Stress Testing Tool

to check the performance of a Moodle Instance

D.G.C.Galpaya

2019

Stress Testing Tool to check the performance of a Moodle Instance

A dissertation submitted for the Degree of Master of Computer Science

D.G.C.Galpaya

University of Colombo School of Computing

2019

Declaration

The thesis is my original work and has not been submitted previously for a degree at this or any other university/institute.

To the best of my knowledge it does not contain any material published or written by another person, except as acknowledged in the text.

Student Name: Don Galpayage Chathura Galpaya

Registration Number: 2016/MCS/031

Index Number: 16440319

_____

Signature: Date: 06/05/2019

This is to certify that this thesis is based on the work of Mr. Don Galpayage Chathura Galpaya under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Certified by:

Supervisor Name: Professor K.P. Hewagamage

_____

Signature: Date: 06/05/2019

Abstract

This thesis describes in detail the development and implementation of Stress Testing Tool to check the performance of a Moodle instance. The main objective of this project is to develop a tool to test the performance of a Moodle instance/LMS by simulating multiple virtual users by considering four scenarios, and to record the vital information of the server such as CPU usage, RAM, disk input/output, bandwidth, SQL usage, maximum load can handle etc. to minimize the possible bottlenecks.

Most of the existing tools for load testing support the creation of simple test cases consisting of a fixed sequence of operations. In this thesis it is presented a new approach to perform load testing of Moodle instance by simulating realistic user behavior. In order to give the generated load some variety or randomness it requires using dynamic URL concept to modify and parameterize these test cases manually. This is usually both time-consuming and difficult. In this thesis by simply doing simple changes to the URL it can be achieved. This thesis discusses four types of stress tests. User can do the simulation by using specified no of virtual users with (1).Single user login credential, (2).Multiple user logins credentials system generated users,(3).Multiple user logins credentials provided by a CSV file, (4).simulation with several load engines with multiple user login credentials.

This approach is further can be improved by using several category of users in the simulation process and also testing from several geographical locations with multiple load engines. This tool also can be used as a preventive maintenance tool which can predict the system behavior to avoid and minimize downtime and keep the server and online applications running.

Acknowledgement

This thesis of the “Stress Testing Tool to check the performance of a Moodle” prepared for the Master of Computer Science program of the University of Colombo School of Computing. I was able present this successfully with the help of people surrounded me.

First and foremost I offer my sincerest gratitude to my supervisor Professor K.P.Hewagamage the Director of University of Colombo School of Computing, who gave the suggestion of developing a Stress Testing Tool for a Moodle Instance. Also Professor K.P.Hewagamage gave me useful instructions, guidance and supervision though out the project from project proposal to the final thesis submission.

I take this opportunity to give extent sincere thanks to Mr.Geeth Hettiaarchchi the Moodle Administrator of University of Colombo School of Computing gave me an introduction to the Learning Management System of the UCSC..

Finally, express my gratitude to my family members and colleagues in MCS program at UCSC for helping to complete this project successfully. Completing this Thesis work would become more difficult if I did not get the support from the people above mentioned.

Chathura Galpaya

Contents

List of Figures and Tables

List of Acronyms

URL - Uniform Resource Locator

URI – Uniform Resource Identifier

LMS - Learning Management System

MOODLE – Modular Object-Oriented Dynamic Learning Environment

UI – User Interface

CSM - Computational Structure Model

SNMP - Simple Network Monitoring Protocol

WMI - Windows Management Instrumentation

ICMP - Internet Control Message Protocol

CPU – Central Processing Unit

RAM – Random Access Memory

RAID - Redundant Array of Independent Disks

GPL - General Public License

Chapter-1

Introduction

1.1. Project Domain

Learning Management System (LMS) or modular object-oriented dynamic learning environment (MOODLE) applications are very popular among the educational institutes since these applications facilitate the administration, documentation, tracking, reporting and online delivery of educational courses or training programs. Besides the quality of teaching material within the LMS, the user interface (UI) and the speed with which the LMS responds are the two principal aspects in an LMS. If the LMS has a poorly constructed UI which requires significant effort to navigate or if the LMS is slow to respond to user interaction, then there is a risk the student's concentration will be lost and their learning performance harmed.

1.2. Problem

Performance of the Moodle or LMS sites significantly hinders during the examinations, assignments and quizzes period since hundreds of students simultaneously connect to the LMS in order to finish their course works before the deadline. The main reason for this is during the development stage programmers and QA team focused solely on catching bugs, many websites ignore functionality testing, usability testing and performance testing which are three critical elements in defining the user experience with a website or web application.

The main challenges of a Moodle administrator are to ensure that the webserver is as fast and reliable as possible so that students can do their learning without getting distracted. This is made possible by testing different scenarios, with specific numbers of users. The testing is necessary to carry out to find out:

- How long it takes to perform a specific task, for one or multiple concurrent users?
- How many people can use LMS simultaneously to perform a series of tasks without crashing the system?
- How much RAM, CPU, bandwidth, etc. is used during a test and check whether the system hardware configuration is adequate.

- If an upgrade is needed to my hardware/hosting package and when it needs to upgrade?

1.3. Main Objective

The main objective of this project is to develop a tool to test the performance of an LMS server by incorporating realism or randomness to the load test in order to mimic the real user behaviour. Thereby collecting the vital information such as Response time, latency, Memory & CPU utilization, SQL usage and maximum load can handle etc. to minimize the possible server outages. To get meaningful results it is necessary to have a testing strategy that mimics real-world usage as much as possible. To achieve that, the following scenarios need to be considered.

- Test a mixture of activities
    - some that are database intensive e.g. chat
    - some that are disk intensive e.g. download large files
    - some that are CPU intensive e.g. quiz
- Add randomness to the test, if the tool allows doing so. For example, apply some random 'waiting' time between clicks
- Test a mixture of accounts (teacher, admin, student) – not always able to do this
- If the test is on a live server, perform the tests when there is a low traffic period.
- Repeat the same test several times, at different times of the day. This is especially important if the testing on a 'live' server.
- Also, simulate the same tests from several geographical locations with different load engines.

1.4. Motivation

There are a lot of free LMS or Moodle sites readily available on the internet but it is difficult to identify the usability of these sites with the increasing number of users or students. Without fully loading the Moodle with the data such as courses, students and lecturers it is not possible to get an idea about the performance of those Moodle or the LMS which is freely available on the internet. This issue paved the path to my MSc project proposal to develop a Stress Testing tool in order to simulate the Moodle Instance with a specified number of users.

In this project, the Moodle Server Stress Test Testing Tool generates a simulated "brute force" attacks that apply excessive load to the webserver and analyse the performance using an analyser. In real world situation, this can be created by a massive spike of users caused by a large advertising campaign or an email marketing campaign sent to prospective customers that asks them to come to the website to register for a service or request additional information. An inadvertent denial of service to the users that are ready to learn more about your product could have a serious impact on your bottom line.

1.5. The exact computer science problem

The purpose of stress testing is to estimate the maximum load that target webserver can support. Moodle Server Stress Testing Tool can help to learn the traffic thresholds of the webserver and how it will respond after exceeding its threshold.

The speed with which an LMS responds can be measured through a metric called latency. Latency is the length of time between a user action (e.g. clicking on a link) and the corresponding response is received (being shown the page the link refers to). Lower latency is generally considered to be better and has been demonstrated to lead to increased user satisfaction in commercial web environments.

In practice, Moodle servers cannot be simulated by using a single user login. Each user has its own workspace and assigned task. For example, if a quiz is assigned to a student by a teacher that cannot be simulated by multiple users because when the student consumes the particular link it is not visible or available for the even though the same user connects with a different session. Also, the assigned quiz could only be done by that particular user with the login credentials. A load test is valid only if virtual users' behavior has characteristics similar to those of actual users because failure to mimic real user behavior can generate totally inconsistent results. Therefore it is not practical to virtualize the real world scenario using a single login credential with multiple sessions.

However, in order to give the generated load some variety, it is usually necessary to modify or parameterize these test cases manually. This is usually both time-consuming and difficult. To cater to this issue a dynamic URL concept is used to eliminate the barrier of testing using single user login.

1.6. Research contribution

There are three principal methods for computer performance evaluation and analysis i.e. direct measurement, simulation and analytic modeling. Direct measurement is the most accurate of the methods, but requires the system to be implemented first in order to collect specific performance information. Besides, the motivation of performance engineering is to find the performance bottleneck at the design phrase, so one can avoid an efficient design at the earliest time. Simulations are prototypes of the real system that provide a fairly accurate performance measurement, but are often time consuming and difficult to construct. Analytic modeling which exercises techniques such as queuing networks, Markov models, Perti-nets, state charts and CSM, is the least expensive because hardware and software do not need to be implemented. It also provides insight into the variable dependencies and interactions that are difficult to determine using other methods.

In the Internet there are several Webserver Stress/Performance Testing tools available but most applications does is create simultaneous URL clicks and check the performance. The Web Server Stress Testing Tool developed by this project is specifically design to test a Moodle instance server (LMS). This tool expected to be used to stimulate a Quiz/upload and download assignments with specified number of virtual users in order to mimic the real world situation, because Moodle servers get slow specially during the period of online quizzes and the deadline of Assignments. Not like the proprietary tools in the market this tool is customizable depending on the requirement for example to stimulate an online exam or uploading and downloading assignments.

1.7. Scope

This Webserver Stress Testing Tool is design to simulate independent user requesting webpages based on a set of URLs which includes pages, images, frames, videos etc. These URLs can be fed to the system by manually typing it or by the URL recorder. URL recorder is expecting to develop as a separate web browsers and it enables the user to record the URLs. URL recording is done at each POST while the user surfing through the Moodle.

After recording the activities of the individual user (recording the URLs) the user is simulated with specified number of virtual users by using separate thread with its own session information. These URLs can be parameterized for each user and the executing sequence of URLs can be varied.

This Webserver Stress Testing tool requires the webserver end support otherwise it is difficult to automate the requests for a "brute force" attack since most of the existing webservers are designed in such a way that cannot be overloaded the server by "brute force" attack. Also this cannot be used as generic Webserver Stress Testing Tool because this requires a little bit changes in the scripts depending on the web pages GET/POST parameters.

This Webserver Stress Testing tool is preliminary designed to be used in Microsoft Windows based operating systems. This tools can be used to analyze the vital information of the server such as CPU usage, RAM, disk input/output, bandwidth, SQL usage, maximum load can handle etc. to minimize the possible bottlenecks. To get meaningful results it is necessary to have a testing strategy that mimics real world usage as much as possible. To achieve that following scenarios need to be considered.

