrate parallelism in [***TestNG***](https://testng.org/doc/), we will make use of the thread id. The function that returns the thread id is ***Thread.currentThread().getId()*** method as in the following code snippet:

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.openqa.selenium.chrome.\*;

import org.testng.annotations.Test;

public class ParallelTest {

public WebDriver driver;

@Test

public void FirefoxTest() {

//Initializing the firefox driver (Gecko)

System.out.println("The thread ID for Firefox is "+ Thread.currentThread().getId());

driver = new FirefoxDriver();

driver.get("https://www.demoqa.com");

driver.findElement(By.xpath("//\*[@id=\"app\"]/div/div/div[2]/div/div[1]/div/div[1]")).click();

driver.quit();

}

@Test

public void ChromeTest()

{

//Initialize the chrome driver

System.out.println("The thread ID for Chrome is "+ Thread.currentThread().getId());

driver = new ChromeDriver();

driver.get("https://www.demoqa.com");

driver.findElement(By.xpath("//\*[@id=\"app\"]/div/div/div[2]/div/div[1]/div/div[1]")).click();

driver.quit();

}

}

Mention the thread-count inside the XML file.

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >

<suite name = "Parallel Testing Suite">

<test name = "Parallel Tests" parallel = "methods" thread-count = "2">

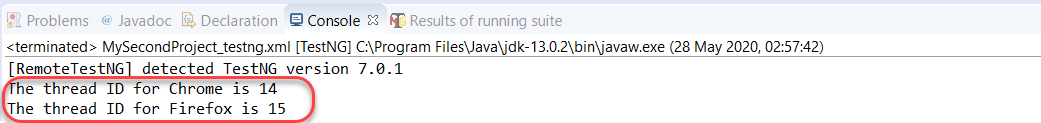
<classes>

<class name = "ParallelTest" />

</classes>

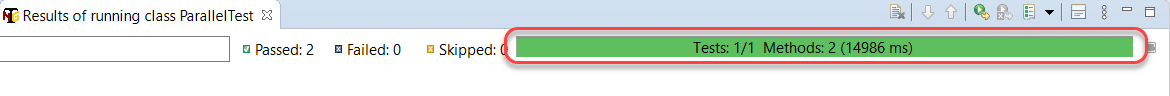
</test>

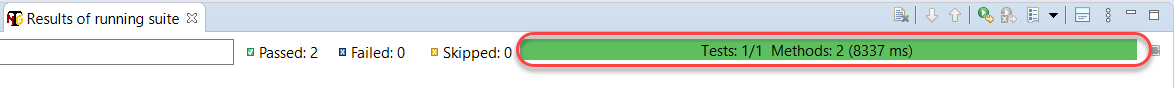
</suite>

Run the test suite, and the console will display the thread ID.

***Note***: The thread value depends on the operating system, and it can be assigned any value. Do not worry about the same number as in the above screenshot.

### ***Performance comparison between Serialized and Parallelized test execution in TestNG***

In the above couple of sections, we ran the methods parallelly. As I mentioned, if we run the test methods directly through the test case file (Run As -> TestNG Test) rather than the XML file, they will run serially. Run the file in the same way (serially) and notice the time taken for overall execution.

Now run the test suite parallelly using the XML file and observe the time taken.

For just two methods in the test case, there was a significant time difference in both the methods. Imagine the time it would save for us during practical test execution with a lot of test cases and multiple browsers.

### ***Running test classes Parallelly in TestNG using Selenium***

As I mentioned, we can also mention the value of the parallel attribute as "***classes***" to run the methods of different classes. To demonstrate this, we will declare two classes as ChromeTest.java and FirefoxTest.java, which contains their respective tests.

The following code belongs to the file ***ChromeTest.java***.

