

Performance Analysis of Bodhitree Application

CS691 - RnD Report

Vadapalli K Chaitanya Varma - 203050026

Bandapalli Saikumar - 203050095

Under the guidance of

Prof. Varsha Apte

June 28, 2022



Department of Computer Science and Engineering
Indian Institute of Technology, Bombay

Contents

1	Introduction	5
2	JMeter Test Plan	6
3	Server side scripts	17
4	Run load test	18
5	Results	20
5.1	Intermediate results	20
5.2	Final results	22
6	Conclusion	24
7	Acknowledgement	24
8	References	25

List of Figures

1	Baseline Load Test Plan	6
2	User defined variables	7
3	CSV Data Set Configuration	7
4	Extracting CSRF token using Regular expressions	8
5	Student Login	8
6	Extracting Course ID using Regular expressions	9
7	Access course page	9
8	Extracting Assignment ID using Regular expressions	10
9	Get Solutions folder	10
10	Pick a Random Solution file	11
11	Uploading solution file	11
12	Extracting submission id	12
13	Run Practice Test Cases	12
14	While Controller to loop test case status	13
15	Test Cases Status Request	13
16	Exclude Auto refresh requests from overall throughput	14
17	Test Case Results page	14
18	Practice test case Response time calculator	15
19	Extract Compare IDs using Regular Expressions	15
20	ForEach Controller to compare test case outputs	16
21	Compare Outputs	16
22	Load Test Code	19
23	Intermediate - Response time Vs Number of Users	20
24	Intermediate - Throughput Vs Number of Users	21
25	Intermediate - CPU Utilization Vs Number of Users	21
26	Final - Response time Vs Number of Users	22
27	Final - Throughput Vs Number of Users	23
28	Final - CPU Utilization Vs Number of Users	23

Abstract

As a part of this project we have performed load test on Bodhi-tree eval pro system ,to find what is the maximum no. of users(Saturation number) the system supports,maximum throughput, CPU utilization when no.of users are beyond saturation point,slope of the response time graph when it becomes linear.

We have developed a JMeter script which will simulate load test for the Bodhi-tree eval pro features which include 1. Login 2. Course page 3. Upload assignment 4. Run practice test 5. Compare failed test cases.We have automated the load test using SSH Pass.After load test is completed we will get summary of metrics in a folder with the name including the time stamp, when load test is performed.

1 Introduction

Bodhi-tree is an application, which provides course management features, in which students can login, view course page for announcements, upload assignments, check their performance by running practice tests and also compare their output with actual output. Instructors can upload course content, manage auto grading programming assignments and carry out typical tasks of running a course.

JMeter is an open source software which is designed to load test applications and measure performance. It is a multi-threading framework which allows concurrent sampling of multiple threads. It includes a wide variety of features which help to simulate complex features of an application, from load test. It also generates metrics in a HTML report.

We have developed a JMeter script for following Bodhi-tree eval pro features 1. Login 2. Course page 3. Upload assignment 4. Run practice test 5. Compare failed test cases and performed our experiments in two phases

1. Intermediate phase

In this phase, we only simulated the load test from login to accessing course page. Load test is run for 5 minutes.

2. Final phase

In this phase, we simulated load test for all the five features. Load test is run for 15 minutes

We have automated load test, using ssh pass by which load test will run automatically without manual effort. We developed scripts which will collect snapshots for metrics while the load test is happening. After load test is completed, a post processing script will read the snapshots and generate performance metrics summary for the load test.

The next section will discuss about our JMeter test plan, how we have simulated different Bodhi-tree eval pro features by using JMeter

2 JMeter Test Plan

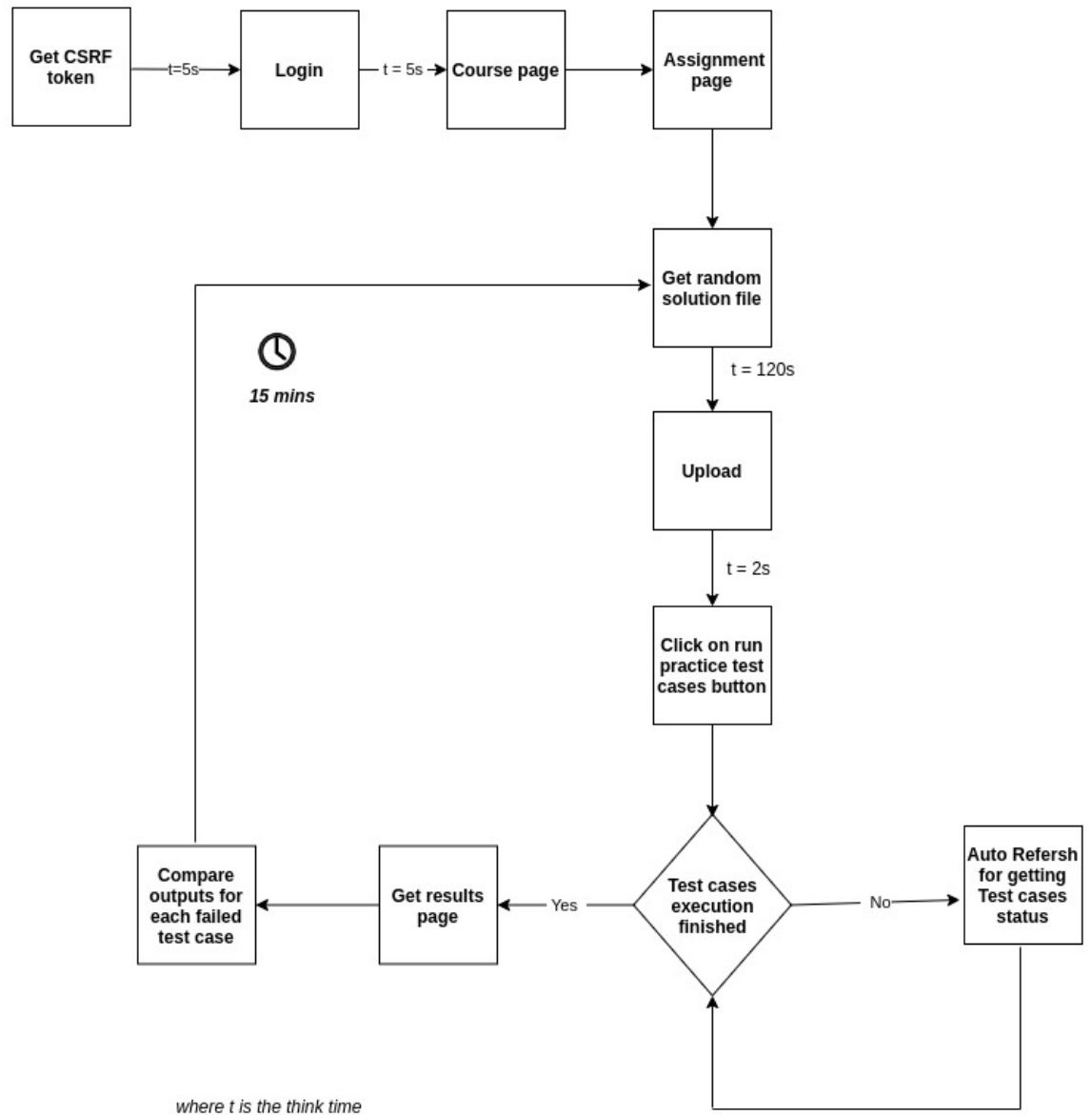
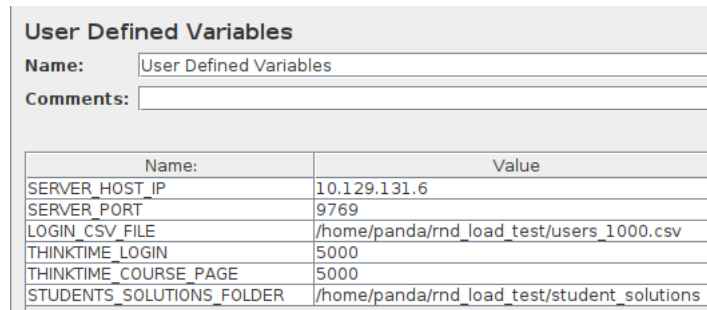


