Characterizing the Behavior of Website Developed using Node.js

Sudeep Pereje
Electrical and Computer Engineering
Department
University of Calgary
sudeep.pereje@ucalgary.ca

I. ABSTRACT

In the real-world scenario, the performance of the webserver developed under a scripting language must be efficient as it provides an enriching experience towards the end-user. The request made by the user to the response achieved by the user plays a vital role to characterize the performance of the webserver. Many real-world applications and websites are being developed under the belt of some powerful scripting languages and their performances are rigorously tested under various configurations. This project discusses about the characteristics and behavior of the website developed using node.js scripting language and analyzed its performance with the help of a performance testing tool and a proper testbed configuration to provide a better web page browsing experience for an end-user. The experimental results show that the webserver has achieved an average response time of 0.12 seconds and average throughput of 21,764 user requests per second.

II. INTRODUCTION

Many websites are currently being developed using different scripting languages to provide user a hassle-free experience. To provide this rich experience, these web applications and web servers are tested with different software and hardware configurations to analyze its performance and its capacity of serving end-users. This type of testing is called as performance testing. Usually this testing is done on a proper setup with high end software and hardware configurations. In this project we will deploy a working node.js website on the cloud and we will test the performance of the website with the help of a load testing tool.

Node.js: Node.js is an open-source, cross platform, JavaScript runtime environment that executes the JavaScript code outside of a browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting. It represents a "JavaScript everywhere" paradigm, unifying web application development around a single programming language, rather than different languages for server and client-side scripts. Node.js operates on a single threaded event loop, using non-blocking I/O calls, allowing it to support tens of thousands of concurrent connections without incurring the cost of thread context switching. To accommodate the single-threaded event loop, it uses the libuv library which in turn, uses a fixed-sized thread pool that handles some of the non-blocking asynchronous I/O operations.

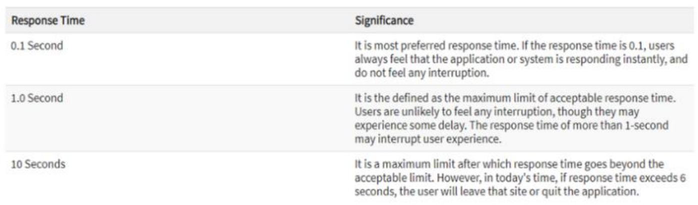
Performance Testing: Performance testing is the process of determining the speed, responsiveness and stability of a software program under a specific workload to identify the bottleneck within a system. Performance testing can involve quantitative tests done in a lab or occur in the production environment in limited scenarios. Performance testing can help identify the nature or location of a software-related performance problem by highlighting where an application might fail or lag. Some of the important metrics in performance testing include Response time and Throughput.

Types of Performance Testing: There are two main performance testing methods: Load Testing and Stress Testing.

Load Testing: Load Testing helps developers understand the behavior of a system under a specific load value. In the load testing process, we can simulate the expected number of concurrent users and transactions over a duration of time to verify expected response times and locate bottlenecks. This type of test help developers determines how many users an application or system can handle before that app or system goes live.

Stress Testing: Stress tests enables to understand a workload's scalability. Stress tests put strain on hardware resources, such as CPU's, memory, hard disks and solid-state drives, to determine the potential breaking point of an application on these finite resources. System strain can also lead to slow data exchanges, memory shortages, data corruption and security issues. Stress tests can occur before or after a system goes live.

Response Time: The time difference between the submission of a request until the response begins to be received is called as Response time. The response time should be as low as possible so that many interactive users can receive an acceptable response time. Some of the important metrics for the response time are Average Response time, Peak Response time and Error Rate.



Response Time	Significance
0.1 Second	It is most preferred response time. If the response time is 0.1, users always feel that the application or system is responding instantly, and do not feel any interruption.
1.0 Second	It is the defined as the maximum limit of acceptable response time. Users are unlikely to feel any interruption, though they may experience some delay. The response time of more than 1-second may interrupt user experience.
10 Seconds	It is a maximum limit after which response time goes beyond the acceptable limit. However, in today's time, if response time exceeds 6 seconds, the user will leave that site or quit the application.