- Test a mixture of activities
    - some that are database intensive e.g. generating summary reports
    - some that are disk intensive e.g. download large files
    - some that are CPU intensive e.g. quiz
- Add randomness to the test, if the tool allows doing so. For example apply some random 'waiting' time between clicks
- Test a mixture of accounts (teacher, admin, student) – not always able to do this
- If the test is on a 'live' server, perform the tests when there is low traffic period.
- Repeat the same test several times, at different times of the day. This is especially important if the testing on a 'live' server.

1.8. Deliverables

As mentioned in the objectives, the main deliverable is the Webserver Stress Analysis tool and the Dissertation. In addition, user manual or help facility is also an essential communication documentation intended to give assistance to people using a particular system. Apart from that the entire system delivers a LMS website which helps to simulate the functionality of the Webserver Stress Analysis Tool.

1.9. Structure of Thesis

Chapter-2

Provide the literature review of the project with reference to published materials in research papers, URLs, Magazine article.

Chapter-3

Provides the detail of the problem analysis, proposing model/design and methodology are presented in this chapter. Overall system architecture and features of the systems explained in details.

Chapter-4

Implementation details are explained in this chapter including implementation environment, tools and techniques. System deployment architecture was briefly discussed.

Chapter-5

Evaluations and results are provided in this chapter. As an innovative application, system was evaluated with a set of predefined evaluation criteria.

Chapter-6

Conclusion and future work were discussed in this chapter. To improve this application, lot of works need to be done.

Chapter-2

Literature Review

With the increased popularity of Web servers and the Moodle Cloud, recently Web server performance modeling and analysis has become an active area of research. To the best of my knowledge there is few published research work that presents a comprehensive analysis of performance for Web server systems. Below is a brief review of the previous work on performance analysis of Moodle servers.

2.1. Related Work/Background Study

Despite the exponential growth of the WWW, a very negligible amount of research has been conducted in Moodle instance web server performance analysis with a view to improve the time a Webserver takes to connect, receive, and analyze a request sent by the client and then sending the answer back to client. Following is current research areas done by webservers or Moodle Stress Testing.

2.1.1. The Form-Oriented Model

Form-oriented analysis is a methodology for the specification of ultra-thin client based systems. Form oriented models describe a web application as a typed, bipartite state machine which consists of pages, actions and transitions between them. Pages can be understood as sets of screens, which are single instances of a particular page as they are seen by the user in the web browser. The screens of a page are conceptually similar, but their content may vary, e.g. in the different instances of the welcome page of a system, which may look different depending on the user. Each page contains an arbitrary number of forms, which in turn can have an arbitrary number of fields. The fields of forms usually allow users to enter information, and each form offers a way to submit the information that has been entered into its fields to the system.

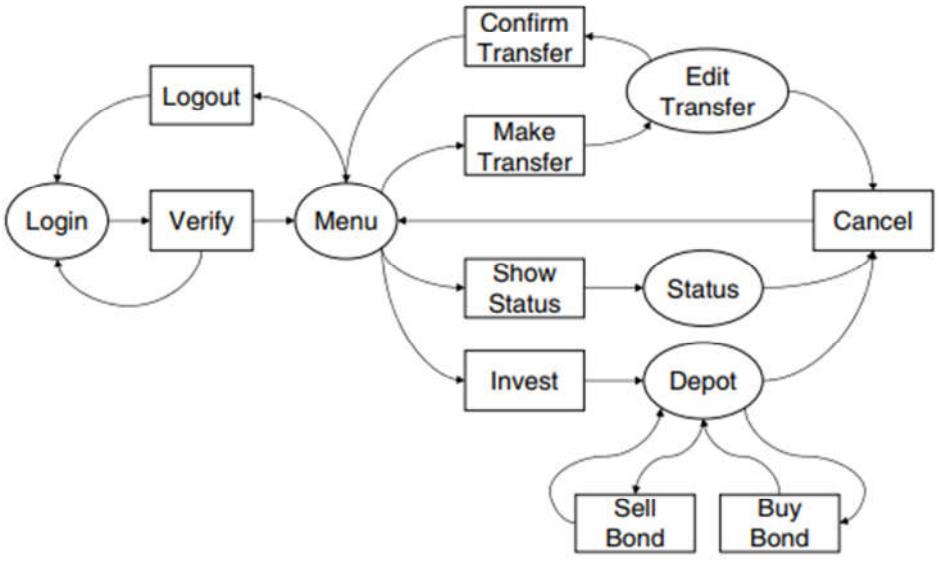


Figure:-2. 1: Form-chart of example home banking web application.

A submission invokes an action on the server side, which processes the submitted information and returns to the client a new screen in response. Hyperlinks are forms with no fields or only fields that are hidden to the user**.**

2.1.2. Load Testing of Legacy Systems

Often we face the task of load testing a legacy system, i.e. a system which is already deployed and running, and for which the information necessary to create a realistic user model is not available. In such a case we could either use a very simple user model, such as a generic one that invokes actions randomly with a uniform distribution, or try to extract the necessary information by means of reverse engineering. It is proposed a methodology and a tool called Revangie which is able to reconstruct form-oriented analysis models for existing web applications. Models can be constructed online, i.e. during system exploration, but also offline e.g. from recorded user data. There also exist other tools for model recovery of web sites that can be useful for the creation of load models.

2.1. Server Monitoring & Statistics Software

Server Monitoring Software is an essential tool for system administrators, as it allows for automated reporting, scheduled checks and pre-emptive warnings about the health of your many servers within your operating environment. Server monitoring software is able to check everything about your system, such as: CPU usage, RAM utilization, Hard Disk Space, System Temperatures, Server Alerts (Hardware status warnings), RAID Array health checks Virtual Machine Alerts etc.

It is also possible to monitor user logins, suspicious activity on your server, and the status of your services and daemons. There are multiple layers of technology at work inside server monitoring software suites, and some of the most common protocols that we will be mentioning today are:

- SNMP (Simple Network Monitoring Protocol)
- WMI (Windows Management Instrumentation)
- ICMP (Ping)
- PerfMon (Microsoft Windows Performance Monitor)

Below are some popular software tools which are used for Moodle Server Monitoring.

2.1.1. The Webalizer

The Webalizer is a fast, free web server log file analysis program. It produces highly detailed, easily configurable usage reports in HTML format, for viewing with a standard web browser.

2.1.2. Nagios

Nagios is a host and service monitor designed to inform you of network problems before your clients, end-users or managers do. It has been designed to run under the Linux operating system, but works fine under most *NIX variants as well. The monitoring daemon runs intermittent checks on hosts and services you specify using external "plugins" which return status information to Nagios. When problems are encountered, the daemon can send notifications out to administrative contacts in a variety of different ways (email, instant message, SMS, etc.). Current status information, historical logs, and reports can all be accessed via a web browser.

2.1.3. Munin

Munin is a networked resource monitoring tool that can help analyze resource trends and "what just happened to kill our performance?" problems. It is designed to be very plug and play. A default installation provides a lot of graphs with almost no work. There are Munin plugins available for Moodle:

- ***Munin plugin for Moodle*** - Munin plugin for generating various Moodle stats developed and released under GPL terms by Lancaster University, UK. Works with PostgreSQL database only at the moment.
- ***Monitorización de Moodle con Munin***- Alternative Munin plugin for Moodle found in the search, stats data is collected by Moodle plugin and recorded to the text files on cron, these text files are then being read by the Munin plugin itself. Docs is only in Spanish.

2.1.4. nmon

The nmon tool is designed for AIX and Linux performance specialists to use for monitoring and analyzing performance data.

2.1.5. Zenoss

Zenoss (Zenoss Core) is a free and open-source application, server, and network management platform based on the Zope application server. Released under the GNU General Public License (GPL) version 2, Zenoss Core provides a web interface that allows system administrators to monitor availability, inventory/configuration, performance, and events.

2.1.6. Awstats

AWStats is a free powerful and featureful tool that generates advanced web, streaming, ftp or mail server statistics, graphically. This log analyzer works as a CGI or from command line and shows you all possible information your log contains, in few graphical web pages. It uses a partial information file to be able to process large log files, often and quickly. It can analyze log files from all major server tools like Apache log files (NCSA combined/XLF/ELF log format or common/CLF log format), WebStar, IIS (W3C log format) and a lot of other web, proxy, wap, streaming servers, mail servers and some ftp servers.

2.1.7. Netdata

Netdata is a free, open source, simple and real-time performance and health monitoring tool with a beautiful web front-end. You can monitor CPU, RAM usage, disk I/O, network traffic, Postfix and much more using Netdata. Netdata gathers real-time performance data from Linux, FreeBSD, MacOS and SNMP devices quickly and effectively.

2.2. Load Testing Tools

Below is a comprehensive list of the most widely used performance testing tools for measuring web application performance and load stress capacity. These load testing tools will ensure your application performance in peak traffic and under extreme stress conditions. The list includes open source as well as licensed performance testing tools. But almost all the licensed tools have a free trial version so that you can get a chance to work hands-on before deciding which the best tool for your needs is.

2.2.1. LoadStorm

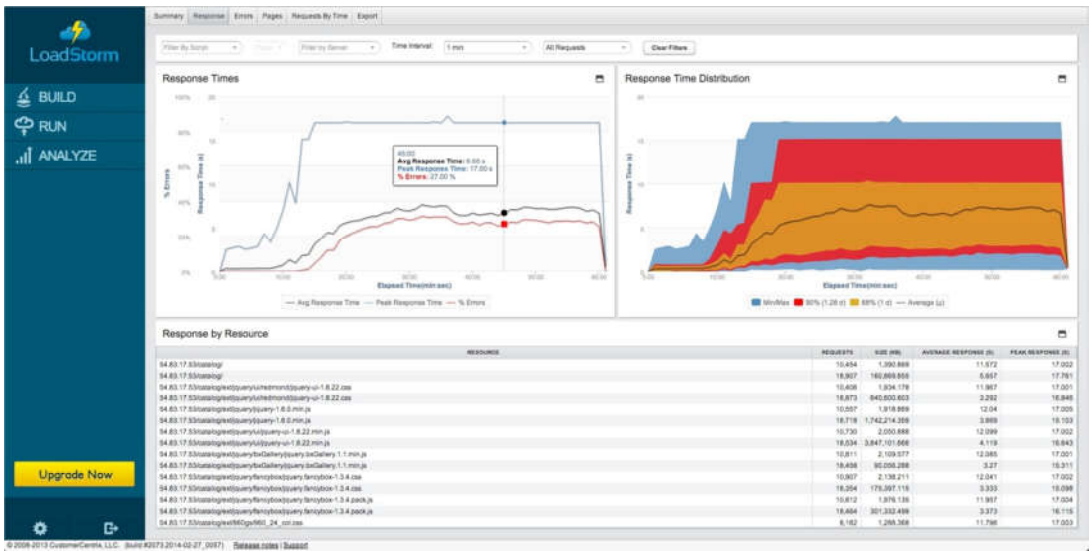


Figure:-2.2: LoadStorm load testing tool.