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.\*;

import org.testng.annotations.AfterTest;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Test;

public class ChromeTest {

public WebDriver driver;

@BeforeTest

public void beforeTest() {

System.out.println("Initilizing the Google Chrome Driver");

driver = new ChromeDriver();

}

@Test

public void ChromeTestMethod()

{

//Initialize the chrome driver

System.out.println("The thread ID for Chrome is "+ Thread.currentThread().getId());

driver.get("https://www.demoqa.com");

driver.findElement(By.xpath("//\*[@id=\"app\"]/div/div/div[2]/div/div[1]/div/div[1]")).click();

}

@AfterTest

public void afterTest() {

System.out.println("Closing the browser ");

driver.close();

}

}

The following code belongs to another class called ***FirefoxTest.java***.

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.annotations.AfterTest;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Test;

public class FirefoxTest {

public WebDriver driver;

@BeforeTest

public void beforeTest() {

System.out.println("Initilizing the Google Chrome Driver");

driver = new FirefoxDriver();

}

@Test

public void FirefoxTestMethod()

{

//Initialize the chrome driver

System.out.println("The thread ID for Firefox is "+ Thread.currentThread().getId());

driver.get("https://www.demoqa.com");

driver.findElement(By.xpath("//\*[@id=\"app\"]/div/div/div[2]/div/div[1]/div/div[1]")).click();

}

@AfterTest

public void afterTest() {

System.out.println("Closing the browser ");

driver.close();

}

}

Once we finish, we just need to parallelize them in the XML file.

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >

<suite name = "Parallel Testing Suite">

<test name = "Parallel Tests" parallel = "classes" thread-count = "2">

<classes>

<class name = "ChromeTest" />

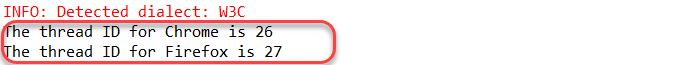
<class name = "FirefoxTest" />

</classes>

</test>

</suite>

***Note***: The parallel attribute contains the value as "***classes***" since we are focusing on running the classes in parallel.

Run the above TestNG suite and notice that two different threads will create during our parallel execution in TestNG.Alright. It was how we run parallel classes in TestNG. In the next section, we will run parallel suites in TestNG.

### ***Running Test Suites parallelly in TestNG using Selenium***

With the value of parallel parameter changed to "***tests***", we can run all the tests that are available inside the suite parallelly. But do remember to put appropriate thread-count value otherwise lesser threads creation would decrease the productivity.

Observe the following code written in selenium web driver to use a chrome driver.

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.\*;

import org.testng.annotations.AfterTest;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Test;

public class ChromeTest {

public WebDriver driver;

@BeforeTest

public void beforeTest() {

System.out.println("Before Test Thread Number Is " + Thread.currentThread().getId());

driver = new ChromeDriver();

}

@Test

public void ChromeTestMethod()

{

//Initialize the chrome driver

System.out.println("The thread ID for Chrome Test is "+ Thread.currentThread().getId());

//System.setProperty("webdriver.chrome.driver","chromedriver.exe");

driver.get("https://www.demoqa.com");

driver.findElement(By.xpath("//\*[@id=\"app\"]/div/div/div[2]/div/div[1]/div/div[1]")).click();

}

@AfterTest

public void afterTest() {

System.out.println("After Test Thread Number Is " + Thread.currentThread().getId());

driver.close();

}

}

The XML file to run the above test would be as follows:

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >

<suite name = "Parallel Testing Suite" parallel = "tests" thread-count = "2">

<test name = "Parallel Tests1">

<classes>

<class name = "ChromeTest" />

</classes>

</test>

<test name = "Parallel Tests2">

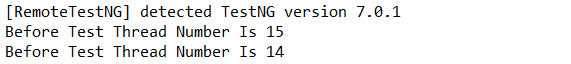
<classes>

<class name = "ChromeTest" />

</classes>

</test>

</suite>

In the console, observe the thread count for the three methods.test execution time for parallel testing in TestNGtest execution time for after annotation

We are getting the number 14 and 15 for every test which denotes that they ran parallelly.