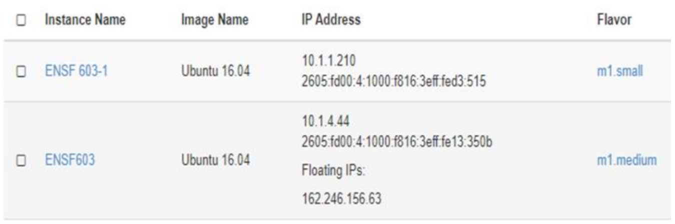
Figure 1: Response times and its significance

Throughput: The number of processes that are completed per unit time is called the Throughput of that system. The higher the Throughput the better is the server's performance and the lower the deviation the better the performance.

Apache JMeter: JMeter is an Apache performance testing tool that can generate load tests on web and application services. JMeter plugins provide flexibility in load testing and cover areas such as graphs, thread groups, timers, functions and logic controllers. JMeter supports an integrated development environment for test recording for browsers or web applications as well as command-line mode for load testing Java-based operating systems.

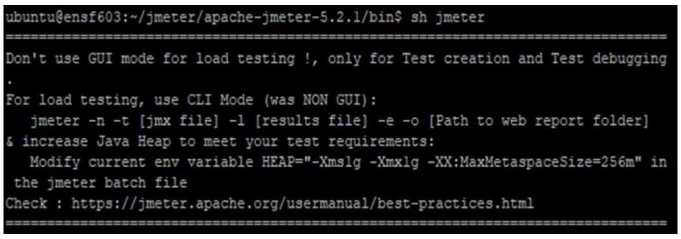
III. EXPERIMENTAL DESIGN

In the experimental setup we first deploy our Node.js website on the cloud. The cloud platform we used in this project is Rapid Access Cloud. In Rapid Access cloud we have created multiple instances with flavors m1.small and m1.medium. The Small instance is configured with 2 CPU's, 2GB of RAM and Disk space of 20GB. The Medium instance is configured with 4 CPU's, 4GB of RAM and Disk space of 40GB. Both the instances are configured with the virtual image of Ubuntu 16.04 running on them. The Apache JMeter is installed in the m1.medium instance and it is connected to m1.small instance as a Master to Slave Configuration. The entire setup is connected to a High-Speed Internet connection (LAN) with a speed of 1GBPS with no noisy neighbors.



☐	Instance Name	Image Name	IP Address	Flavor
☐	ENSF 603-1	Ubuntu 16.04	10.1.1.210 2605:fd00:4:1000:f816:3eff:fed3:515	m1.small
☐	ENSF603	Ubuntu 16.04	10.1.4.44 2605:fd00:4:1000:f816:3eff:fe13:350b Floating IPs: 162.246.156.63	m1.medium

Figure 2: Instances Configuration



```
ubuntu@ensf603:~/jmeter/apache-jmeter-5.2.1/bin$ sh jmeter
================================================================================
Don't use GUI mode for load testing !, only for Test creation and Test debugging
.
For load testing, use CLI Mode (was NON GUI):
   jmeter -n -t [jmx file] -l [results file] -e -o [Path to web report folder]
& increase Java Heap to meet your test requirements:
   Modify current env variable HEAP="-Xms1g -Xmx1g -XX:MaxMetaspaceSize=256m" in
 the jmeter batch file
Check : https://jmeter.apache.org/usermanual/best-practices.html
================================================================================
```