Loadstorm is a web based load testing service. Their business is to simulate multiple virtual users simultaneously navigating a website, following a pre-recorded scenario. Loadstorm is free to use for up to 25 simultaneous virtual users (forever, not just once), which is quite handy considering it is quite close to a 'regular' classroom size. The 25 user-limit is usually enough to test sites on shared hosting, and bring the server down. It is possible purchase extra virtual users to test larger sites.

Pros:

- Can be used on any device with a web browser and access to the Internet
- Easy to use for simple workflows
- Company behind the product provides excellent support
- Servers based around the World, mimics real-world usage for students accessing Moodle from home
- Relatively gentle learning curve

Cons:

- Pay-for service if you want to test more than 25 concurrent users
- Some more complex workflows are difficult to set up (e.g. multiple quizzes)
- Web based service so there is some extra latency involved

2.2.2. Webserver Stress Tool-7

Webserver Stress Tool is a powerful HTTP-client/server test application designed to pinpoint critical performance issues developed by Paessler.

By simulating the HTTP requests generated by hundreds or even thousands of simultaneous users, you can test your web server performance under normal and excessive loads to ensure that critical information and services are available at speeds your end-users expect.

Detailed test logs and several easy to read graphs make analyzing results a snap. Webserver Stress Tool for Windows (2003 R2, Vista, 7, 2008) can benchmark almost any HTTP server (e.g. static pages, JSPs/ASPs, or CGIs) for performance, load, and stress-tests.

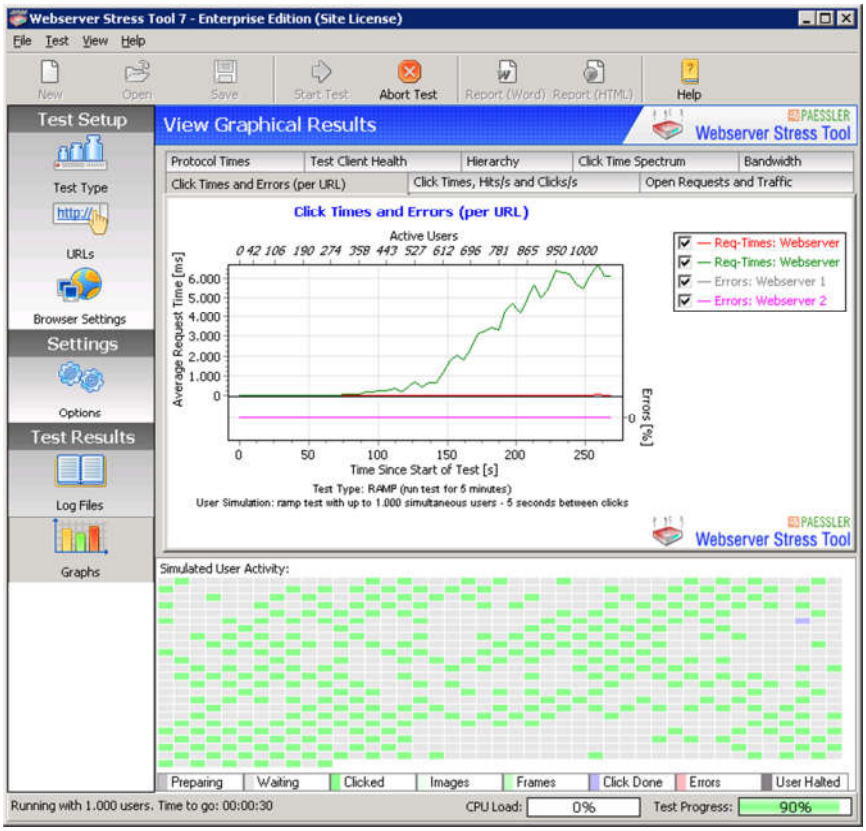


Figure:-2.3: Webserver Stress testing tool.

Pros:

- Built-in report generator: Reports can be generated as HTML files and MS WORD documents
- Includes a URL recorder to select the URL(s) you want to test (rather than typing them into a list)
- Works on any HTTP-URL or HTTPS-URL and can test any script (CGI, ASP, PHP etc.)
- Can also be used to test requests of larger download files (e.g. ZIP)
- Works with any webserver (no part of the software has to be installed on the server.
- Freeware

Cons:

- Some more complex workflows are difficult to set up (e.g. multiple quizzes)
- Web based service so there is some extra latency involved

2.2.3. Apache JMeter

Apache JMeter is a fully-fledged load and performance open-source software by the Apache foundation. It is much more powerful than the other tools mentioned but also has a much steeper learning curve. Luckily, an excellent JMeter script generator was created and shared by the good people of the Open University UK. Without this tool, it would be rather difficult to create scripts (scenarios) to test your Moodle installation. This tool will help to generate users, and get those users to post to forums, participate in chat sessions and take quizzes.

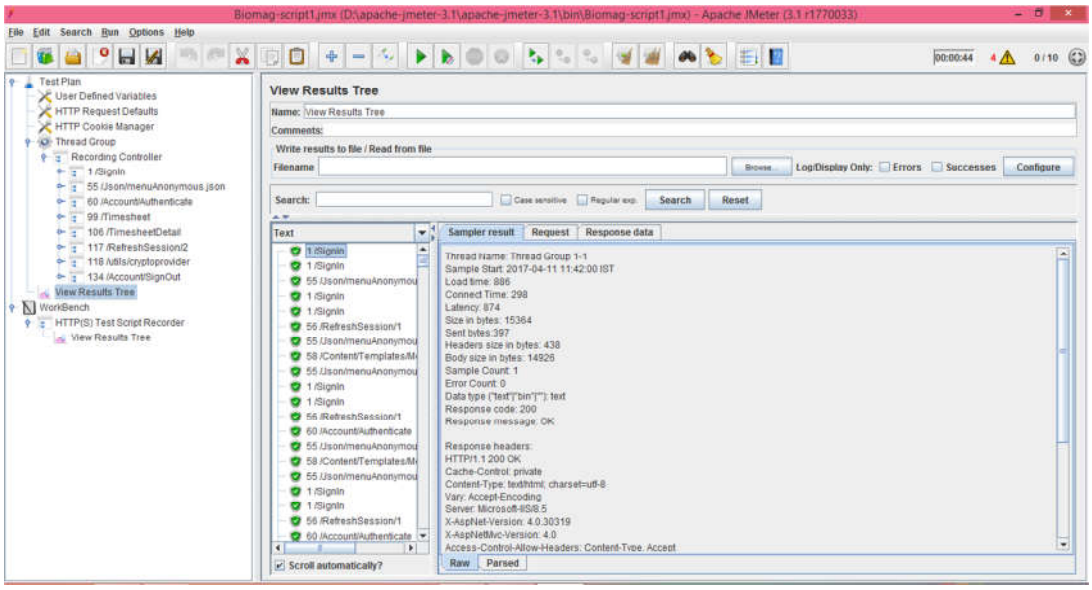


Figure:-2. 4: Apache JMeter load testing tool.

Pros:

- Java based, so can be used on all platforms
- Scripts are portable – test from within or outside your network
- Extremely powerful
- Freeware

Cons:

- Steep learning curve

2.3. Bibliography

[1] Dirk Draheim, John Grundy, John Hosking, Christof Lutteroth, Gerald Weber, "Realistic Load Testing of Web Applications"

[2] Institute of Computer Science, The University of Auckland

[3] Sorin POPA, Associate Professor, PhD," Web Server Monitoring", University of Craiova

[4]

[5] Jonathan Barber, Rodolfo Matos, Susana Leitão," Moodle Monitoring Best Practices", University of Porto (PORTUGAL)

[6] Analyzing server response time using Testing Power Web Stress tool, [Online]. Available:https://ieeexplore.ieee.org/document/5773700/

[7] Apache Jmeter, [Online]. Available:

http://jmeter.apache.org/download_jmeter.cgi

[8] Locust, [Online]. Available:

https://locust.io/

[9] Webserver Stress Tool, [Online]. Available:

https://www.paessler.com

[10] Apache Log Analyzer, [Online]. Available: https://www.manageengine.com/products/eventlog/apache-web-server-log-analyzer.html

[11] I teach with Moodle

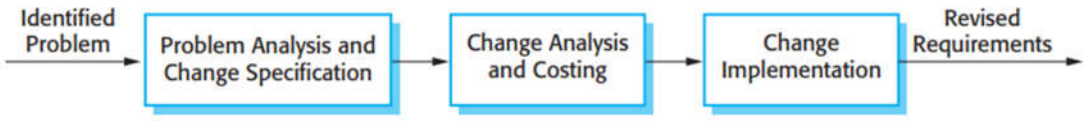
http://www.iteachwithmoodle.com/2012/10/10/3-free-tools-to-test-if-your-moodle-server-can-cope-with-large-amounts-of-students/

Chapter-3

Problem Analysis & Methodology

3.1. Problem Analysis

In this thesis the main problem is to find the performance of a Moodle instance or LMS with large number of users. In order to test that in real world we have to subscribed large number of users and ask them to do a Quiz or to upload an assignment to make the Moodle Server busy and check the performance of the Moodle Server against the each scenario. However this method is not practical because each time if there is modification in the Moodle the performance need to be checked. In order to address this issue Web Server Stress testing tool is designed.



3.1. Requirements change management

There are three principal stages to a change management process:

1. ***Problem analysis and change specification:*** The process starts with an identified requirements problem or, sometimes, with a specific change proposal. During this stage, the problem or the change proposal is analysed to check that it is valid. This analysis is fed back to the change requestor who may respond with a more specific requirements change proposal, or decide to with draw the request.

2. ***Change analysis and costing:*** The effect of the proposed change is assessed using traceability information and general knowledge of the system requirements. The cost of making the change is estimated both in terms of modifications to the requirements document and, if appropriate, to the system design and implementation. Once this

analysis is completed, a decision is made whether or not to proceed with the requirements change.

3. ***Change implementation:*** The requirements document and, where necessary, the system design and implementation, are modified. It needs to organize the requirements document so that it can make changes to document without extensive rewriting or reorganization. As with programs, changeability in documents is achieved by minimizing external references and making the document sections as modular as possible. Thus, individual sections can be changed and replaced without affecting other parts of the document

3.2. How a Browser Interacts with a Web Based LMS

When the Moodle user requests a URL (e.g. "http://example.com/moodle") the browser looks up the host in the address ("example.com") and the URL is resolved by the Domain name Server (DNS) and sends the request to that host. The host runs a web server which is listening for browser requests, when the web server receives the request it passes it to the LMS software which examines the address and composes and sends the response. The response is normally a HTML document that can contain addresses of other resources in the LMS. When the browser receives the HTML document it reads it and makes any additional requests for other resources (such as images) that are required to fully display the page to the user. The time between the user hitting "Go" and the page being displayed is the latency for the whole page, which is composed of the latency for each of the resources that the page requires. Because the browser doesn't know the resources required to display for a requested address, the latency for a whole page is greater than the individual resources.