Figure 1: Baseline Load Test Plan

1. Creating user defined variables

For Creating user defined variables we added User Defined Variables JMeter element, As shown in Figure 2, key value pairs are added, where key is the JMeter variable name



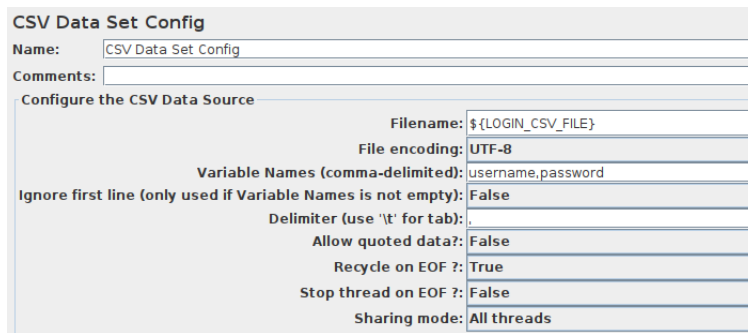
The screenshot shows the 'User Defined Variables' configuration window. It has a 'Name' field with the value 'User Defined Variables' and an empty 'Comments' field. Below these fields is a table with two columns: 'Name' and 'Value'.

Name	Value
SERVER_HOST_IP	10.129.131.6
SERVER_PORT	9769
LOGIN_CSV_FILE	/home/panda/rnd_load_test/users_1000.csv
THINKTIME_LOGIN	5000
THINKTIME_COURSE_PAGE	5000
STUDENTS_SOLUTIONS_FOLDER	/home/panda/rnd_load_test/student_solutions

Figure 2: User defined variables

2. Get the username and password from csv file

For reading username, password from a csv file, we added CSV Data Set Config JMeter element. As shown in Figure 3, it will read username, password into JMeter variables, csv file will be shared among multiple threads.



The screenshot shows the 'CSV Data Set Config' configuration window. It has a 'Name' field with the value 'CSV Data Set Config' and an empty 'Comments' field. Below these fields is a section titled 'Configure the CSV Data Source' with several configuration options:

- Filename: \${LOGIN_CSV_FILE}
- File encoding: UTF-8
- Variable Names (comma-delimited): username,password
- Ignore first line (only used if Variable Names is not empty): False
- Delimiter (use '\t' for tab): ,
- Allow quoted data?: False
- Recycle on EOF?: True
- Stop thread on EOF?: False
- Sharing mode: All threads

Figure 3: CSV Data Set Configuration

3. Get CSRF token

For extracting CSRF token from HTTP response, we used Regular Expression Extractor JMeter element. As shown in Figure 4, CSRF token is extracted into a JMeter variable by using regular expression.

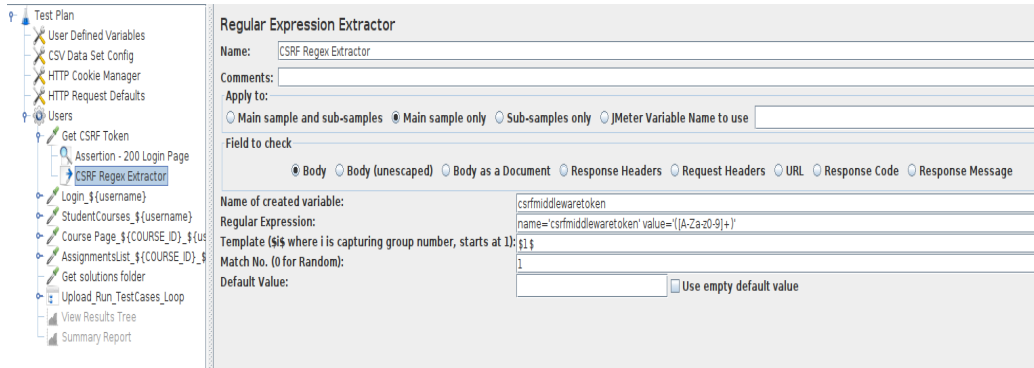


Figure 4: Extracting CSRF token using Regular expressions

4. Student login

Student Login is simulated from load test using HTTP Request JMeter element. As shown in Figure 5, login API is being called using HTTP post request, which sends username, password and CSRF token in the URL as parameters

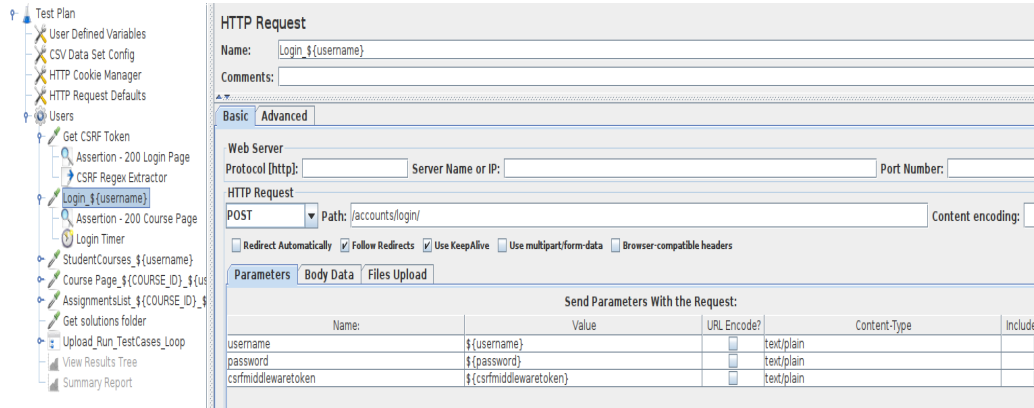


Figure 5: Student Login

5. Access course page

Before accessing course page, course id is extracted from an API response, which will give course details of student in json format. As shown in Figure 6, course id is extracted into a JMeter variable COURSE_ID by using Regular Expression Extractor JMeter element.