Figure 3:Apache JMeter installed on Instance m1.medium

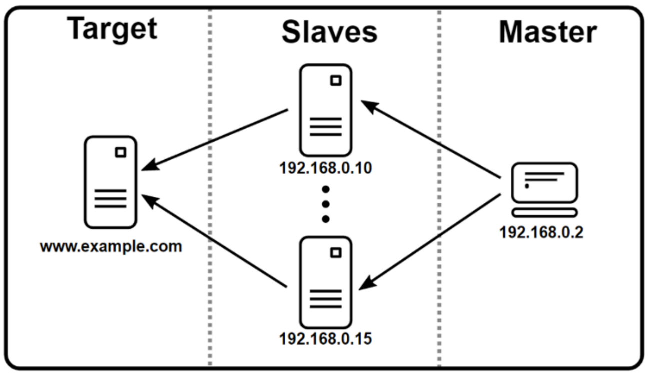


Figure 4:An Overview of Master-Slave Configuration

IV. EXPERIMENT

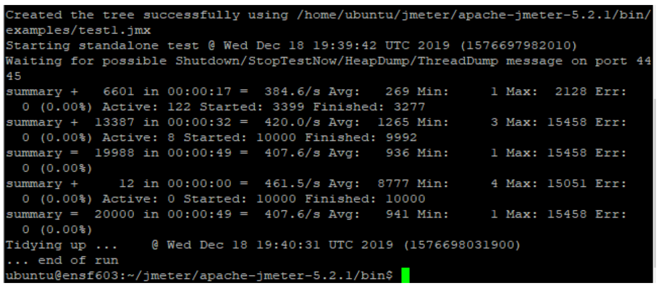
Objectives and Research Questions:

Our main objective is to perform load test on the Node.js webserver using Apache JMeter performance testing tool and analyse its response time and throughput.

- **RQ:** What could be the best Response Time and Throughput achieved by the Node.js webserver?

V. EXPERIMENTAL RESULTS

Below are the experimental results obtained after performing the Apache JMeter load testing on node.js webserver.



```
Created the tree successfully using /home/ubuntu/jmeter/apache-jmeter-5.2.1/bin/
examples/test1.jmx
Starting standalone test @ Wed Dec 18 19:39:42 UTC 2019 (1576697982010)
Waiting for possible Shutdown/StopTestNow/HeapDump/ThreadDump message on port 44
45
summary +   6601 in 00:00:17 =  384.6/s Avg:   269 Min:     1 Max:  2128 Err:
  0 (0.00%) Active: 122 Started: 3399 Finished: 3277
summary +  13387 in 00:00:32 =  420.0/s Avg:  1265 Min:     3 Max: 15458 Err:
  0 (0.00%) Active: 8 Started: 10000 Finished: 9992
summary =  19988 in 00:00:49 =  407.6/s Avg:   936 Min:     1 Max: 15458 Err:
  0 (0.00%)
summary +     12 in 00:00:00 =  461.5/s Avg:  8777 Min:     4 Max: 15051 Err:
  0 (0.00%) Active: 0 Started: 10000 Finished: 10000
summary =  20000 in 00:00:49 =  407.6/s Avg:   941 Min:     1 Max: 15458 Err:
  0 (0.00%)
Tidying up ...    @ Wed Dec 18 19:40:31 UTC 2019 (1576698031900)
... end of run
ubuntu@ensf603:~/jmeter/apache-jmeter-5.2.1/bin$
```

Figure 5: Running JMeter Script on Medium Instance through Master-Slave Configuration

The aggregated Response time results for the webserver for 200 concurrent users with 2 iterations and a total of 400 samples is 0.3 seconds on an average for around 10 JMeter Script executions.

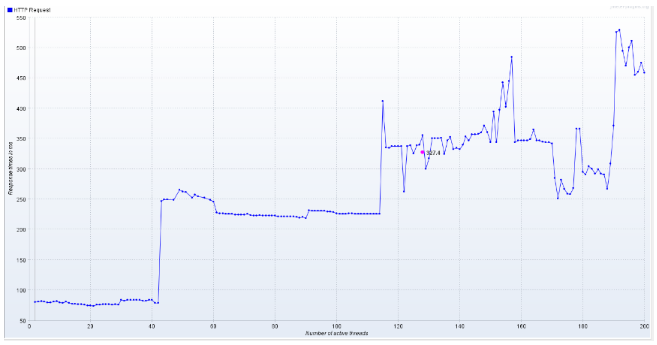


Figure 6 Aggregated Response time with 200 users and 400 samples

The aggregated Response time results for the webserver for 500 concurrent users with 2 iterations and a total of 1000 samples is 0.11 seconds on an average for around 10 JMeter Script executions.