3.3. Request/Response Lifecycle

The lifecycle of a request and response is illustrated in Figure 1 and the major actions in the figure are as follows:

1. The user request is sent across the network to the web server.

2. The web server receives the request and passes it to Moodle, Moodle then parses the request and determines the actions it needs to take to fulfill the request - normally this involves creating a HTML document from the result of many MySQL queries

3. MySQL parses the Moodle queries and searches its database (DB)
4. The MySQL DB files and static artifacts (e.g. images) are accessed via a file system
5. The file system is stored on a block device such as a hard drive

We will now examine each of these stages with the tools that report metrics that describe their state.

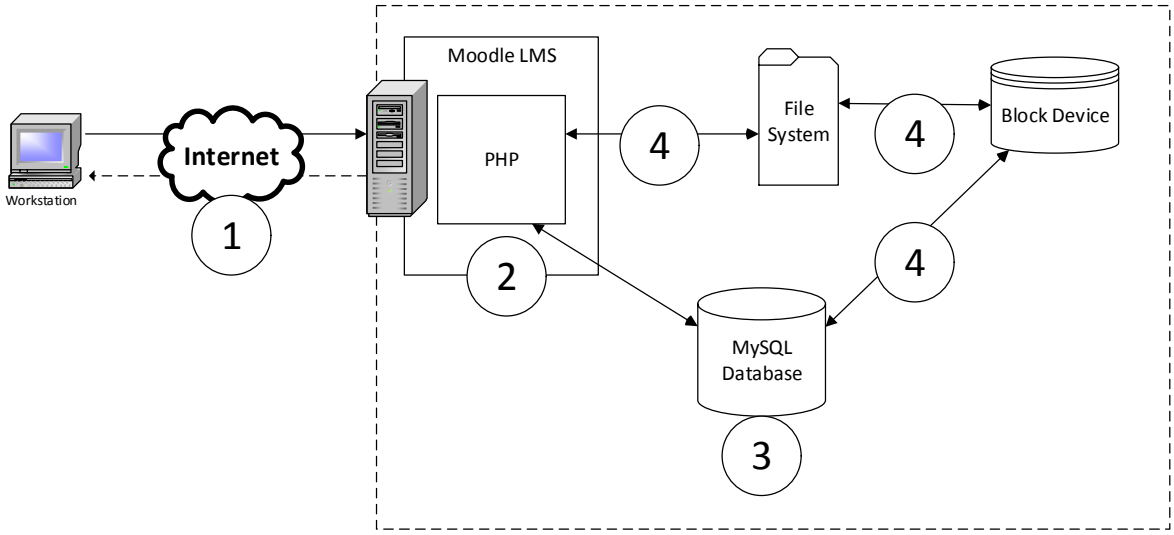


Figure:-3.2: User request and its lifecycle.

Before the LMS can respond to requests, those requests must first be transmitted over the network to the LMS and this introduces the first source of latency. It is difficult to evaluate the network performance as it's not normally possible to monitor the steps between the client and the LMS.

3.4. Proposing Model/Design

Webserver Stress Testing Tool is designed to simulate specified number of Moodle users using multi-threading technology and to get the response from the server. Webserver Stress Testing Tool monitors for a proper HTTP return code. By HTTP specifications RFC 2616, any web server returns several HTTP codes.

Analysis of the HTTP codes is the fastest way to determine the current status of the monitored web server. The monitoring process is largely dictated by the Webserver Stress Testing Tool, but there are certain elements that are likely to be found in most non-trivial monitoring systems.

Status code	Meaning
200	OK
201	Created
202	Accepted
204	No Content
301	Moved Permanently
302	Moved Temporarily
304	Not Modified
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable

Table-3.1: Web Server HTTP Return Codes

Depending on the response of the webserver programmers/QA people can decide whether the Moodle or LMS can withstand the specified number of users in the webserver stress testing tool.

3.5. Methodology

Webserver Stress Tool simulates the Moodle from few users to several hundred users accessing a website via HTTP/HTTPS simultaneously. To perform the load test it is planned to create a LMS using www.moodlecloud.com website and then populate real world sample data such as Mangers/Teachers/Students and courses etc. Then this LMS site can be used as the simulated environment with a set of virtual users by increasing the number of users gradually until the site become unresponsive.

Using this Webserver Stress Testing tool first it is decided to record the single user activities with Moodle for example user creation process, online quiz activity, online exams, some course work upload/download etc. Then the activities done by the single user is simulated by the Webserver Stress Testing Tool to mimics the high volume traffic condition for the Moodle.

In other wards this also can be done with the help of the development team of the Moodle to generate scripts to send the request to webserver from this tool. When generating scripts it is required to consider the much time consuming activities to achieve the real world worst-case behavior. Also this cannot be used for testing commercial websites because this requires knowledge about the

internal parameters of the Moodle to generate scripts depending on the web pages GET/POST parameters.

3.5.1. Use Case Diagram

Use case diagram is one of five diagrams in the UML for modeling the dynamic aspect of the systems. This make system and classes approachable and understandable by presenting outside view of how those elements may be used in context. Below use case diagram explains the main functionality of the Stress Testing Tool.

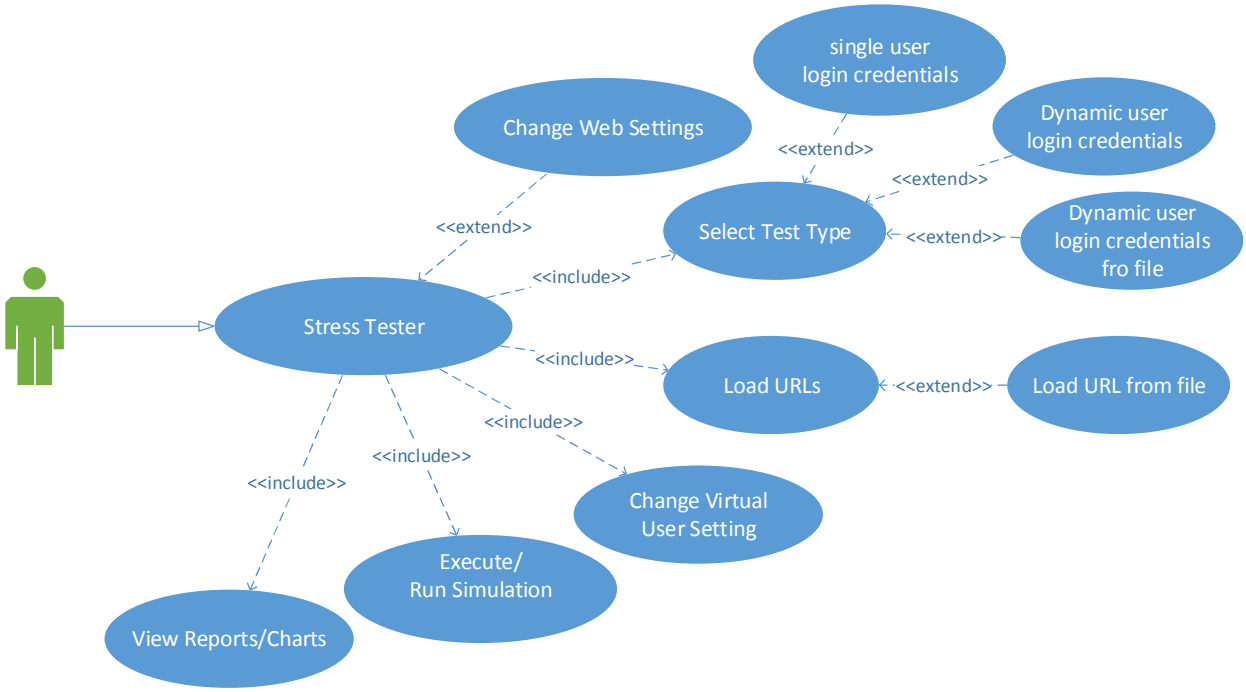


Figure:-3.3: Use Case Diagram for Stress Testing Tool

3.5.2. Class Diagram

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects. The class diagram is the main building block of object-oriented modeling. (Class diagram can be seen on APPENDIX-A)

- ***HttpWebUser Class***

The class diagram represents all the classes used for the system. There are three main classes used for this project. HttpWebUser class is the main class. Other two classes Webresponse and Websetting classes support the functionality of the HttpWebUser.

Main class.The HttpWebUserclass provides support for the properties and methods defined in HttpWebRequest class and for additional properties and methods that enable the user to interact directly with the Moodle server using HTTP. Use the WebRequest.Create method to initialize new HttpWebRequest objects. If the scheme for the Uniform Resource Identifier (URI) is http:// or https://, Create returns an HttpWebRequest object. (See APPENDIX-B)

- ***Webresponse Class***

The *HttpWebUser.ClickDynamicURL* method makes a synchronous request to the resource specified in the RequestUri property and returns an Webresponse that contains the response object. The response data can be received by using the stream returned by GetResponseStream. If the response object or the response stream is closed, remaining data will be forfeited. The remaining data will be drained and the socket will be re-used for subsequent requests when closing the response object or stream if the following conditions hold: it's a keep-alive or pipelined request, only a small amount of data needs to be received, or the remaining data is received in a small time interval. If none of the mentioned conditions hold or the drain time is exceeded, the socket will be closed. For keep-alive or pipelined connections, we strongly recommend that the application reads the streams until EOF. This ensures that the socket will be re-used for subsequent requests resulting in better performance and less resource used.

When you want to send data to the resource, the GetRequestStream method returns a Stream object to use to send data. The BeginGetRequestStream and EndGetRequestStream methods provide asynchronous access to the send data stream.

The HttpWebRequest class throws a WebException when errors occur while accessing a resource. The WebException.Status property contains a WebExceptionStatus value that indicates the source of the error. When WebException.Status is WebExceptionStatus.ProtocolError, the Response property contains the HttpWebResponse received from the resource.

HttpWebRequest exposes common HTTP header values sent to the Internet resource as properties, set by methods, or set by the system; the following table contains a complete list. You can set other headers in the Headers property as name/value pairs. Note that servers and caches may change or add headers during the request. (See APPENDIX-B)

- ***Websettings Class***

Webresponse Class maintains the Websetting of the HttpWebResponse Class. Setting can be retrieved using Getsetting() method and changed settings can be saved using save setting() method. (See APPENDIX-B)

3.5.3. C# - Multithreading

A thread is defined as the execution path of a program. Each thread defines a unique flow of control. If the application involves complicated and time consuming operations, then it is often helpful to set different execution paths or threads, with each thread performing a particular job. Threads are lightweight processes. One common example of use of thread is implementation of concurrent programming by modern operating systems. Use of threads saves wastage of CPU cycle and increase efficiency of an application.

