Performance and Scale Testing

Adhitya Venkatraman Summer 2020 Intern Project

Introduction

Performance and scale (PnS) testing is essential to creating reliable, scalable services for the cloud. This application offers a simple, automated way for developers to test their APIs for performance and scale.

This program, pnstestapp.sh, utilizes JMeter, an open-source testing tool, and runs from the command line. User input and test parameters are fed into the program via .json files that are packaged with the script. By using .json files, users can test several APIs at unique parameters. Each test measures throughput at different thread levels to determine if an API can continue to perform efficiently under increasing stress. Threads simulate users making API calls. Tests increment threads until the throughput either decreases or fails to experience a significant increase (> 5 bytes/second or >10% of previous run). At that point, the test stops and indicates the thread level at which the highest throughput was achieved. Throughout the testing process, live summary statistics for throughput, response time, latency, and errors will be printed so that developers know exactly how well their API is running. When one API has been tested, the tool will proceed to the next one. After each test, a results file is also produced for each run.

Dependencies

There are a few tools that developers must download to run the program, if they do not have them already. Many of these tools may already be installed, but are listed below in case alone with relevant links to read about/download them.

- **Bash**: The script is written in Bash.
- **jq**: A .json parser.
  - https://stedolan.github.io/jq/download/
- **awk**: A command line calculation tool.
  - https://www.gnu.org/software/gawk/manual/html_node/index.html#SEC_Contents
- **bc**: Another calculation tool.
  - https://www.gnu.org/software/bc/
- **JMeter**: The tool that runs these tests.
  - https://jmeter.apache.org
  - The JMeter will install as a folder titled apache-jmeter-"version". Place this folder inside an empty folder titled "**jmeter**". This is done to ensure that the script will continue to perform, even as new versions of JMeter are released. Ensure that the new **jmeter** folder is in the same directory as **pnstestapp.sh**

Files

Along with the script itself, two additional files and one folder are included: Test-Script.jmx, Test-Input.json, and BodyData. This include a .jmx file that is populated by the script and interpreted by JMeter to perform the tests, a .json file that contains basic API and test information, and a folder that contains additional .json files for body data carried by APIs. Because many process are automated by the script, certain file naming conventions must be maintained for the script to work. Also, ensure that the **pnstestapp.sh**, the **Test-Script.jmx** file, the **Test-Input.json** file, the **BodyData** folder, and the **jmeter** folder are all in the same directory.

Of these files, only the contents of Test-Input.json and BodyData (which contains more .json files) should be edited by the user. Again, users avoid changing any file names and follow the above naming convention for BodyData.

pnstestapp.sh	When a user invokes the script, it will immediately begin the testing process. Before invoking the script, the .json files should be populated.
Test-Script.jmx	Users should not make any adjustments to this file. Simply ensure that it is within the same directory as pnstestapp.sh.
Test-Input.json	Users should edit the contents of this file to input their API information and parameters here. The input file may be pre-loaded with a few sample APIs that can be replaced.
BodyData	This folder contains body data for API calls that send information, like those that use a PUSH method. Within the folder, users will create a separate .json file containing the Body Data for each API that needs one. When creating new files, within BodyData, name each new file BodyData-X.json, where X is the index value of the corresponding API within the Test-Input.json, starting with 0. For example, if I am creating a .json file for the body of the first API in Test-Input.json, then the body file will be named BodyData-0.json. Maintain this naming convention to ensure that the each API is matched with the correct body data.

Configuring a Test

Once the script, the supplementary files, and the dependencies are downloaded, users can begin configuring their tests by editing **Test-Input.json** and **BodyData**.