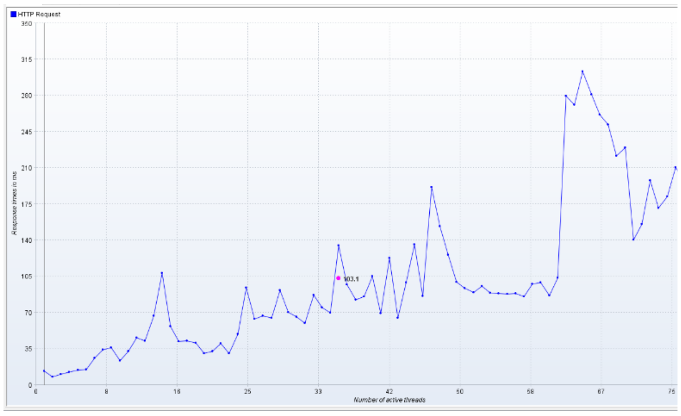


Figure 7: Aggregated Response time with 500 users and 1000 Samples

The aggregated Response time results for the webserver for 1000 concurrent users with 2 iterations and a total of 2000 samples is 0.12 seconds on an average for around 10 JMeter Script executions.

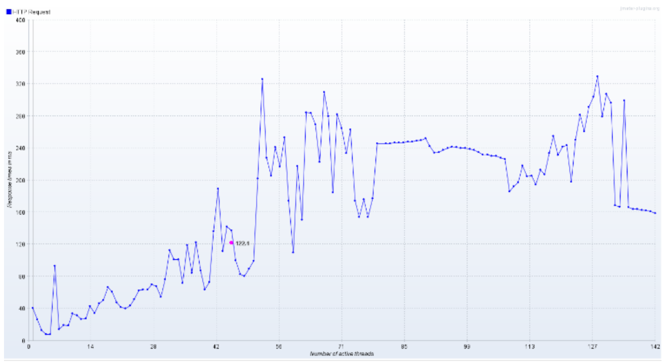


Figure 8: Aggregated Response time with 1000 users and 2000 samples

The aggregated Throughput results for the webserver for 200 concurrent users with 2 iterations and a total of 400 samples is 22,222 requests per second on an average for around 10 JMeter Script executions with less deviation of 129 samples

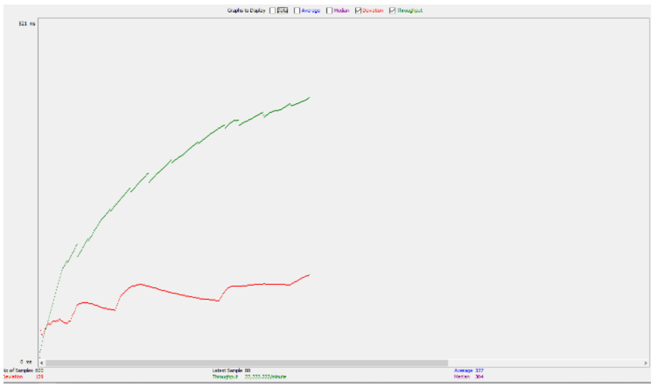


Figure 9: Aggregated throughput with 200 users and 400 samples

The aggregated Throughput results for the webserver for 500 concurrent users with 2 iterations and a total of 1000 samples is 21,149 requests per second on an average for around 10 JMeter Script executions with less deviation of 97 samples.