A program which uses a single thread runs as a single process which is the running instance of the application. However, this way the application can perform one job at a time. To make it execute more than one task at a time, it could be divided into smaller threads. In C#, the *System.Threading.Thread* class is used for working with threads. It allows creating and accessing individual threads in a multithreaded application. The first thread to be executed in a process is called the main thread. When a C# program starts execution, the main thread is automatically created. The threads created using the Thread class are called the child threads of the main thread. You can access a thread using the CurrentThread property of the Thread class.

It is often use background threads when a time-consuming process needed to be executed in the background without affecting the responsiveness of the user interface. This is where a BackgroundWorker component comes into play. In C# a Background- Worker component executes code in a separate dedicated secondary thread while the main thread is still available to the user interface. (See Appendix C)

3.5.4. Thread Life Cycle

The life cycle of a thread starts when an object of the *System.Threading.Thread* class is created and ends when the thread is terminated or completes execution. Following are the various states in the life cycle of a thread:

- **The Unstarted State** − It is the situation when the instance of the thread is created but the Start method is not called.
- **The Ready State** − It is the situation when the thread is ready to run and waiting CPU cycle.
- **The Not Runnable State** − A thread is not executable, when
    - Sleep method has been called
    - Wait method has been called
    - Blocked by I/O operations
- **The Dead State** − It is the situation when the thread completes execution or is aborted.

3.6. File System

To save the responses I used saving files in the file system rather than saving it in a database. Here are the pros and cons involved in saving files in the file system.

3.6.1. Pros of the File System

- **Performance can be better than when you do it in a database**. To justify this, if you store large files in DB, then it may slow down the performance because a simple query to retrieve the list of files or filename will also load the file data if you used Select * in your query. In a files system, accessing a file is quite simple and light weight.
- **Saving the files and downloading them in the file system is much simpler** than it is in a database since a simple "Save As" function will help you out. Downloading can be done by addressing a URL with the location of the saved file.
- **Migrating the data is an easy process**. You can just copy and paste the folder to your desired destination while ensuring that write permissions are provided to your destination.

- **It's cost effective** in most cases to expand your web server rather than pay for certain databases.
- **It's easy to migrate it to cloud storage** i.e. Amazon S3, CDNs, etc. in the future.

3.6.2. Cons of the File System

- **Loosely packed**. There are no ACID (Atomicity, Consistency, Isolation, and Durability) operations in relational mapping, which means there is no guarantee. Consider a scenario in which your files are deleted from the location manually or by some hacking dudes. You might not know whether the file exists or not. Painful, right?
- **Low security**. Since your files can be saved in a folder where you should have provided write permissions, it is prone to safety issues and invites trouble, like hacking. It's best to avoid saving in the file system if you cannot afford to compromise in terms of security.

3.6.3. Use Cases

- **If your application is responsible for handling large files** (i.e. over 5MB) and the lots of file uploads.
- **If your application will have a large number of users**.

Chapter-4

Implementation

This chapter discusses the implementation of the system. Here it is clearly describes the development language, Operating system requirement, Major code and Module and Development Environment.

4.1. Development Language

Microsoft C#.net language of Visual Studio 2012 and .NET 4.5 was selected as the programming language due to several reasons. They are

- The main feature of this project is Multi-Threading Technology.
- Easy to use graphical user interfaces.
- Fully Object Oriented Language
- Easy exception Handling
- Easy file handling Technique

4.2. Operating System

The system was developed to run on Microsoft Windows 7/8/10 and server versions such as Microsoft Windows Server 2008/2012/2016 which is capable of installing Microsoft .Net framework 4.5 or above.

4.3. Major code and Module Structure

4.3.1. Background worker Implementation

A basic Windows application runs on a single thread usually referred to as UI thread. This UI thread is responsible for creating/painting all the controls and upon which the code execution takes place. So when you are running a long-running task such as sending set of URLs request to a webserver and waiting for a replying, downloading data from a website and deleing large number of the UI thread locks up and the UI application turns white (remember the UI thread was responsible to paint all the controls) rendering the application to Not Responding state.

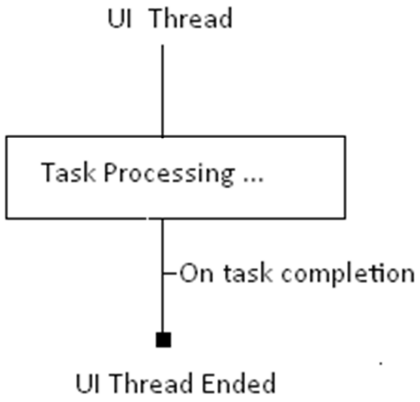


Figure:-4.1. Shows the UI thread

Using a Backgroundworker the burden of the heavy processing can be shifted to different thread. Leave the UI thread free for painting the UI. C#.NET has made the BackgroundWorker object available in order to simplify threading. This object is designed to simply run a function on a different thread and then call an event on your UI thread when it's complete.

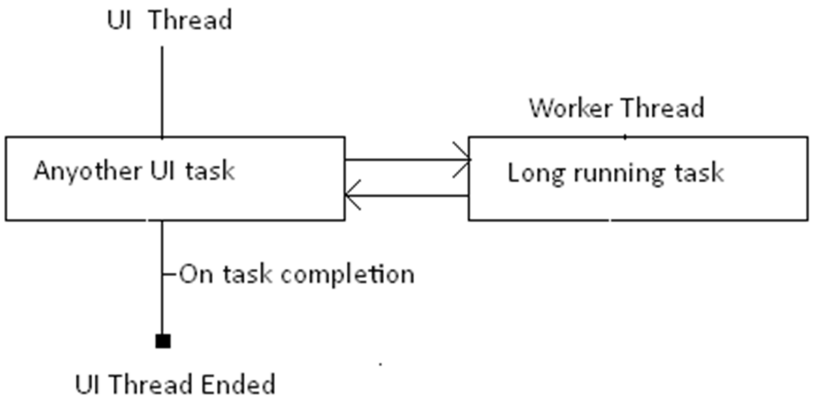


Figure:-4.2. UI thread with *BackgroundWorker*

Below is the sample code which is used to execute the virtual web users in this application.

4.3.2. VirtualUserHandler

Response received by the HttpWebUser objects are processed and displayed in the datagrid view and Treeview using this method. (See HttpWebUser Class APPENDIX-B)

4.3.3. Response Time Graph

All the response data loaded in to the grid using VirtualUserHandler is used to draw the graph. Below is the source code of the response time graph.

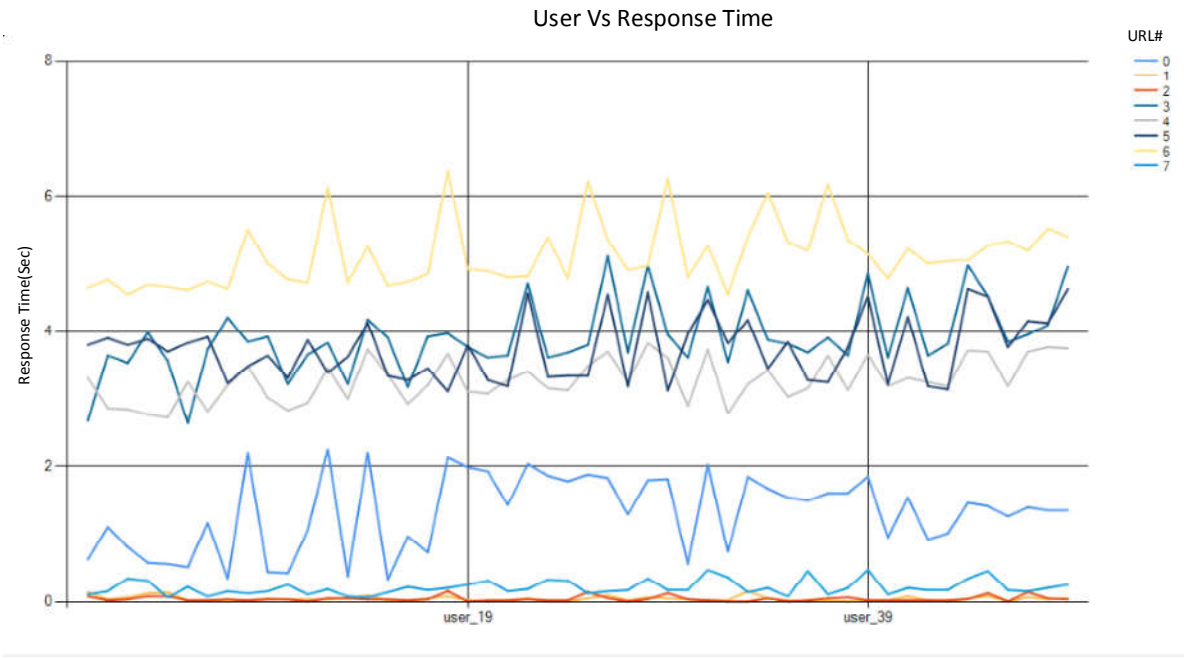


Figure:-4.3: User Vs Response Time Graph for each URL

4.3.4. Other Reusable Components

All the source code of classes and other modules implementations cannot be included here as it increases the page limit therefore please refer Appendix B.

4.4. Development Environment

4.4.1. Hardware Requirement

The minimum hardware requirement of this application is depend on the no of virtual users need to be simulated by this tool. It is recommended to have minimum of following Hardware requirement.

Intel Core i3 or above

8GB Memory or above

500 GB Hard disk (optional)

4.4.2. Software Requirement

Microsoft .net frame work 3.5

Chapter-5

Evaluation

5.1. Expected functionality

The main functionality of this project is to simulation of the Moodle user in order check the Maximum stress or the load which can handle by the server with the existing infrastructure. Following is the sequence diagram in order to show the core functionality of the project. This application simulates n number of threads starting from one and executes given number of web requests. Each thread represents a single user in the real world scenario.