The **Test-Input.json** file includes several different variables and parameters that the user can set. There are 10 key-value pairs that should be filled in for each API. Each API is a separate object. The first six are string values that will be unique to each API:

- testname: A string that the user can set to identify the results of each test when they are completed
- apipath: The API Path
- header: The API header

- token: A token generated that enables one to access the API
- clusterpath: The cluster from which the API calls should be made
- method: The type of HTTP method to be made

The latter four are general test parameters, and have default values. In the descriptions below, the default values are after the semicolon. If the user intends to use the default values, these fields can remain either blank or with the default values. If the user chooses to override the defaults, then the script will proceed using the values specified in **Test-Input.json.**

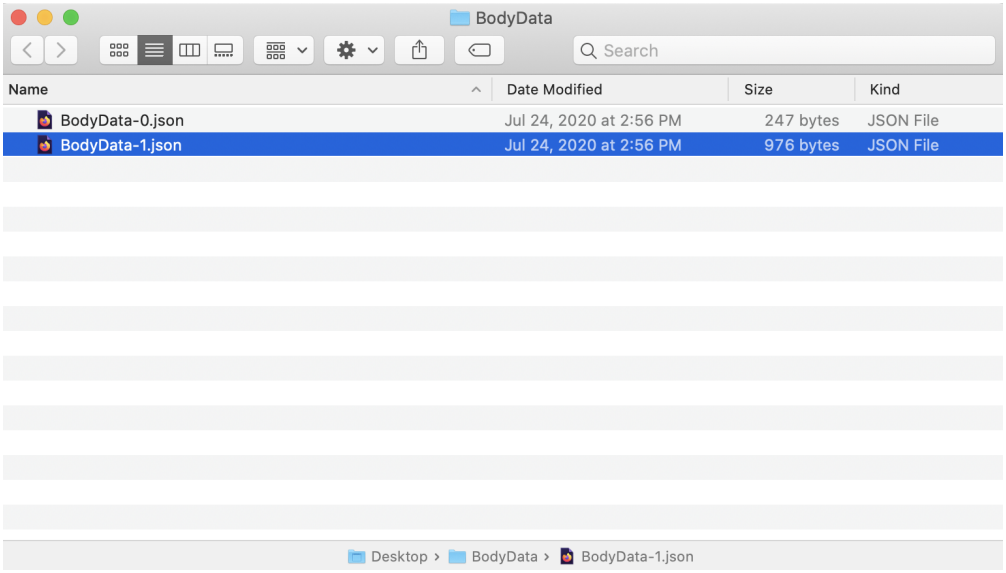
- threads1: An initial thread count value; 5
- threads2: A subsequent thread count value for initial comparison; 10
- duration: Length of the test (seconds); 120
- pods: The multiplier applied to increment the thread count; 2

The below image captures the structure of the **Test-Input.json** file.


```
[
    {
        "testname" : "POST-Test-1",
        "apipath" :
"/api/v1/dxhub/pubsub/publish/dxhubtest2--stream-A-May20",
        "header" : "X-Auth-Token",
        "token" :
"eyJhbGciOiJFUzI1NiIsImtpZCI6IiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiJDRE5BIiwiYXV0
aFNvdXJjZSI6ImlkbS50ZXNzZXJhY3RpbnRlcm5hbC5jb20iLCJjbGllbnRJZCI6ImRhc2hib2
FyZCIsImV4cCI6MTk1NDA5MTcwMiwiaWF0IjoxNTk0MDkxNzAyLCJpc3MiOiJpZG0udGVzc2Vy
YWN0aW50ZXJuYWwuY29tIiwicHIiOiJ1c2VyIiwicm9sZXMiOlsiU1VQRVItQURNSU4iXSwic3
ViIjoiNTA1Yzg1MWMtNTIxZS00ZjAzLTk5NzMtMmFiOTY2ZmRiNmYzIiwidGVuYW50SWQiOiI1
MDVjODUxYy01MjFlLTRmMDMtOTk3My0yYWI5NjZmZGI2ZjMiLCJ0ZW5hbnROYW1lIjoiQ0ROQS
BBZG1pbiIsInVzZXJuYW1lIjoiQ0ROQSBBZG1pbiJ9.Ft5iPlOoLMtzdi98QV-dgJOfcOd-IxF
4ITHo0V-a8FoKlE8l9fCZSkuLYxcUvw4sru-6lWRgCxDznt2D8mwrlg"
,
        "clusterpath" : "maglev.perf-mm-11.tesseractinternal.com",
        "method" : "POST",
        "threads1" : 5,
        "threads2" : 10,
        "duration" : 120,
        "pods" : 2
    },
```


The **BodyData** folder will contain multiple .json files. Because body data is often stored in a .json format, there is generally no additional formatting that needs to be done here.