## Configuring the Test Methods to run parallelly in TestNG

In this complete tutorial, we discussed how to run the test methods, classes, and suites parallelly in TestNG using selenium web driver. It included mentioning the thread-count to tell TestNG how many threads we would like to create. But, ***in TestNG, we also get the liberty to run a single test method parallelly by configuring it inside the test code itself***. We do it by making a few changes in @Test annotation. Observe the following code and the parameters used in @Test annotation.

import org.testng.annotations.Test;

public class TestNG

{

@Test(threadPoolSize = 4, invocationCount = 4, timeOut = 1000)

public void testMethod()

{

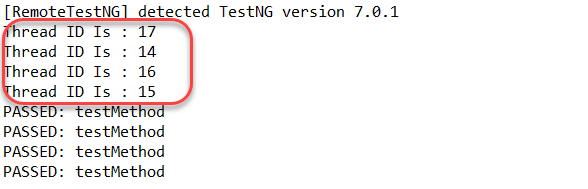
System.out.println("Thread ID Is : " + Thread.currentThread().getId());

}

}

In the @Test annotation, we need to mention three parameters:

* ***threadPoolSize***: The number of threads we would like to create and run the test parallelly.
* ***invocationCount***: The number of times we would like to invoke this method.
* ***timeOut***: The maximum time a test execution should take. If exceeded, the test fails automatically.

Run this test and notice the console.

There are four different numbers because we mentioned the thread count as four in the *@Test* annotation. This way, we can use the test method itself to run it parallelly in TestNG.

## Parallel test execution using DataProviders in TestNG

Apart from running the tests parallelly through the XML file, we can also use dataproviders in TestNG to achieve the same goal. It is an option for parallel execution of tests in TestNG. Before jumping onto the code, I recommend going through the concepts of ***[DataProviders in TestNG](https://www.toolsqa.com/testng/testng-dataproviders/)*** so that everything is understood clearly.

Observe the following code, which uses a dataprovider called "***dp***" to return a number.

import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;

public class TestNG

{

@Test(dataProvider = "dp")

public void testMethod(int number)

{

System.out.println("The Thread ID for "+ number + " Is : " + Thread.currentThread().getId());

}

@DataProvider(name = "dp",parallel=true)

public Object[][] dp1() {

return new Object[][] {

new Object[] { 1 },

new Object[] { 2 },

new Object[] { 3 },

new Object[] { 4 },

new Object[] { 5 },

new Object[] { 6 },

new Object[] { 7 },

new Object[] { 8 }

};

}

}

***Note***: An extra parameter "***parallel***" is required to initiate parallel execution in TestNG using dataproviders.

Now we need to call this dataprovider using the XML file.

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >

<suite name="ParallelExecutionDP" data-provider-thread-count="3" >

<test name="ExecuteDPParallelly">

<classes>

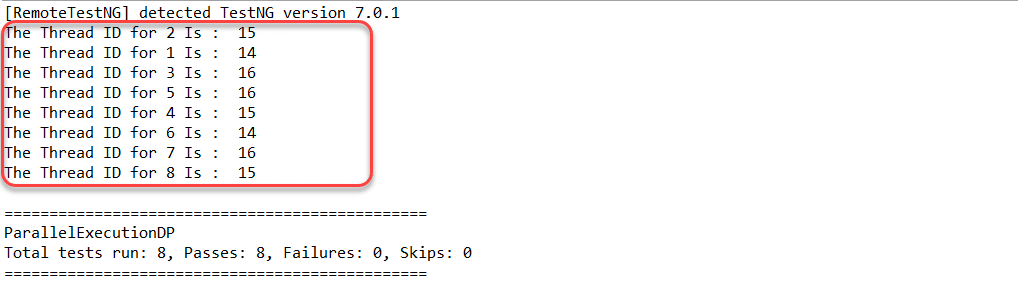
<class name="TestNG"></class>

</classes>

</test>

</suite>

***In the XML file, we need to add one more parameter called data-provider-thread-count to mention the thread count we need to initialize***. We can't use the standard thread-count parameter here as we did with the "***parallel***" parameter above.

Run the test suite using the XML file and see the thread ids.

And there you go. The three threads have executed parallelly, giving us more efficiency and lesser time.