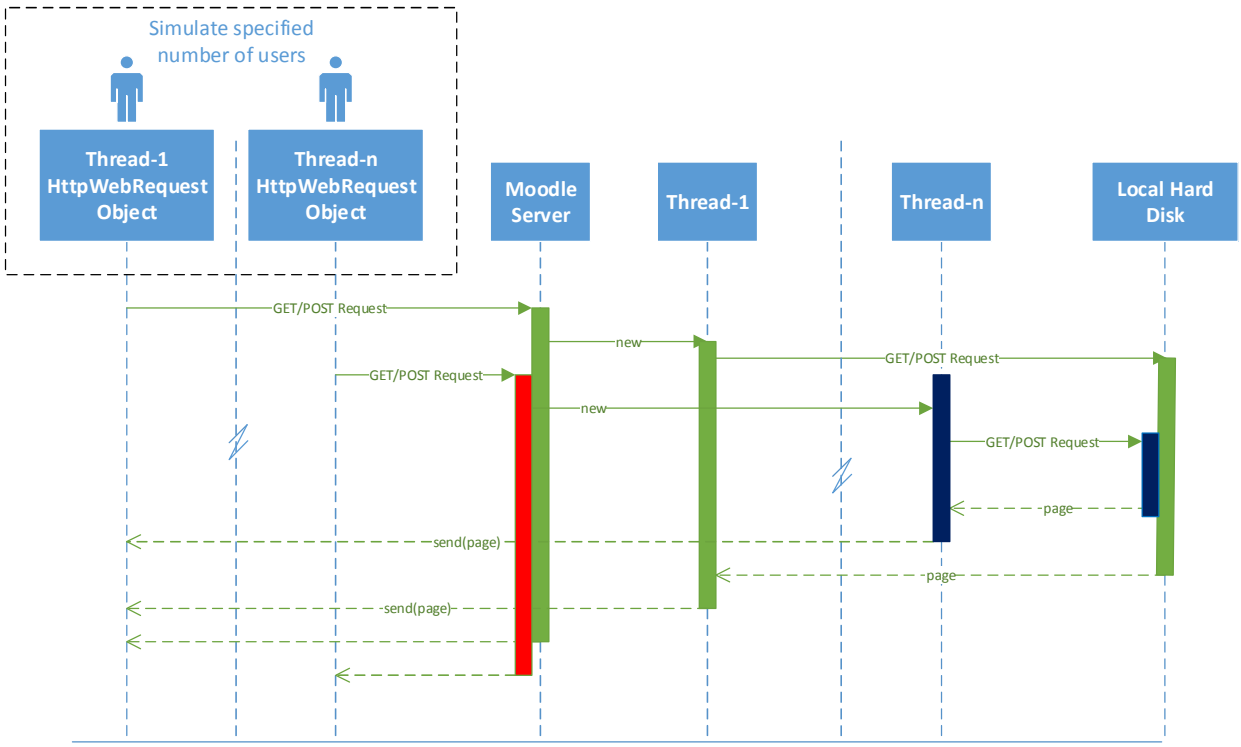


Figure:-5.1: sequence diagram of the main functionality

Then each response from the users is obtained and display in a data grid view. Also the response time is calculated for each thread.

5.2. HttpWebuser Class

HttpWebUser class is the main class which is designed to represent each thread or virtual user in this project. When executing each thread HttpWebUser class instantiated and name is given to

the each user in order to keep a track of the each user. Using of Httpuser class increased the overall efficiency of the application. By creating virtual users in using BackgroundWorker class helped to eliminate the UI freezing issue. (See APPENDIX-B)

5.3. Sample Moodle site

Developed a sample website to test the Moodle Stress Testing Tool as it is not ethical and illegal to use a commercial websites since the load testing is similar to brute force attack. It consists of features such as user login, user creation and sample quizzes creation, sample quiz execution etc. This test Moodle site developed using php and mysql as the database server.

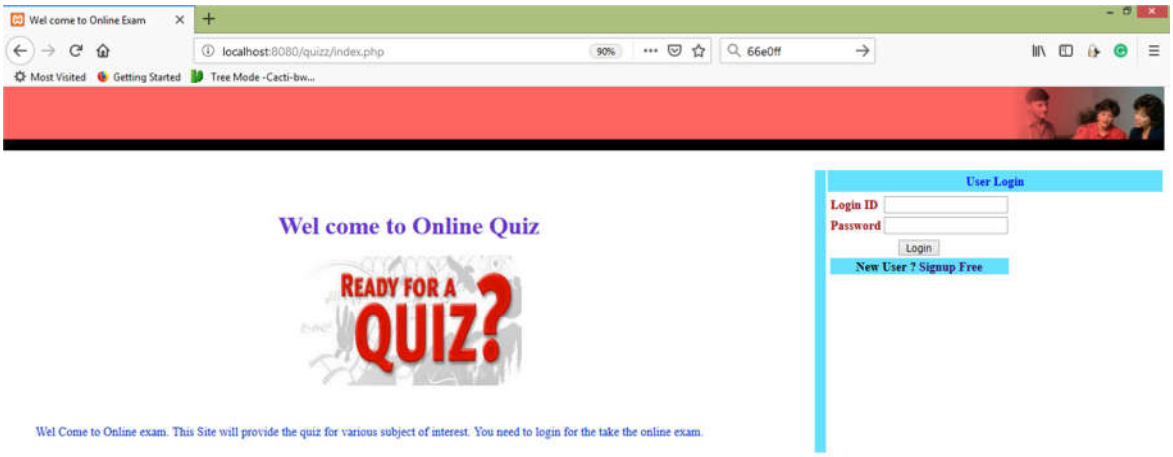


Figure:-5.2 show the home page of the Sample Moodle site.

5.4. Program Execution

This thesis discusses four types of stress tests. User can do the simulation by using specified no of virtual users with

(1). Single user login credential.

(2). Multiple user logins credentials system generated users.

(3). Multiple user logins credentials provided by a CSV file.

(4). several load engines with multiple user login credentials.

First three tests can be done using a single load engine by simply changing test types in the Stress testing Tool. It categorized depending on the virtual user authentication method.

The fourth one Simulation with several load engines with multiple user login credentials need several load engines i.e. separate PC with stress testing tool installed.

5.4.1. Simulation with Single user login credential

URL Click by Single User login credentials means the virtual users are created using single login credentials. The main disadvantage of this method is by using this method some tasks such as completing a quiz or some other tasks which are allowed to do once cannot be simulated. That is because when the particular task activated is no longer available to execute another time. So simulation is not possible using single login credentials. Test type is selected as follows.

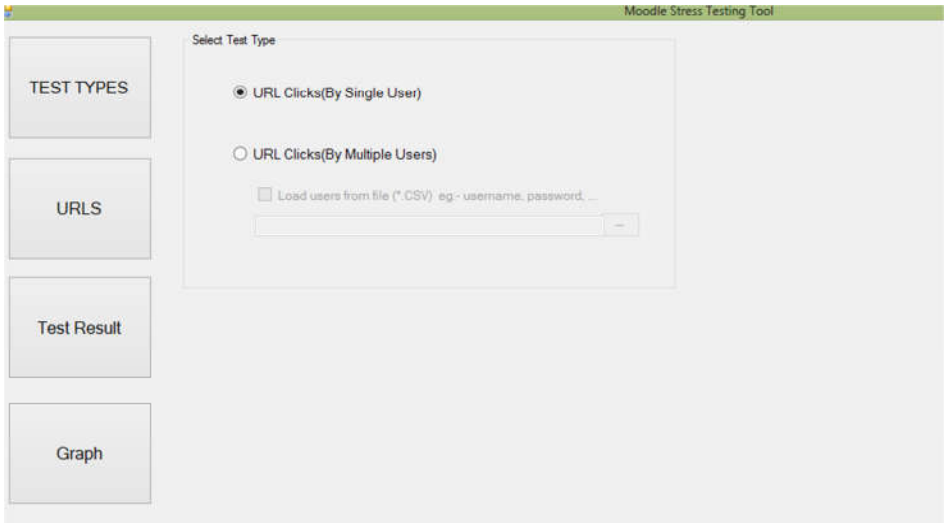


Figure:-5.3: Test Types selection interface

URLs can be added using Add URL button is used to add URLs one by one. If the URL uses POST method Post Data can be sent with the request as Key/value pair. User interface support to enter the post parameters required for URL post request. If the POST Vars added to the Listview and Add URL button is clicked it inserted to the URL DataGridview (List of URLs) needs to be executed. Load button can be used to add list of URLs from a CSV file.

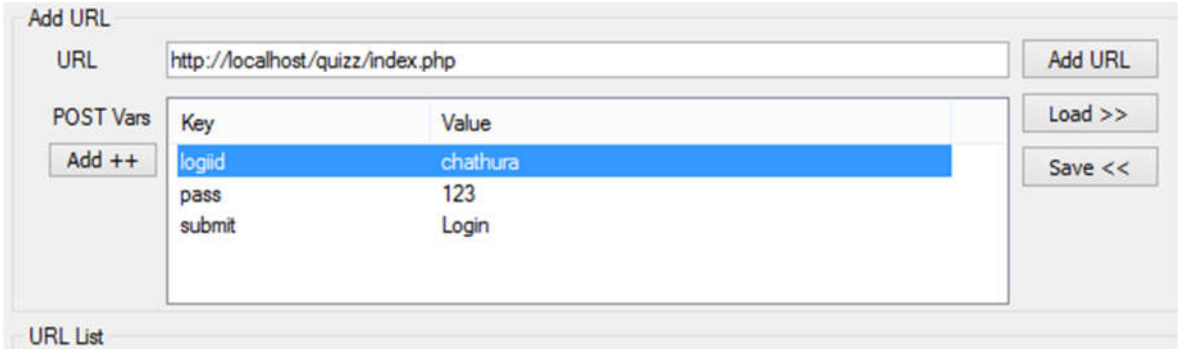


Figure:-5.4: POST request using key/value pair

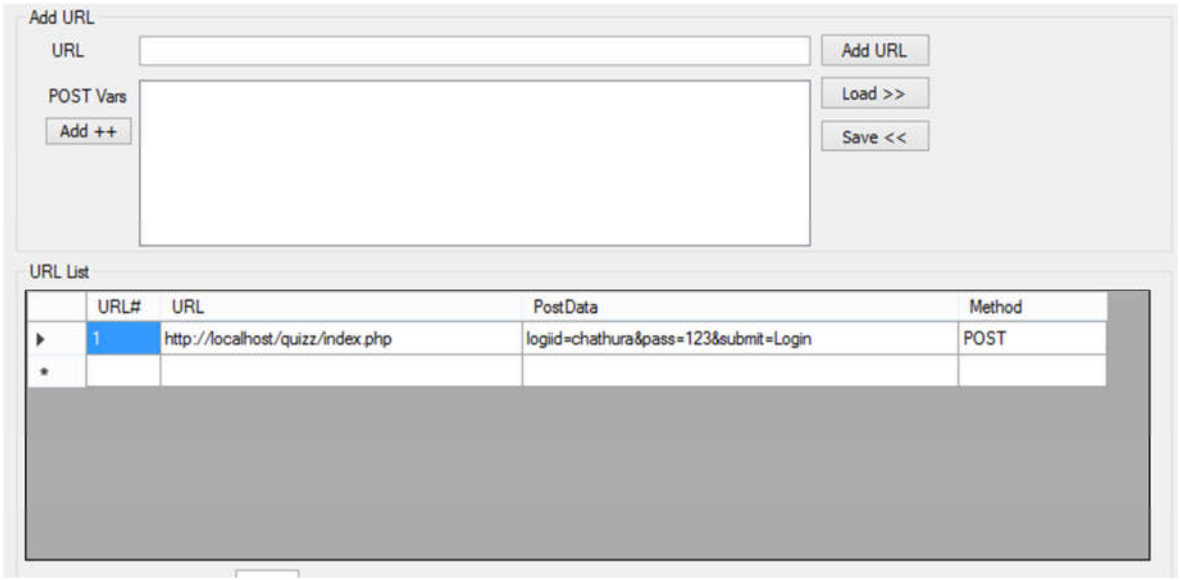


Figure:-5.5: POST request inserted to the URL List

CSV file can be created directly using Microsoft Excel or as comma separated text using a Text Editor. Also it can be created by adding URLs either GET or POST method to the system one by one as mentioned above and saving it to reuse for future testing purpose.