See below for an example of both the file structure of the **BodyData** folder and an example of what the body data looks like within a BodyData-X.json file.




```
{
  "messages": [
    {
      "messagekey": {
        "SCHEMAID": "e6c07a5a-c4c4-11e9-aa8c-2a2ae2dbcce4"
      },
      "binaryPayload":
"ewogICJmaWVsZDEiOiAidmFsdWUxIiwKICAiZmllbGQyIjogImZpZWxkMiIKfQ==",
      "encodingType": "base64"
    }
  ]
}
```


Running a Test

After the input data is configured, we can begin testing APIs. Below is a walkthrough that demonstrates a complete test through several runs with pictures and some brief commentary of how the program operates.

1. After invoking the program, the test will automatically prompt the user if they wish to proceed with default values for the test. In this case, the user has selected No, so they will proceed with the defaults. Note that in Bash, selection of a menu requires the user to enter the numerical value of their answer in a list, rather than a string of the answer itself. Therefore, to select No, the user enters 2.

```
root@platformqa-loadtest-jmeter-master-57b7b446fc-6rfv5:/# ./pnstestapp.sh
```

```
root@platformqa-loadtest-jmeter-master-57b7b446fc-6rfv5:/# ./pnstestapp.sh
Do you wish to override default thread count (5,10), duration (120), or worker pods (2)?
If so, make changes via the .json input file, then select 1. If not, select No.
1) Yes
2) No
#? 2
You have chosen No.
API being tested: /api/v1/dxhub/pubsub/publish/dxhubtest2--stream-A-May20
```

2. The test will then begin with 5 threads making API calls and will continue for the duration of the test (in this case 120 seconds). Every 30 seconds, the script will produce a live count of the number of calls, time elapsed, average throughput, response time (average, minimum, maximum), and a running error count. At the conclusion of the run, the script will produce relevant summary statistics to measure the performance of the API, such as the mean throughput, as well as the mean, median, 90th, and 95th percentile of latency. While throughput is the key dependent variable measured, the other values provide important measures for the success of the API run.

```
Starting the test @ Tue Jul 28 17:57:36 UTC 2020 (1595959056632)
Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445
summary +   1736 in 00:00:23 =   75.9/s Avg:    61 Min:    15 Max:  3069 Err:     0 (0.00%) Active: 5 Started: 5 Finished: 0
summary +   2376 in 00:00:30 =   79.2/s Avg:    63 Min:    13 Max:  7259 Err:     0 (0.00%) Active: 5 Started: 5 Finished: 0
summary =   4112 in 00:00:53 =   77.8/s Avg:    62 Min:    13 Max:  7259 Err:     0 (0.00%)
summary +   2238 in 00:00:30 =   74.6/s Avg:    66 Min:    13 Max:  3077 Err:     0 (0.00%) Active: 5 Started: 5 Finished: 0
summary =   6350 in 00:01:23 =   76.6/s Avg:    63 Min:    13 Max:  7259 Err:     0 (0.00%)
summary +   2543 in 00:00:30 =   84.8/s Avg:    58 Min:    13 Max:  3052 Err:     0 (0.00%) Active: 5 Started: 5 Finished: 0
summary =   8893 in 00:01:53 =   78.8/s Avg:    62 Min:    13 Max:  7259 Err:     0 (0.00%)
summary +    630 in 00:00:10 =   61.8/s Avg:    66 Min:    14 Max:  3067 Err:     0 (0.00%) Active: 0 Started: 5 Finished: 5
summary =   9523 in 00:02:03 =   77.4/s Avg:    62 Min:    13 Max:  7259 Err:     0 (0.00%)
Tidying up ...    @ Tue Jul 28 17:59:40 UTC 2020 (1595959180215)
... end of run
**************************
Summary of Run
Mean Throughput: 79.7091 bytes/second
Mean Latency: 12.5438 ms
Median Latency: 20 ms
90th Percentile Latency: 26 ms
95th Percentile Latency: 46 ms
**************************
```