Course page is accessed using HTTP Request JMeter element. As shown in Figure 7, course page API is being called using HTTP get request, which sends course id extracted in above step in the request URL.

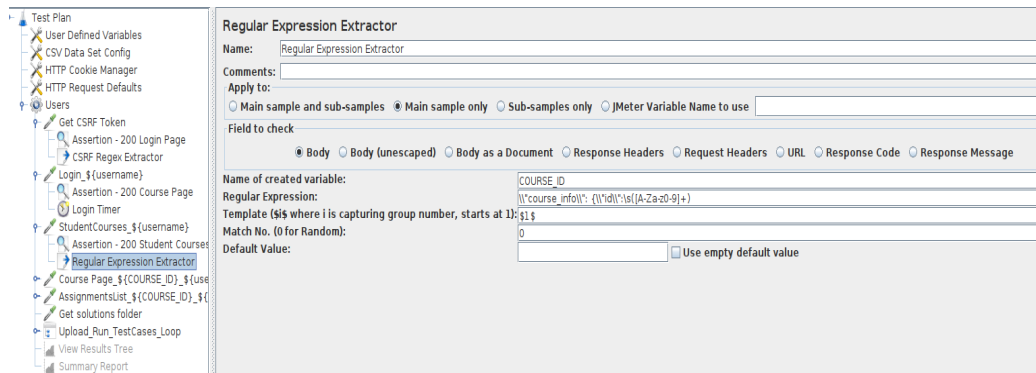


Figure 6: Extracting Course ID using Regular expressions

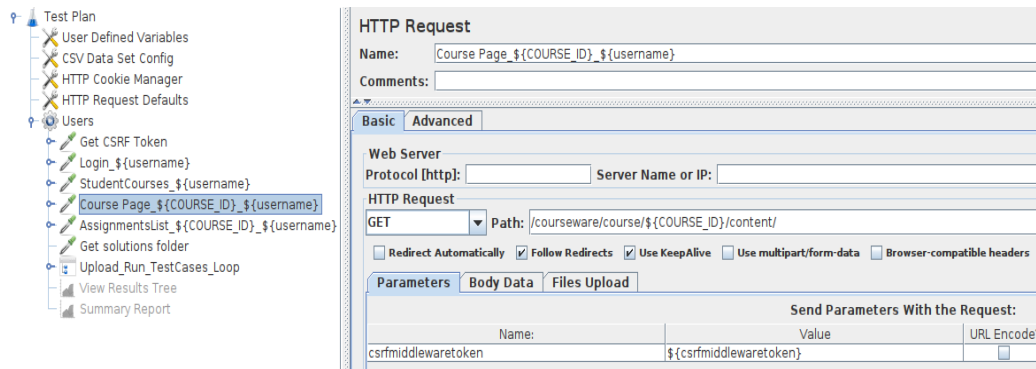


Figure 7: Access course page

6. Extracting Assignment Id

Assignment Id is extracted from an API response, which will give assignment details corresponding to a course of student in json format. As shown in Figure 8, Assignment id is extracted into a JMeter variable ASSIGNMENT_ID by using Regular Expression Extractor JMeter element.

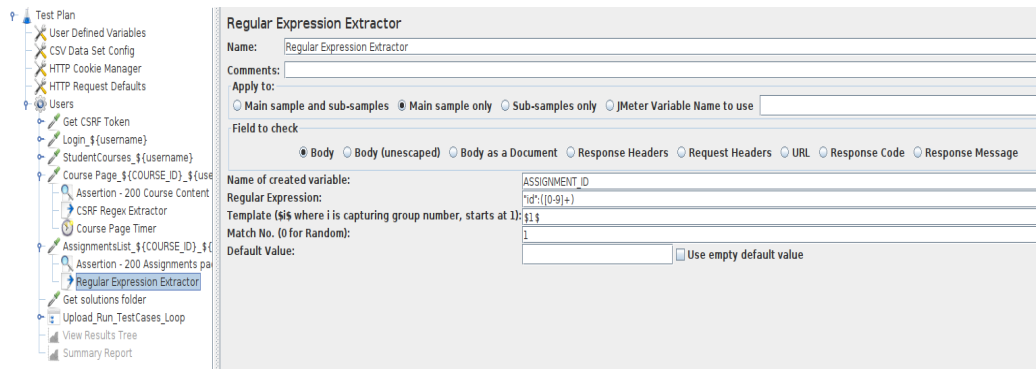


Figure 8: Extracting Assignment ID using Regular expressions

7. Get random solution file and upload it

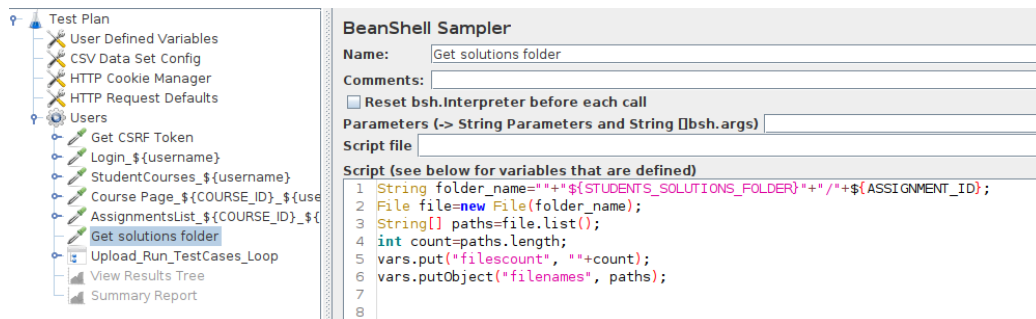


Figure 9: Get Solutions folder

For simulating Upload solution feature in our load test, we are using JMeter BeanShell elements for picking a random solution file to be uploaded.