	A	B	C	D
1	URLNo	URL	POST DATA	METHOD
2	1	http://localhost:8080/quizz		GET
3	2	http://localhost:8080/quizz/index.php	loginid=chathura&pass=123&submit=Login	POST
4	3	http://localhost:8080/quizz/sublist.php		GET
5	4	http://localhost:8080/quizz/showtest.php?subid=1		GET
6				

Figure:-5.6: format of the CSV file need to be loaded

Once loaded to the system it shows as follows. When the GO button executes each URL is processed by each single thread or the user. This feature helps to maintain Test cases for future usage Moodle Testing.

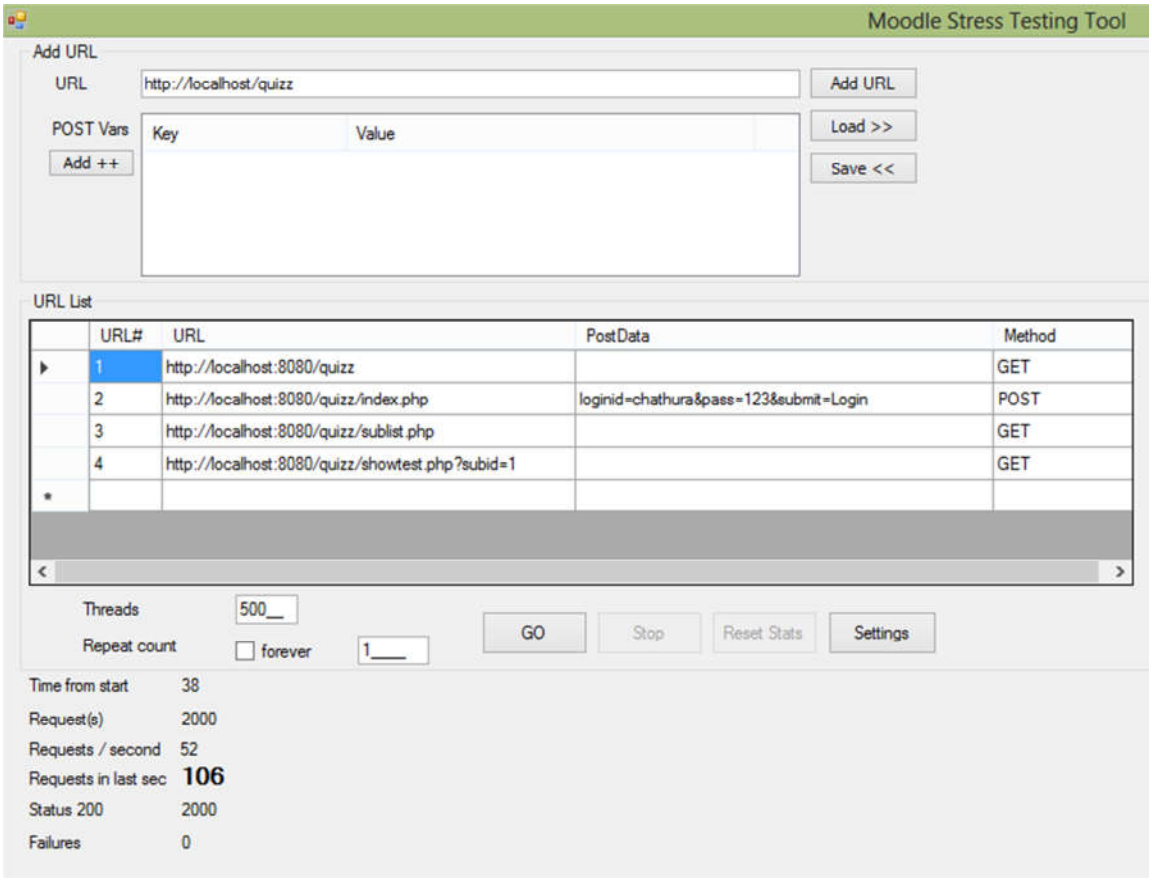


Figure: -5:7: After Loading the CSV file to the system

5.4.2. Multiple user logins credentials system generated users.

URL click by Multiple Users can be done in two ways.

- By using System generated user logins
- By using user credentials provided by .CSV file.

By using System generated user logins is the fastest ways of doing a Stress Test. This is done by dynamically creating users in the Moodle and simulation done by using those separate user login credentials.

Here user creation is automated by changing the URL data dynamically. Post data is sent lid=user#i# ... #i# means i change with the given number of users 1..n and creates n users accounts or logins.

Eg: lid=user#i#&pass=user123&cpass=user123&name=username#i# &address=useraddress&city=user_city&phone=0772516004&email=user@gmail.com&Submit=Signup

In this method main purpose is to create separate user logins therefore only the user name is changed in each user. Then the set of URLS need to be simulated to test particular module is done by using dynamically changing URLs. Eg: user login process is simulated by following URLs

loginid=user#i#&pass=user123&submit=Login. (#i# represents the 1,2,3…….n)
loginid=user**1**&pass=user123&submit=Login
loginid=user**2**&pass=user123&submit=Login

| | | | |

| | | | |

loginid=user(**n-1)**&pass=user123&submit=Login
loginid=user**n**&pass=user123&submit=Login

5.4.3. Multiple user logins credentials provided by a CSV file.

By using CSV file provided by the user is somewhat time consuming as it requires time to create the CSV file. Post parameters need to be inserted to the URL as follows.

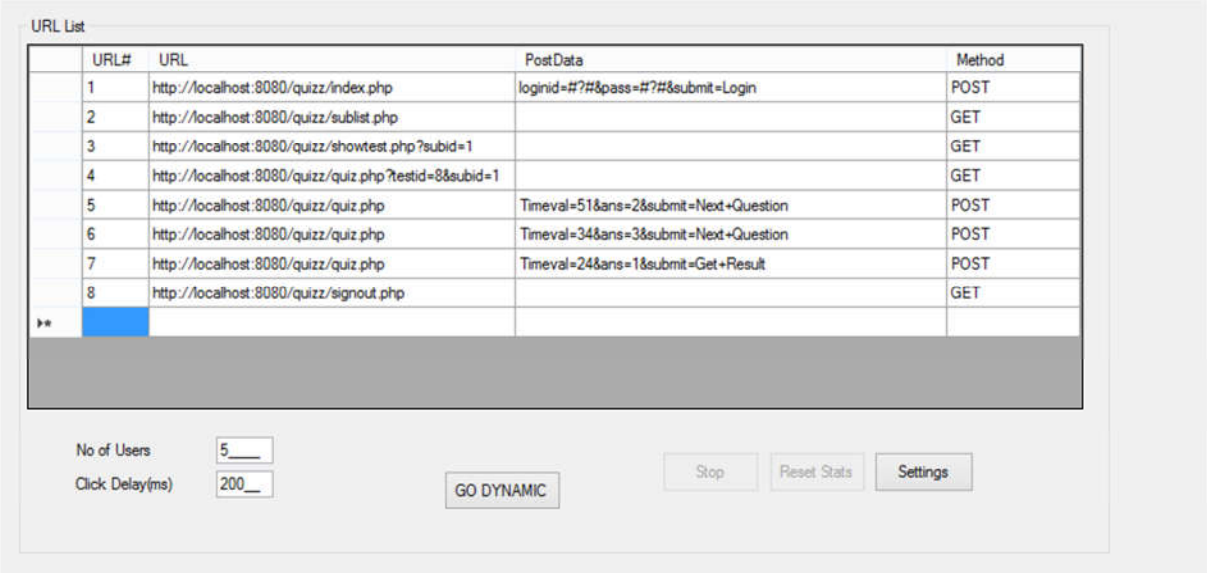


Table.-5.1. URL request with POST DATA

#?# is replaced by the post data provided by the CSV file. No of Users automatically obtained by no of users in the CSV file e.g. no of users in the CSV file is 5 then the No of Users to be simulated is 5. In this case the data in the CSV file need to be in the following format since the post data contains only two parameters.

```
saman,1234
ashish,shah
Dhaval123,a
chathura,123
raj,raj
```

5.4.4. Several load engines with multiple user login credentials.

This can be done by either using scenario-2 or scenario-3 with different test cases by using several load engines. This is somewhat similar to the real user behavior.

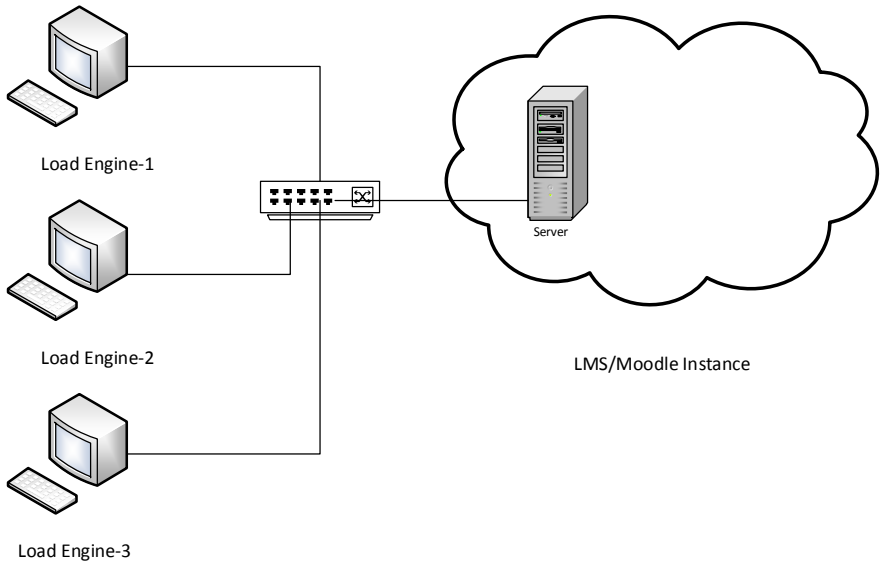


Figure:-5.8: Simulation with several load engines

5.4.5. How each thread Executes

When "GO" button pressed each thread starts executing. The number of threads can be specified depending on the hardware configuration PC where the Moodle Server Stress Testing Tool is installed. If the both webserver and the Moodle Server Stress Testing Tool both are installed in the same machine it is difficult to get a useful result from the load test. After the execution is done application calculates the following details such as Time Taken, Total Number of Request, Request per second, Request in last second , status of the response, number of failures and response time.