3. After completing the first run, the script will proceed to testing at the second default thread level of 10. After completing the run, the script will now produce a list of all the throughput values thus far and compare the most recent ones. This change in throughput, which is displayed for the user, determines if the test will continue. If the change is negative, the test will stop and declare that the second-most recent run was the optimal level. If the change is positive, but insignificant (below 5 bytes/second or 10% of the previous throughput) the test will halt and declare the most recent run to have the optimal thread level. These cutoffs were determined through conversations and guidance from QA engineers. If the throughput is both positive and significant, then the test will continue.

```
Starting the test @ Tue Jul 28 18:00:11 UTC 2020 (1595959211934)
Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445
summary +   1545 in 00:00:18 =   88.3/s Avg:   105 Min:    15 Max:  3065 Err:     0 (0.00%) Active: 10 Started: 10 Finished: 0
summary +   2801 in 00:00:30 =   93.3/s Avg:   107 Min:    14 Max:  3075 Err:     0 (0.00%) Active: 10 Started: 10 Finished: 0
summary =   4346 in 00:00:48 =   91.5/s Avg:   106 Min:    14 Max:  3075 Err:     0 (0.00%)
summary +   2944 in 00:00:30 =   98.1/s Avg:    98 Min:    12 Max:  3088 Err:     0 (0.00%) Active: 10 Started: 10 Finished: 0
summary =   7290 in 00:01:18 =   94.1/s Avg:   103 Min:    12 Max:  3088 Err:     0 (0.00%)
summary +   3796 in 00:00:30 =  126.5/s Avg:    80 Min:    12 Max:  7295 Err:     0 (0.00%) Active: 10 Started: 10 Finished: 0
summary =  11086 in 00:01:48 =  103.1/s Avg:    95 Min:    12 Max:  7295 Err:     0 (0.00%)
summary +   1523 in 00:00:15 =   98.5/s Avg:    89 Min:    13 Max:  3076 Err:     0 (0.00%) Active: 0 Started: 10 Finished: 10
summary =  12609 in 00:02:03 =  102.5/s Avg:    94 Min:    12 Max:  7295 Err:     0 (0.00%)
Tidying up ...    @ Tue Jul 28 18:02:15 UTC 2020 (1595959335481)
... end of run
**************************
Summary of Run
Mean Throughput: 105.407 bytes/second
Mean Latency: 9.48623 ms
Median Latency: 21 ms
90th Percentile Latency: 29 ms
95th Percentile Latency: 1028 ms
**************************
Throughput For Completed Runs:
79.7091 bytes/second
105.407 bytes/second
Comparing Throughputs...
Change in Throughput: 25.6979 bytes/second
**************************
```

4. At this point, the thread count will begin increasing by the pods multiplier. Because the default multiplier is 2, the test will now test the API at 20 threads.