Before picking a solution file, as shown in Figure 9, solution folder is picked based on Assignment Id.

Now as shown in Figure 10, the folder path which we got in the last step is used in the bean shell script to pick a random solution file.

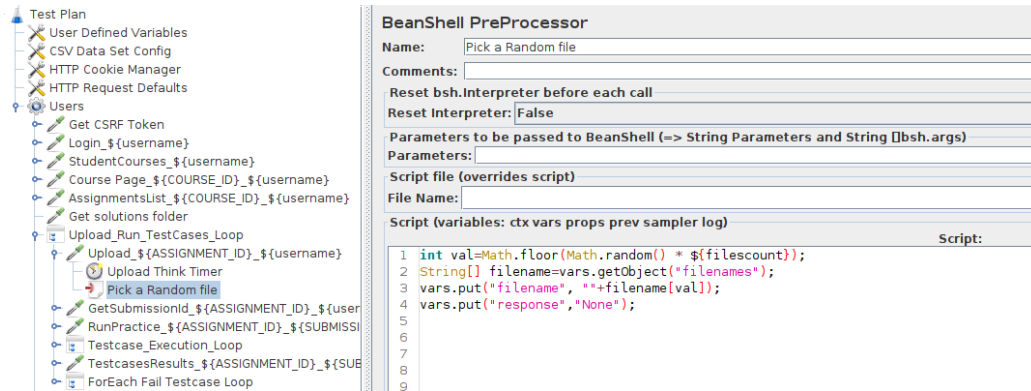


Figure 10: Pick a Random Solution file

The solution file which we got in above step is uploaded using HTTP Request JMeter element. As shown in Figure 11, Upload API is being called using HTTP Post request, which sends Assignment id in the request URL and file data is sent as a payload.

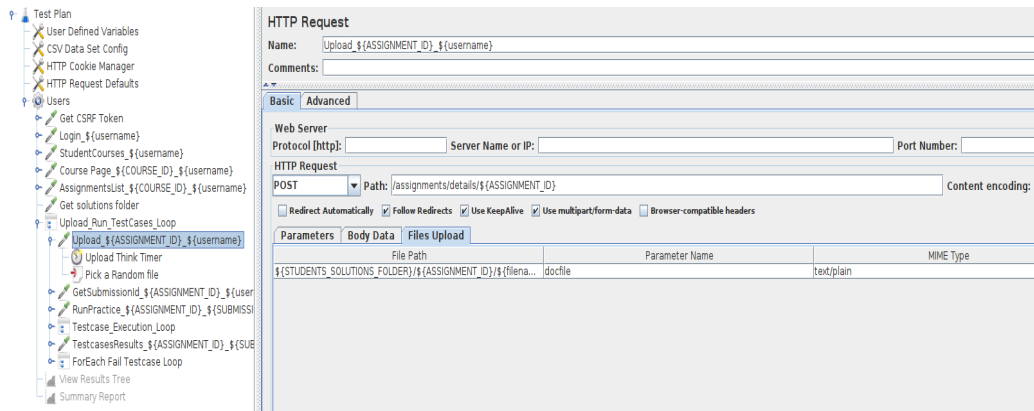


Figure 11: Uploading solution file

8. Get Submission Id

Before running practice test cases button, submission id is extracted from an API response, which will give submission id of the assignment in a HTML document. As shown in Figure 12, submission id is extracted into a JMeter variable SUBMISSION_ID by using Regular Expression Extractor JMeter element

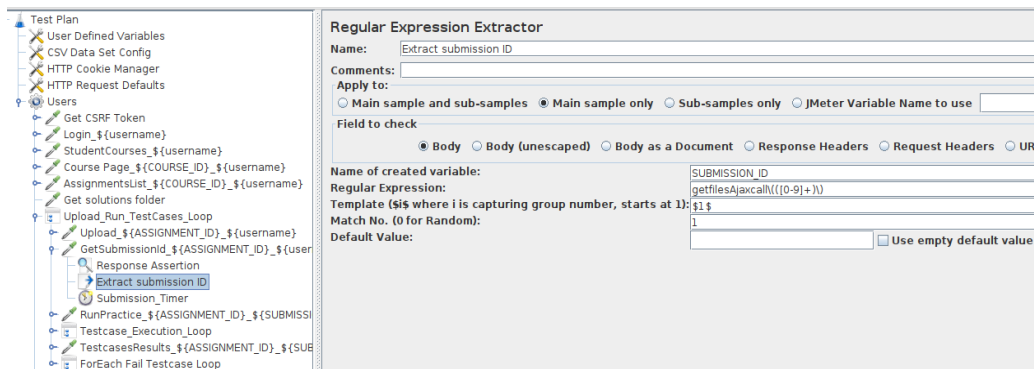


Figure 12: Extracting submission id

9. Click on run practice test cases button

For simulating click on run practice test cases button, we are using HTTP Request JMeter element. As shown in Figure 13, HTTP get request is used in which submission id we got in previous step is sent in the URL. The time when this request is executed is logged, which is used in coming steps.

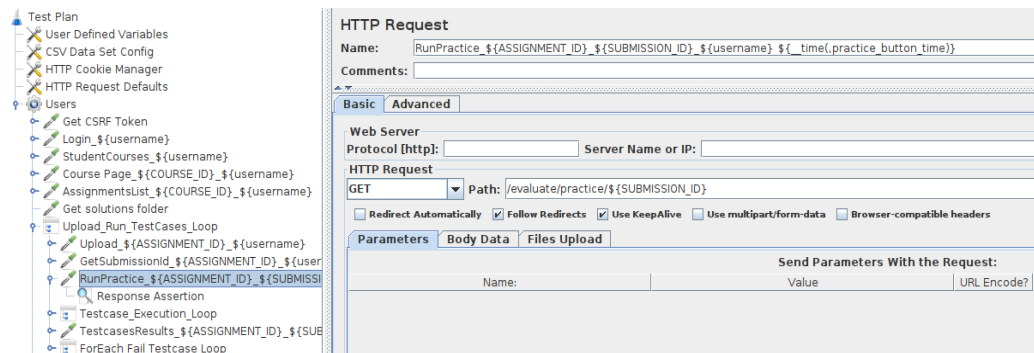


Figure 13: Run Practice Test Cases

10. Auto refreshing Test case status

The test cases execution status is checked periodically, until all the test cases have been completed execution. we are simulating this behaviour using the While Controller JMeter element, which will check test case status for every 3 seconds until all the test cases have completed execution.