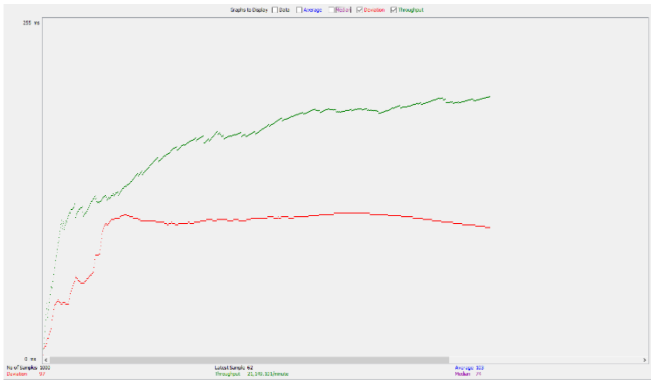


Figure 10: Aggregated throughput with 500 users and 1000 samples

The aggregated Throughput results for the webserver for 1000 concurrent users with 2 iterations and a total of 2000 samples is 21,921 requests per second on an average for around 10 JMeter Script executions with less deviation of 45 samples.

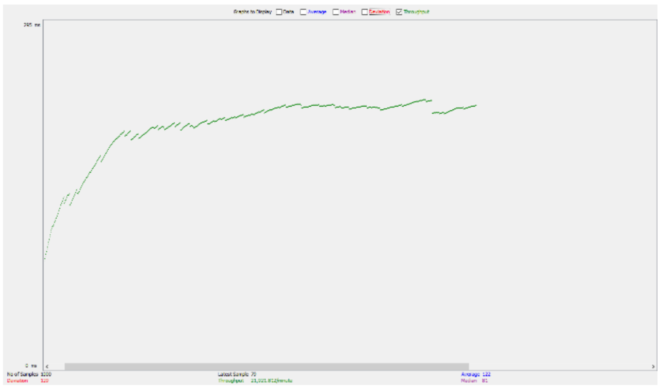


Figure 11: Aggregated throughput with 1000 users and 2000 samples

RQ Result: What could be the best Response Time and Throughput achieved by the Node.js webserver?

By analysing the aggregated results generated from the Section-V, we can say that the Average Response time for 400 Samples is 0.3 seconds, for 1000 samples is 0.1 seconds and for 2000 samples is 0.1 seconds for two iterations and 10 JMeter Script runs. So, from these results we can say that the best response time that a user can get by accessing this website which is developed using Node.js is **0.13** seconds which is mentioned to be the best Response Time according to the current end-user experience statistics according to Figure 1 in Section-II.

By analysing the aggregated results generated from the Section-V, we can say that the Average Throughput for 400 samples is 22,222 requests per second, for 1000 samples is 21,149 requests per second, for 2000 samples is 21,921 requests per second. So, from these results we can say that the best throughput that the server can handle for a given number of users is **21,764** requests per second when users are browsing concurrently and **23,467** requests per second when the users are spending time in ramp up period i.e. Think time.

VI. CONCLUSION

In the real world, several software organizations are testing their applications on various test bed environments so that the applications can perform well without any bottlenecks and provide seamless experience to the end user. This proves the need for performance testing of application before deploying it to the market for end-user experience. In this project, we have executed the load testing on the webserver that is developed using Node.js scripting language using Apache JMeter tool with a possible Software and Hardware configurations. The webserver has achieved a Response Time of **0.13** seconds which tends to be the best Response Time according to end user's experience statistics in Section-II figure-1 and a Throughput of **21,764** requests per second. Both Response Time and Throughput of the webserver shows that the webserver, even if it has a single threaded architecture because of Node.js has provided a peak performance on the test bed configuration for at most 10 JMeter script executions.

VII. REFERENCES

[1] "RAPID ACCESS CLOUD," [Online]. Available: https://cloud.cybera.ca/auth/login/

[2] "Apache JMeter," [Online]. Available: https://jmeter.apache.org/download_jmeter.cgi.

[3] "Response time limits and their significance" https://www.nngroup.com/articles/response-times-3-important-limits/

[4] " Research the performance testing and performance improvement strategy in web application-Kunhua Zhu, Junhui Fu, Yancui Li- ICETC-2010.

[5] " Performance testing of the web applications based on test case management" – Rijwan Khan, Mohd Amjad – https://www.sciencedirect.com/science/article/pii/S2213020916300957?via%3Dihub#bib0005