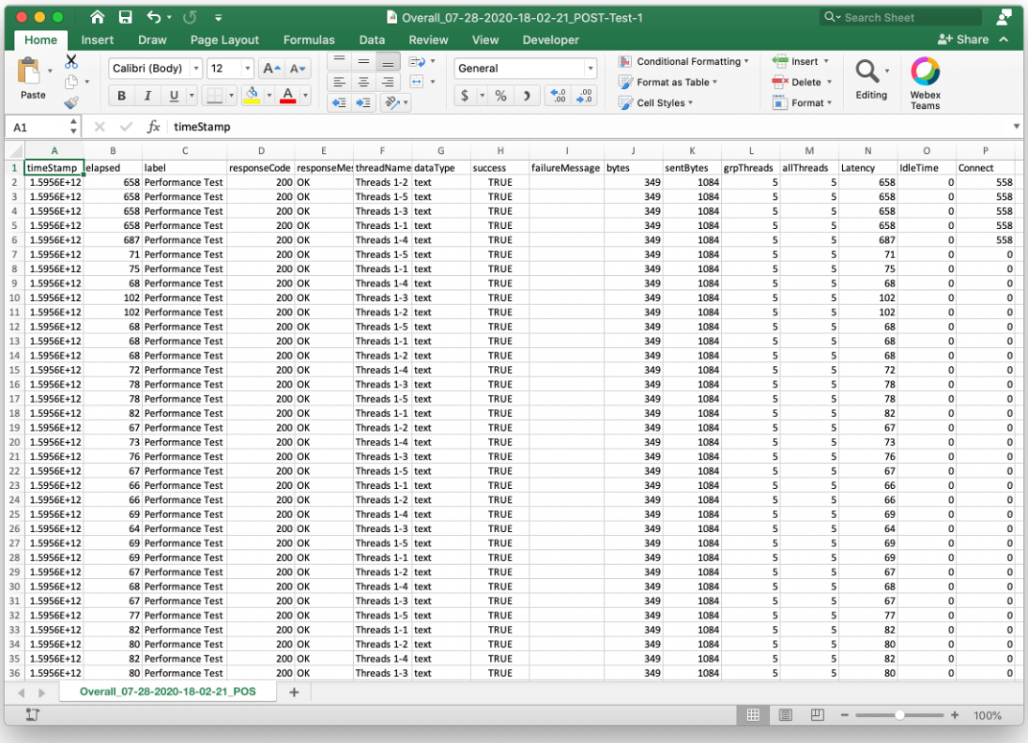
```
Starting the test @ Tue Jul 28 18:02:52 UTC 2020 (1595959372371)
Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445
summary +   1636 in 00:00:07 =  228.8/s Avg:    71 Min:    15 Max:  3124 Err:     0 (0.00%) Active: 20 Started: 20 Finished: 0
summary +   4335 in 00:00:30 =  144.5/s Avg:   136 Min:    12 Max:  7308 Err:     0 (0.00%) Active: 20 Started: 20 Finished: 0
summary =   5971 in 00:00:37 =  160.7/s Avg:   118 Min:    12 Max:  7308 Err:     0 (0.00%)
summary +   5986 in 00:00:30 =  199.5/s Avg:   101 Min:    12 Max:  7265 Err:     0 (0.00%) Active: 20 Started: 20 Finished: 0
summary =  11957 in 00:01:07 =  178.1/s Avg:   110 Min:    12 Max:  7308 Err:     0 (0.00%)
summary +   5949 in 00:00:30 =  198.3/s Avg:    97 Min:    13 Max:  7361 Err:     0 (0.00%) Active: 20 Started: 20 Finished: 0
summary =  17906 in 00:01:37 =  184.3/s Avg:   105 Min:    12 Max:  7361 Err:     0 (0.00%)
summary +   5062 in 00:00:26 =  197.2/s Avg:    98 Min:    12 Max:  7204 Err:     0 (0.00%) Active: 0 Started: 20 Finished: 20
summary =  22968 in 00:02:03 =  187.0/s Avg:   104 Min:    12 Max:  7361 Err:     0 (0.00%)
Tidying up ...    @ Tue Jul 28 18:04:55 UTC 2020 (1595959495667)
... end of run
**************************
Summary of Run
Mean Throughput: 191.904 bytes/second
Mean Latency: 5.21048 ms
Median Latency: 21 ms
90th Percentile Latency: 38 ms
95th Percentile Latency: 1029 ms
**************************
Throughput For Completed Runs:
79.7091 bytes/second
105.407 bytes/second
191.904 bytes/second
Comparing to previous throughputs...
Change in Throughput: 86.497 bytes/sec
**************************
```