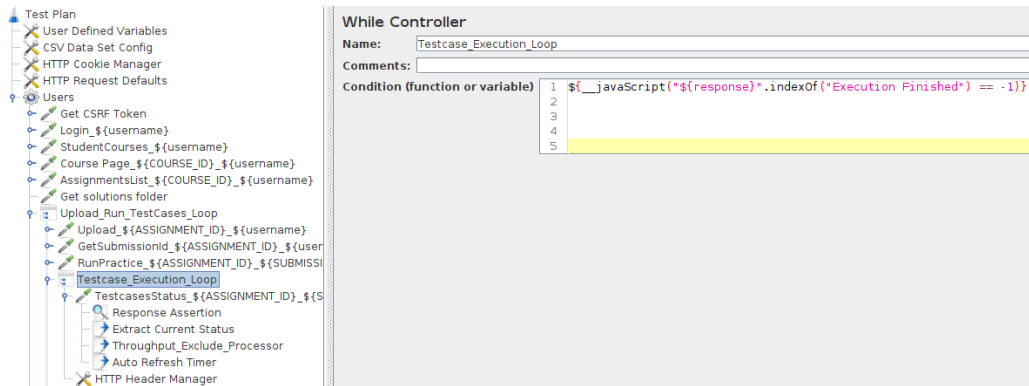


Figure 14: While Controller to loop test case status

As shown in Figure 14, the loop stops when "Execution Finished" comes in the response. Test case status is checked using HTTP Request JMeter element. As shown in Figure 15, HTTP post request is used in which submission id is sent in the request URL

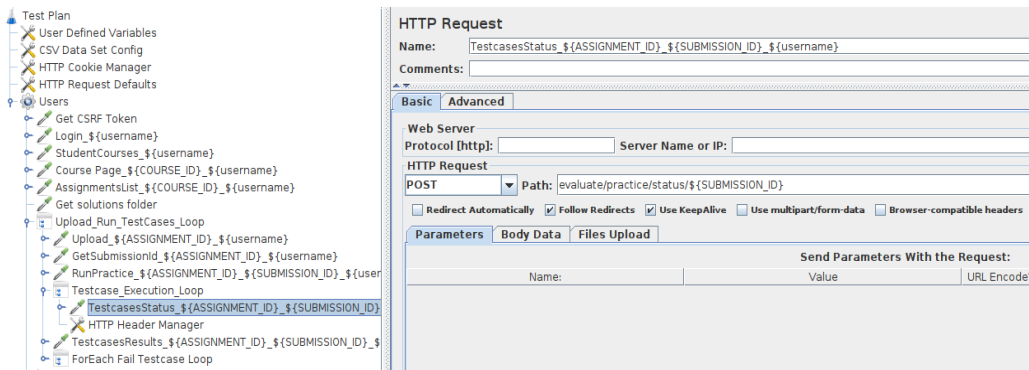


Figure 15: Test Cases Status Request

We are excluding these requests which are invoked in a loop periodically, from overall throughput by using JMeter bean shell element.

As shown in Figure 16, the code excludes the request from JMeter summary, as a result they will not be counted in the throughput.

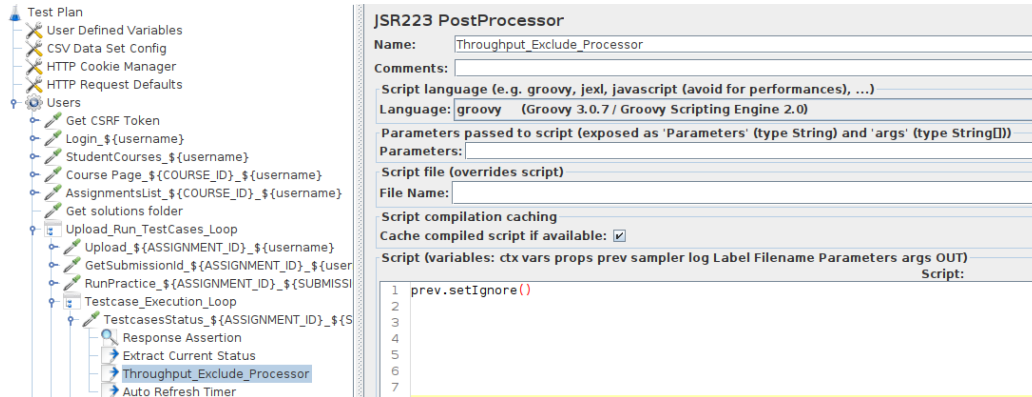


Figure 16: Exclude Auto refresh requests from overall throughput

11. Get the test case results page

After all the test cases have completed execution, test case results page will be displayed automatically. we are simulating this by using HTTP Request JMeter element. As shown in the Figure 17, HTTP get request is used in which submission id is passed in the request URL.

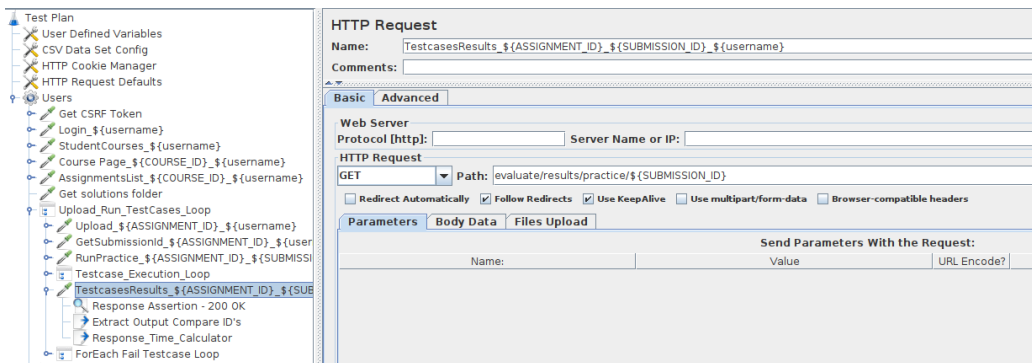


Figure 17: Test Case Results page

The response time of the entire run practice test is the difference between the time of getting the test case results page and time when the run practice test button is clicked. To achieve this we are using

JMeter bean shell element.

