1. What is Thread Group?

A thread group is a set of threads/users executing the same scenario.

**Understand the Process of applying the Load using Thread Group**

**Introduction to Listeners:**

1. View Results tree.
2. Aggregate Report.
3. Graph Results.

**Understand the J-Meter Load parameters to analyze results**

**Samples:** No of users hit that specific request = 2153

**Average:** It is the average time taken by all the samples to execute specific label. = 129 Milli seconds

**Min:** The shortest time taken by a sample for specific label.

**Max:** The longest time taken by a sample for specific label

**Std. Dev.:** This shows the set of exceptional cases which were deviating from the average value of sample response time. The lesser this value more consistent the data. Standard deviation should be less than or equal to half of the average time for a label.

**Error%:** Percentage of Failed requests per Label.

**Throughput:** Throughput is the number of request that are processed per time unit(seconds, minutes, hours) by the server. This time is calculated from the start of first sample to the end of the last sample. Larger throughput is better.

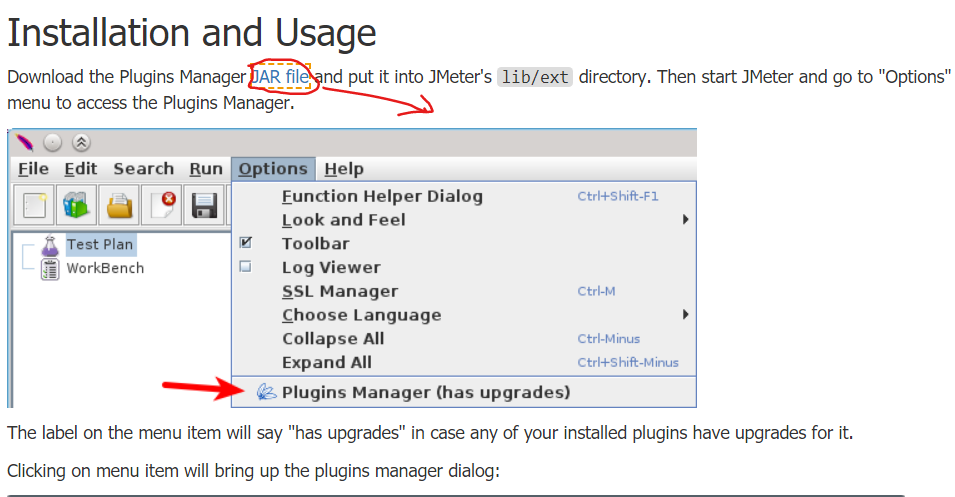
**Median:** It is the time in the middle of a set of samples result. It indicates that 50% of the samples took no more than this time i.e the remainder took at least as long.

***90% Line:*** 90% of the samples took no more than this time. The remaining samples took at least as long as this. (90th percentile)

**Additional Plugins for Simulating real time Load**

Download the jar from:- <https://jmeter-plugins.org/wiki/PluginsManager/>

After landed this below page click on the ***JAR File*** link to download **Plugin Manager**.



Concurrency Thread Group  
Ultimate Thread Group

Created Jmeter script

ThreadGroup – 1

View results tree

200 response Green tick-

200 status code – response may be the wrong one-

AddEmployee –

employee is successfully added 200 status

Employee already exists – 200 stats

3 users

12 15 30

12+15+30/3 = average

Min=12 Max=30

With 100 users of load, through put is 300 Requests/Sec for reserve.php

200 requsts

**Bottomline for Running the selenium scripts**

Because of running *real* browser, WebDriver tests require a lot of resources. In general case, you need 1 CPU core per virtual user with it. & our window cpu is only *4 core* so the rule is (n-1). This makes difficult to scale WebDriver test to have hundreds and thousands of virtual users. However, the load testing cloud providers may help to scale WebDriver test up to thousands of real browsers, look at [BlazeMeter](http://blazemeter.com/?utm_source=jmplinnerpages&utm_medium=cpc&utm_content=jmpininnerpgs&utm_campaign=WebDriverPlugin" \t "_blank) for example.

**Conclusion**

1. JMeter and Selenium are both used for web application testing.
2. JMeter is mostly used to test the performance of web applications under load and stress.
3. Selenium is best suited for automation and cross browser testing of a Website.
4. Based on what type of testing is necessary for the project, one can choose JMeter or Selenium or integrate both if that serves their purpose.

**Performance testing using Selenium webdriver from selenium Documentation**

**(Worst practice)**

Link:- <https://www.selenium.dev/documentation/en/worst_practices/performance_testing/>

Performance testing using Selenium and WebDriver is generally not advised. Not because it is incapable, but because it is not optimised for the job and you are unlikely to get good results.

It may seem ideal to performance test in the context of the user but a suite of WebDriver tests are subjected to many points of external and internal fragility which are beyond your control; for example browser startup speed, speed of HTTP servers, response of third party servers that host JavaScript or CSS, and the instrumentation penalty of the WebDriver implementation itself. Variation at these points will cause variation in your results. It is difficult to separate the difference between the performance of your website and the performance of external resources, and it is also hard to tell what the performance penalty is for using WebDriver in the browser, especially if you are injecting scripts.

The other potential attraction is “saving time” — carrying out functional and performance tests at the same time. However, functional and performance tests have opposing objectives. To test functionality, a tester may need to be patient and wait for loading, but this will cloud the performance testing results and vice versa.

To improve the performance of your website, you will need to be able to analyse overall performance independent of environment differences, identify poor code practices, breakdown of performance of individual resources (i.e. CSS or JavaScript), in order to know what to improve. There are performance testing tools available that can do this job already, that provide reporting and analysis, and can even make improvement suggestions.

Example (open source) packages to use are: JMeter

**Assertion**

When we dealing with the web we mainly use this 3 types of assertion

1. Response assertion
2. Size assertion
3. Duration assertion
4. Html assertion(*rare*)

**How to add project from local Git to remote GitHub**

1. Open folder & r-click & open git bash here & write git init + enter
2. Now .git folder will be created then == > r-click== > git GUI here
3. Then click on remote tab == > click add == > give origin as name &
4. For location give the Repo Url copy from GitHub & select radio btn fetch immediately
5. Then add Unstaged to Staged & write commit message & click commit button.
6. Then click push btn and done.

//Need to import these classes

var support\_ui = JavaImporter(org.openqa.selenium.support.ui.WebDriverWait);

var ui=JavaImporter(org.openqa.selenium.support.ui)

//Set wait time of 5 seconds for our desired object. It can be set in a couple of ways, this is one of them

var wait = new support\_ui.WebDriverWait(WDS.browser, 5000);

WDS.sampleResult.sampleStart()

WDS.browser.get('http://jmeter-plugins.org')

//This is the css path of our object. Again, we can identify the object in a couple of way, cssSelector, classname, id, etc

wait.until(ui.ExpectedConditions.visibilityOfElementLocated(org.openqa.selenium.By.cssSelector('#catalogue > div.list > div:nth-child(1)')))

//additionally check if the title of the page is Jmeter PLug..., if this is not the case, then fail the transaciton

if(WDS.browser.getTitle() != 'JMeter Plugins :: JMeter-Plugins.org') {

    WDS.sampleResult.setSuccessful(false)

}

WDS.sampleResult.sampleEnd()