5. Eventually, the test will place sufficient stress on the API that it will no longer operate efficiently. At this point, the change in throughput will likely be less than 5 bytes/second or 10% of the previous run. In the example below, both of these conditions are met. Thus, the test is concluded, and the script indicates to the user that their API can handle 160 threads.
   An important note is that the test will halt always halt at or before 200 threads. This is because the focus of the tool is to determine if the API can handle a certain threshold of stress, rather than to determine its breaking point. If an API can withstand 200 threads, then it effectively passes the performance and scale test. For this reason, under the default parameters, the test will always halt at 160, regardless of if the API can handle greater load. Even if other default thread counts are used, the test will always halt at or before 200.

```
Starting the test @ Tue Jul 28 18:10:24 UTC 2020 (1595959824210)
Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445
summary +   1302 in 00:00:05 =  245.0/s Avg:   457 Min:    29 Max:  3396 Err:     0 (0.00%) Active: 160 Started: 160 Finished: 0
summary +  12361 in 00:00:30 =  412.0/s Avg:   375 Min:    17 Max: 31672 Err:     0 (0.00%) Active: 160 Started: 160 Finished: 0
summary =  13663 in 00:00:35 =  386.8/s Avg:   383 Min:    17 Max: 31672 Err:     0 (0.00%)
summary +   9293 in 00:00:30 =  309.8/s Avg:   486 Min:    19 Max: 31827 Err:     0 (0.00%) Active: 160 Started: 160 Finished: 0
summary =  22956 in 00:01:05 =  351.5/s Avg:   424 Min:    17 Max: 31827 Err:     0 (0.00%)
summary +  14373 in 00:00:30 =  479.1/s Avg:   357 Min:    19 Max: 66030 Err:     0 (0.00%) Active: 160 Started: 160 Finished: 0
summary =  37329 in 00:01:35 =  391.7/s Avg:   399 Min:    17 Max: 66030 Err:     0 (0.00%)
summary +  12037 in 00:00:30 =  395.7/s Avg:   340 Min:    21 Max: 15737 Err:     0 (0.00%) Active: 13 Started: 160 Finished: 147
summary =  49366 in 00:02:06 =  392.6/s Avg:   384 Min:    17 Max: 66030 Err:     0 (0.00%)
summary +     12 in 00:00:09 =    1.3/s Avg: 19782 Min:  7312 Max: 131238 Err:     0 (0.00%) Active: 0 Started: 160 Finished: 160
summary =  49378 in 00:02:15 =  365.9/s Avg:   389 Min:    17 Max: 131238 Err:     0 (0.00%)
Tidying up ...    @ Tue Jul 28 18:12:39 UTC 2020 (1595959959648)
... end of run
**************************
Summary of Run
Mean Throughput: 410.869 bytes/second
Mean Latency: 2.43376 ms
Median Latency: 209 ms
90th Percentile Latency: 1042 ms
95th Percentile Latency: 1237 ms
**************************
Throughput For Completed Runs:
79.7091 bytes/second
105.407 bytes/second
191.904 bytes/second
242.434 bytes/second
406.179 bytes/second
410.869 bytes/second
Comparing to previous throughputs...
Change in Throughput: 4.690 bytes/sec
Testing complete. Optimal Thread Count:  160
```

Post-Test Results

A new "results" folder is created in the same directory that the script is called from. Several comprehensive results files will populate this folder as the tests progress. The testname variable assigned by the user in the **Test-Inputs.json** file will be crucial to distinguishing these files. Each result file name will include the date, time, and API test name. In addition to the summary statistics and results that are displayed live during each test, JMeter will produce these results as .csv files that contain raw data for every single call that was made. Each of these results files are populated live during the test, and are complete as soon as the test is finished. Some data included is the elapsed time, HTTP response code, sent bytes, latency, and connect time for each call. This information will be relevant for any user who wants to further explore their results or is interested in producing additional statistics or conducting further analysis of their tests. To do so within the Bash terminal, awk is recommended to parse the results files or fill new columns with additional calculations. Below is a sample results file.