As shown in the Figure 17, we are calculating difference between the time when the response of practice test cases results page has come and time when practice test cases button is clicked (this we have logged in step 9)

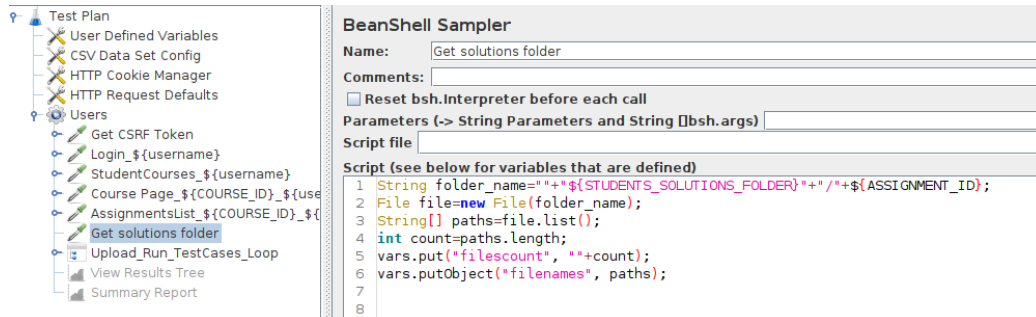


Figure 18: Practice test case Response time calculator

12. Compare outputs for each failed test case

For simulating Compare Outputs behaviour, we have extracted (as shown in Figure 19) Compare ID's for failed test cases from the response of test case results page and stored it in an array.

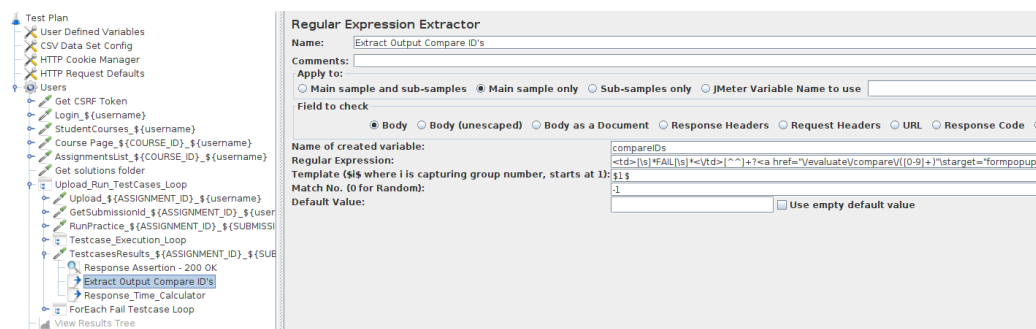


Figure 19: Extract Compare IDs using Regular Expressions

For iterating through the array we have used JMeter ForEach controller element as shown in Figure 20.

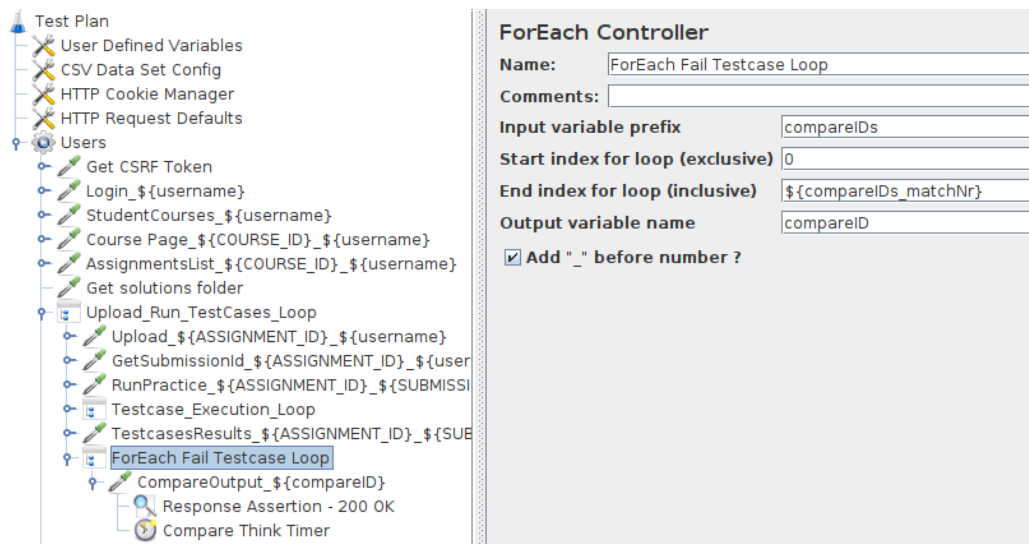


Figure 20: ForEach Controller to compare test case outputs

For simulating compare output for failed test cases we have used JMeter HTTP Request element as shown in Figure 21, HTTP get request is used in which compare id is passed in the request URL. We have used think time of one second after clicking on compare output.

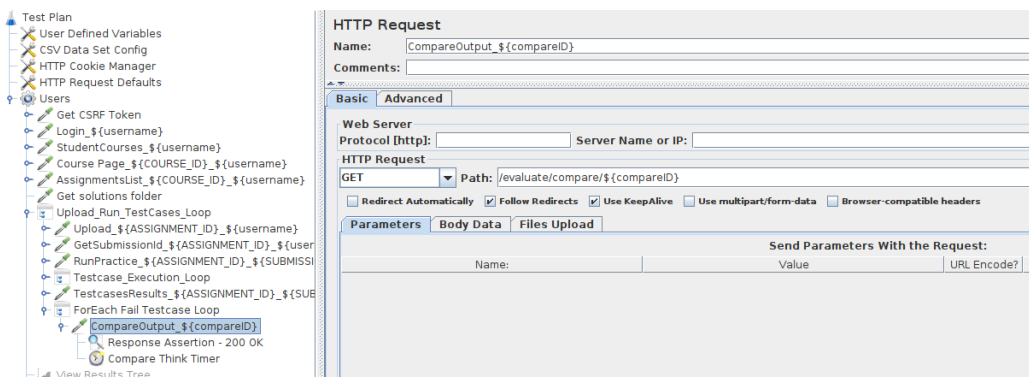


Figure 21: Compare Outputs

13. Goto step 7 and repeat the 7-12 steps until load duration finishes.

3 Server side scripts

For collecting snapshots of performance metrics during the load test, we have developed a script which will use `iostat`, `mpstat`, `netstat`, `vmstat`, `ps` Linux commands and produce snapshots periodically.

we also developed a post processing script which will read the snapshots and generate summary out of them after the load test is completed.

1. `iostat`

`iostat` command gives output about how many bytes read, written over a I/O device. we have developed a bash script for converting `iostat` output produced every 5 seconds to json format. Developed a post processing python script for generating summary in csv format.

2. `netstat`

`netstat -i` command gives output about how many bytes read, transferred over a network device. we have developed a bash script for converting `netstat` command output produced every 5 seconds to json format. Developed a post processing python script for generating summary in csv format.

3. `vmstat`

`vmstat` command gives output about CPU, Virtual memory usage. We have developed a bash script for collecting `vmstat` command output produced every 5 seconds to json. Developed a post processing bash script for generating CPU utilization summary.