Overall_07-28-2020-18-02-21_POST-Test-1

timeStamp	elapsed	label	responseCode	responseMe	threadName	dataType	success	failureMessage	bytes	sentBytes	grpThreads	allThreads	Latency	IdleTime	Connect
1.5956E+12	658	Performance Test	200	OK	Threads 1-2	text	TRUE		349	1084	5	5	658	0	558
1.5956E+12	658	Performance Test	200	OK	Threads 1-5	text	TRUE		349	1084	5	5	658	0	558
1.5956E+12	658	Performance Test	200	OK	Threads 1-3	text	TRUE		349	1084	5	5	658	0	558
1.5956E+12	658	Performance Test	200	OK	Threads 1-1	text	TRUE		349	1084	5	5	658	0	558
1.5956E+12	687	Performance Test	200	OK	Threads 1-4	text	TRUE		349	1084	5	5	687	0	558
1.5956E+12	71	Performance Test	200	OK	Threads 1-5	text	TRUE		349	1084	5	5	71	0	0
1.5956E+12	75	Performance Test	200	OK	Threads 1-1	text	TRUE		349	1084	5	5	75	0	0
1.5956E+12	68	Performance Test	200	OK	Threads 1-4	text	TRUE		349	1084	5	5	68	0	0
1.5956E+12	102	Performance Test	200	OK	Threads 1-3	text	TRUE		349	1084	5	5	102	0	0
1.5956E+12	102	Performance Test	200	OK	Threads 1-2	text	TRUE		349	1084	5	5	102	0	0
1.5956E+12	68	Performance Test	200	OK	Threads 1-5	text	TRUE		349	1084	5	5	68	0	0
1.5956E+12	68	Performance Test	200	OK	Threads 1-1	text	TRUE		349	1084	5	5	68	0	0
1.5956E+12	68	Performance Test	200	OK	Threads 1-2	text	TRUE		349	1084	5	5	68	0	0
1.5956E+12	72	Performance Test	200	OK	Threads 1-4	text	TRUE		349	1084	5	5	72	0	0
1.5956E+12	78	Performance Test	200	OK	Threads 1-3	text	TRUE		349	1084	5	5	78	0	0
1.5956E+12	78	Performance Test	200	OK	Threads 1-5	text	TRUE		349	1084	5	5	78	0	0
1.5956E+12	82	Performance Test	200	OK	Threads 1-1	text	TRUE		349	1084	5	5	82	0	0
1.5956E+12	67	Performance Test	200	OK	Threads 1-2	text	TRUE		349	1084	5	5	67	0	0
1.5956E+12	73	Performance Test	200	OK	Threads 1-4	text	TRUE		349	1084	5	5	73	0	0
1.5956E+12	76	Performance Test	200	OK	Threads 1-3	text	TRUE		349	1084	5	5	76	0	0
1.5956E+12	67	Performance Test	200	OK	Threads 1-5	text	TRUE		349	1084	5	5	67	0	0
1.5956E+12	66	Performance Test	200	OK	Threads 1-1	text	TRUE		349	1084	5	5	66	0	0
1.5956E+12	66	Performance Test	200	OK	Threads 1-2	text	TRUE		349	1084	5	5	66	0	0
1.5956E+12	69	Performance Test	200	OK	Threads 1-4	text	TRUE		349	1084	5	5	69	0	0
1.5956E+12	64	Performance Test	200	OK	Threads 1-3	text	TRUE		349	1084	5	5	64	0	0
1.5956E+12	69	Performance Test	200	OK	Threads 1-5	text	TRUE		349	1084	5	5	69	0	0
1.5956E+12	69	Performance Test	200	OK	Threads 1-1	text	TRUE		349	1084	5	5	69	0	0
1.5956E+12	67	Performance Test	200	OK	Threads 1-2	text	TRUE		349	1084	5	5	67	0	0
1.5956E+12	68	Performance Test	200	OK	Threads 1-4	text	TRUE		349	1084	5	5	68	0	0
1.5956E+12	67	Performance Test	200	OK	Threads 1-3	text	TRUE		349	1084	5	5	67	0	0
1.5956E+12	77	Performance Test	200	OK	Threads 1-5	text	TRUE		349	1084	5	5	77	0	0
1.5956E+12	82	Performance Test	200	OK	Threads 1-1	text	TRUE		349	1084	5	5	82	0	0
1.5956E+12	80	Performance Test	200	OK	Threads 1-2	text	TRUE		349	1084	5	5	80	0	0
1.5956E+12	82	Performance Test	200	OK	Threads 1-4	text	TRUE		349	1084	5	5	82	0	0
1.5956E+12	80	Performance Test	200	OK	Threads 1-3	text	TRUE		349	1084	5	5	80	0	0

For additional questions about this tool, contact avenkatraman22@cmc.edu.