4. `mpstat`

`mpstat` command gives output about CPU usage. We have developed a bash script for collecting `mpstat` command output produced every 5 seconds. Developed a post processing bash script for generating CPU utilization summary.

5. `ps`

`ps` command output about process level resource usage i.e CPU, physical memory. we have developed a bash script for converting `ps` output produced every 60 seconds to json format. Developed a post processing python script for generating summary in csv format.

The next section will discuss how our load test is performed.

4 Run load test

We have completely automated load test without any requirement of manual effort by using SSH Pass. The following are the steps involved in running the load test

1. Connect to the server using SSH Pass command and start the Server side scripts which we discussed in the last section in the background.
2. We have used the SED command to modify the number of users dynamically in the JMeter script file. JMeter script will run in non-GUI mode and stores the log files and results in output folder. The load test time is configured in JMeter file
3. After JMeter completes load test, connection to server is established using SSH Pass and all the Server side scripts running are stopped.
4. We have kept the cooling period, the time between one load test to next load test as 10min.
5. Steps 1 to 4 are repeated until the loop containing number of users input is completed.
6. After completion of all the load tests, connection to server is established using SSH Pass and post processing scripts are executed, which summarize the metrics

The code to our load test is shown in the Figure [22](#).

The next section will discuss our experiment results both intermediate and final.

```

load_test.sh
1  #!/bin/bash
2  for ((i=$1;i<=$2;i+=3))
3  do
4      echo "run started"
5      sshpass -p "panda" ssh panda@10.129.131.6 "cd rnd_final;./scripts.sh $i" &
6      bash -c "cd rnd_final;./scripts.sh $i" &
7      echo "Number of users : ${i}"
8      sed -i -E "s/\"ThreadGroup.num_threads\">[0-9]+)/\"ThreadGroup.num_threads\">${i}/g" evalpro_load_test.jmx
9      TIMESTAMP=`date +%d%m%Y_%H%M%S`
10     path="test_results/users_${i}_${TIMESTAMP}"
11     if [ ! -d $path ]
12     then
13         mkdir $path
14     fi
15     /home/panda/rnd_load_test/apache-jmeter-5.3/bin/jmeter -n -t evalpro_load_test.jmx -l $path/results.csv -e -o $path/output
16     sshpass -p "panda" ssh panda@10.129.131.6 "ps -ef | grep -v grep | grep scripts.sh | awk '{print \$2}' | xargs kill"
17     ps -ef | grep -v grep | grep scripts.sh | awk '{print \$2}' | xargs kill
18     sleep 600
19     if [ -f jmeter.log ]
20     then
21         mv jmeter.log $path/
22     fi
23 done
24
25 echo "post processing started"
26 for ((i=$1;i<=$2;i+=3))
27 do
28     sshpass -p "panda" ssh panda@10.129.131.6 "cd rnd_final; ./post_processing.sh $i"
29     bash -c "cd rnd_final;./post_processing.sh $i"
30 done
31 echo "post processing completed"
--

```

Figure 22: Load Test Code

5 Results

5.1 Intermediate results

For intermediate results ,we have run our experiments on a single core and we have simulated our JMeter script only up to course page, load test is run for 5 minutes

From the Figure 24, we can say that maximum throughput is around 47 requests/sec.

From the Figure 23, we can say that that Response time graph has become linear at around 200 users, maximum number of users the system supports or saturation number is around 170 to 200 users. This result also matches with utilization graph 25 as at this number of users server utilization is around 75 percent.

We can say that our intermediate results, are satisfying theoretical laws, utilization law, Little's law

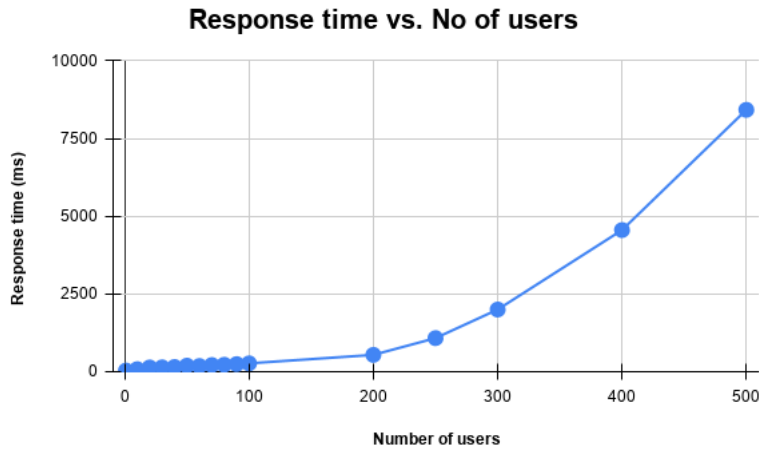


Figure 23: Intermediate - Response time Vs Number of Users

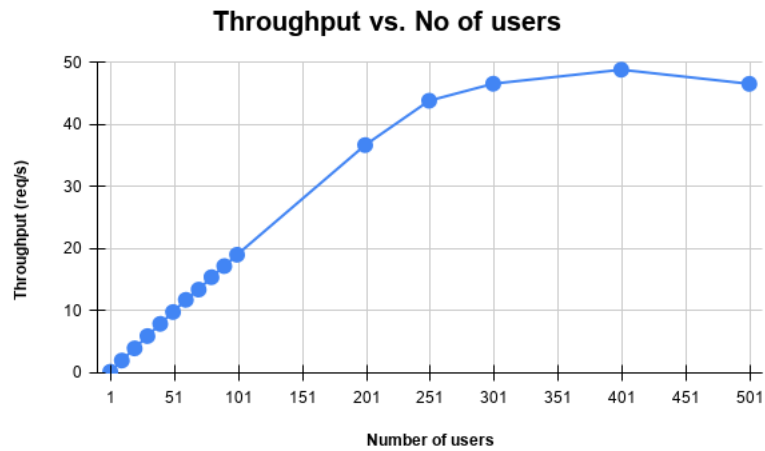


Figure 24: Intermediate - Throughput Vs Number of Users

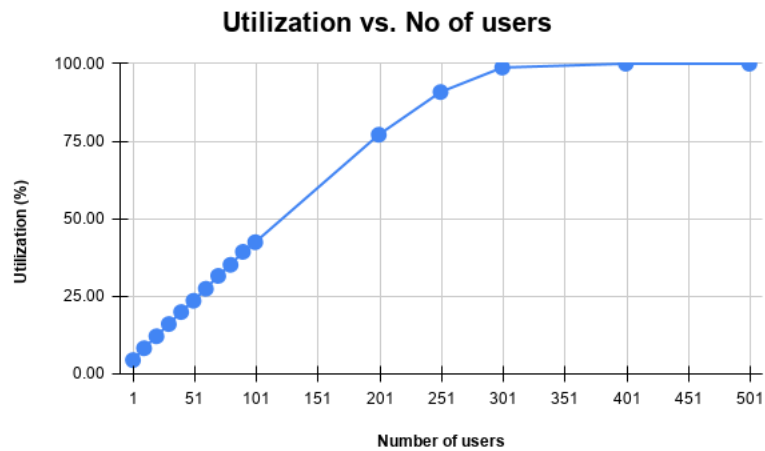


Figure 25: Intermediate - CPU Utilization Vs Number of Users

5.2 Final results

For final results, we have run our experiments on 16 cores and we have simulated our JMeter script for all the features. The load test is run for 15 minutes.

From the Figure 26 we can say that Response time got saturated from 450 users, which is surprising as we expected the Response time graph to become linear.

From the Figures 27, 28 we can say that throughput and utilization didn't get saturated.

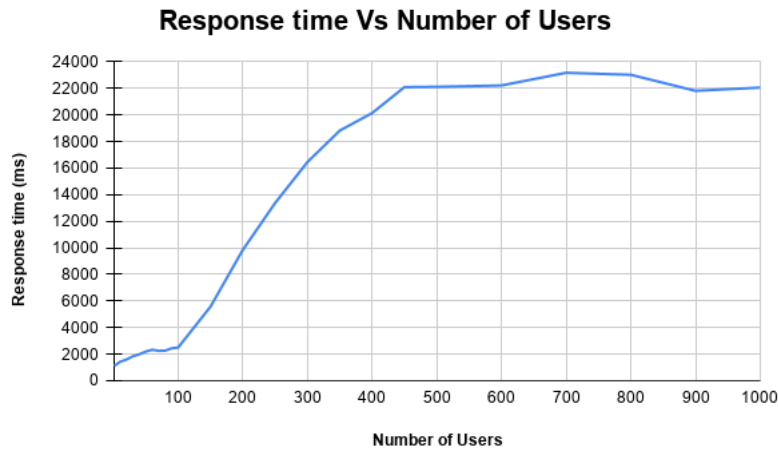


Figure 26: Final - Response time Vs Number of Users

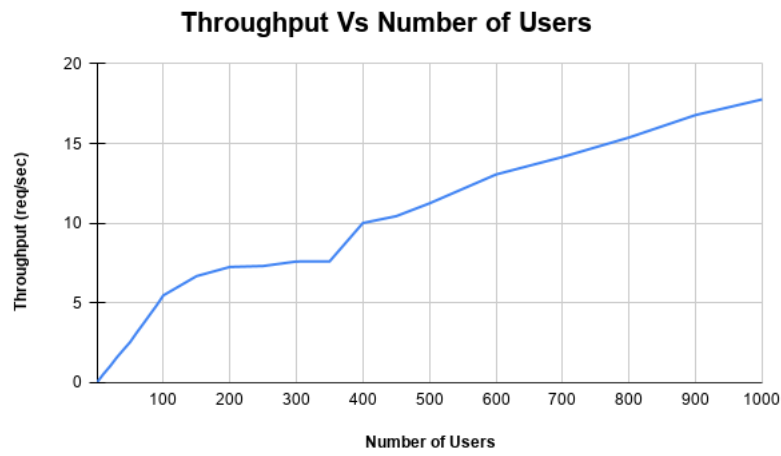


Figure 27: Final - Throughput Vs Number of Users

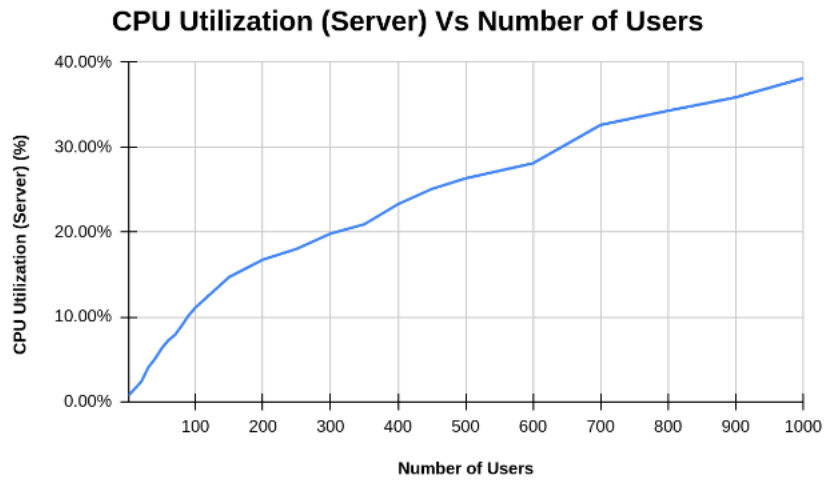


Figure 28: Final - CPU Utilization Vs Number of Users

6 Conclusion

We now have a complete JMeter script from all the eval pro features of Bodhi-tree. We also have scripts which automated load test and the post processing of metric snapshots after load test.

In our intermediate results, the metrics we got are satisfying theoretical laws i.e Utilization law, Little's law. But in our final results which includes Assignment upload, Run practice tests and compare outputs we are getting abnormalities in the metrics, compared to theoretical laws.

The reason for these abnormalities need to be closely looked at, which is of future scope. Identifying bottleneck service demand is also of future scope.

7 Acknowledgement

We would like to thank Prof. Varsha Apte for her guidance and constant support throughout the semester. We also thank Ishtiyag Husain for setting up lab machines to perform our experiments.

8 References

1. https://wiki.bodhi-tree.in/get-started/installation/installing_bodhitree/
2. https://drive.google.com/file/d/1E_rRf2wu07NvVJjGyiWEDH0P_Fmol28q/view
3. https://docs.google.com/document/d/16_oApb4uXP1RF4q1p619te0rKx6WV-RPmcJqHokxGPA/edit#heading=h.19m1v14q4ouo
4. <https://docs.google.com/document/d/10NJ27gwbD-F1jcd1WNErXX4BYcH0VLXgZr2Q9rofeLE/edit>
5. https://docs.google.com/document/d/1vDY9iajMyr9JJx_gd1KFYcHrxHykLYcJKNFzl920Nak/edit
6. <https://unix.stackexchange.com/questions/417672/how-to-disable-one-cpu>
7. <https://www.blazemeter.com/blog/comprehensive-guide-using-jmeter-timers>
8. <https://stackoverflow.com/questions/53857365/how-to-remove-response-time-samplers-from-the-jmeter-results>
9. <https://jmeter.apache.org/>