Results also can be viewed in tree view which shows the Response Header and the Response data stream. If the response is type="Text" it will be displayed in the Text box. Tree View node green implies the response is successfully completed and node red implies the failure.

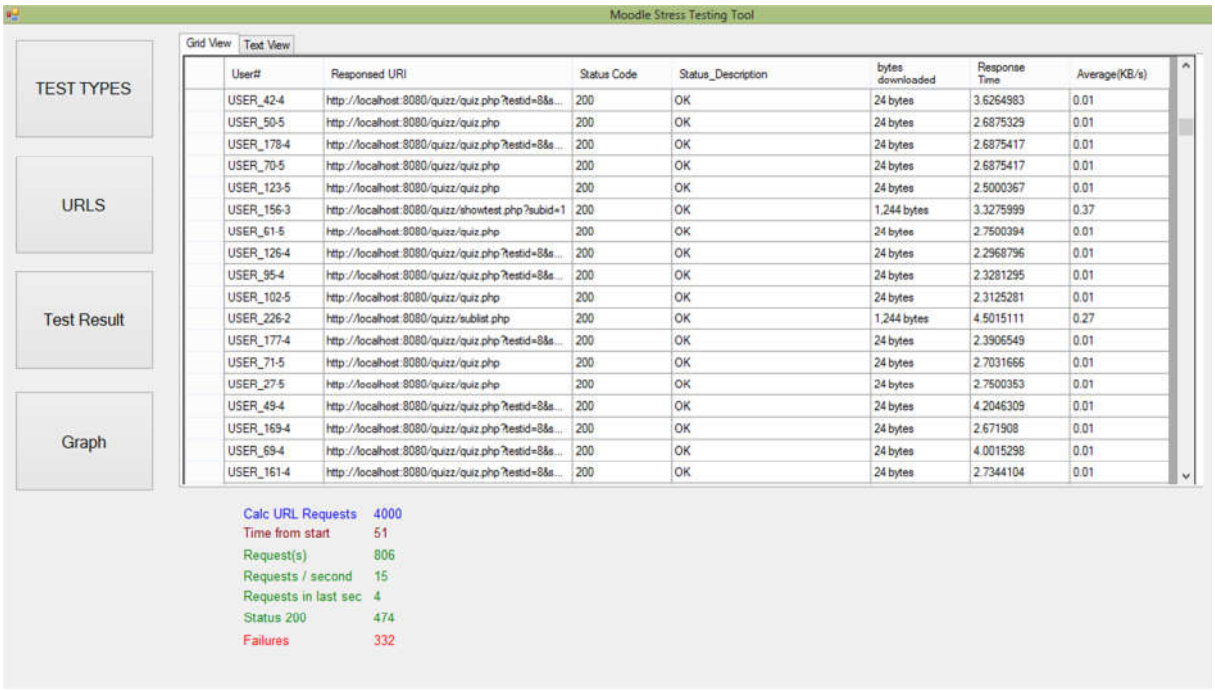


Figure:-5.9: Application Interface after executing 500

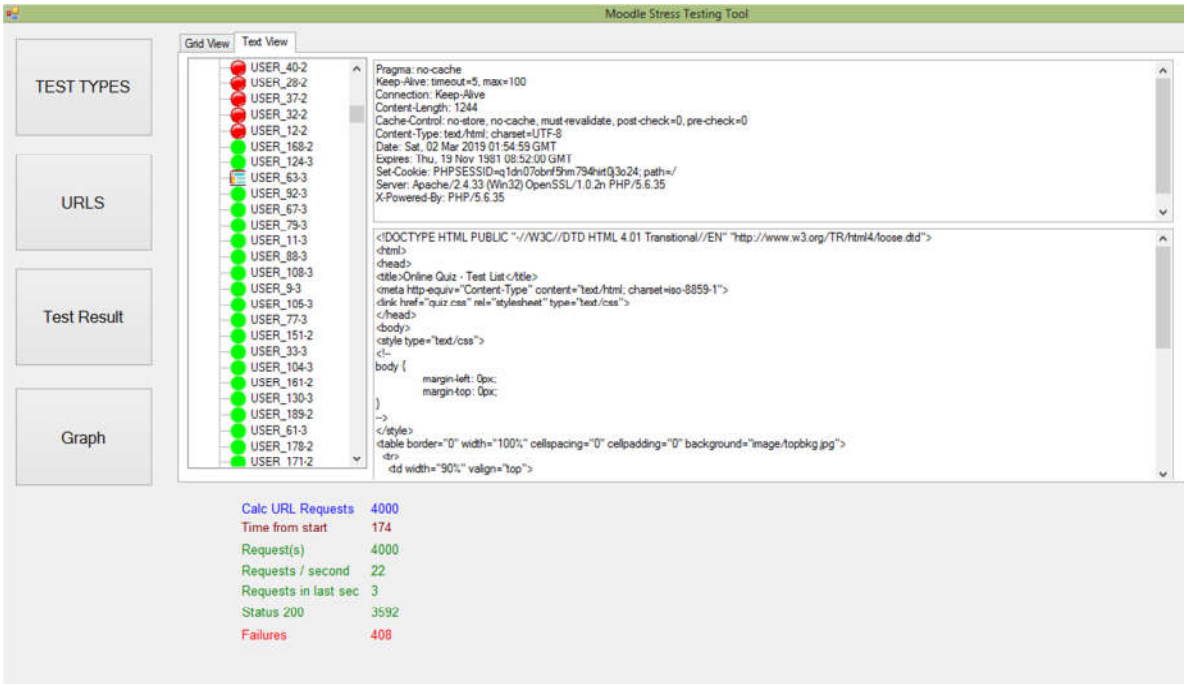


Figure:-5.10: Output shown from the tree view

5.5. Server Performance Monitoring

Netdata is used to monitor server side real-time system metrics monitoring. Monitors and renders various system metrics in real time, such as CPU, memory, disk I/O, network traffic, system processes, Apache/Nginx status, MySQL status, Postfix message queue, and others. It is highly optimized to use minimal CPU, memory, and disk I/O. Provide stunning real-time metrics graphics in an intuitive web interface. Netdata can be installed simply on centos 7 using following list of commands ref: (https://www.vultr.com/docs /installing-netdata-on-centos-7)

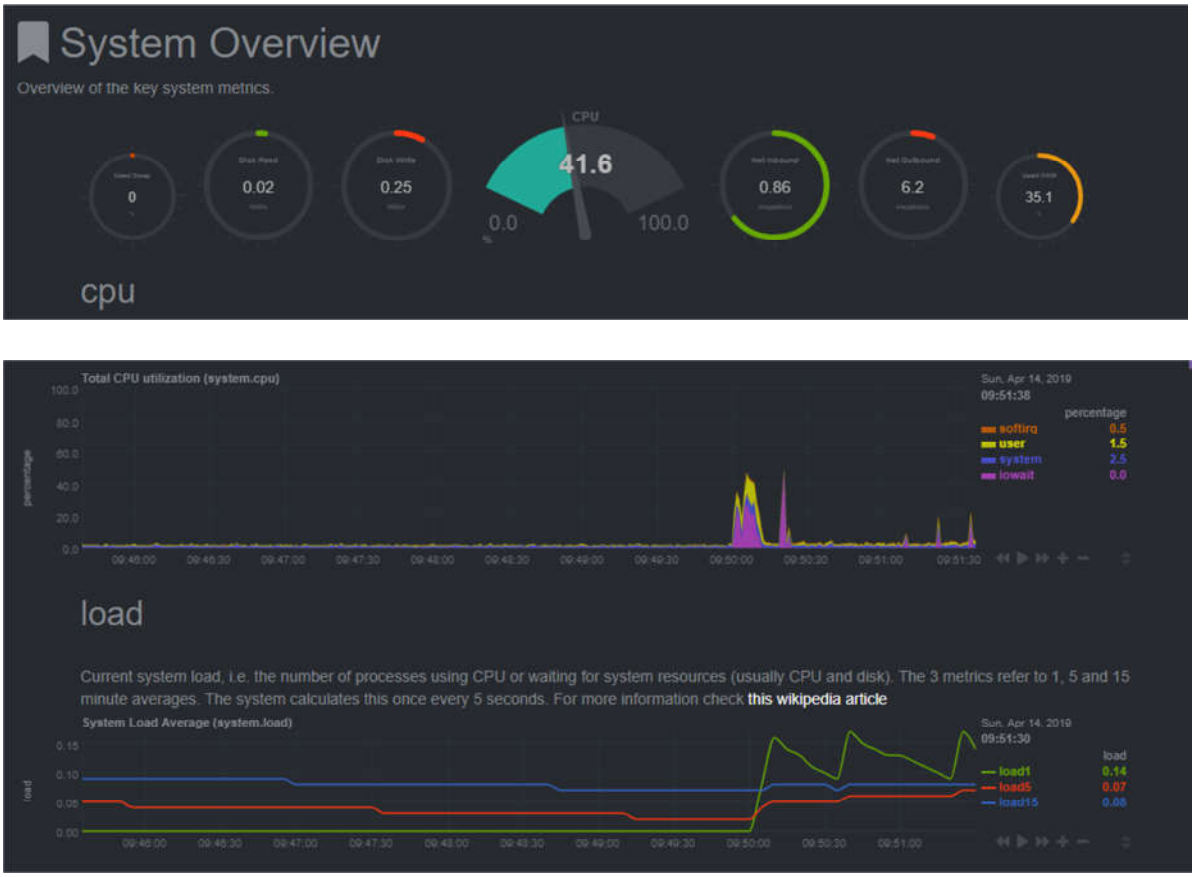


Figure:-5.11: NetData Monitoring tool dashboard

Chapter-6

Conclusion

The main objective of this Project was to develop a tool to check the performance of a Moodle in order to performance issue from the development stage to the implementation of the final product. Actually most of the cases during the development stage performance are not being tested only the unit testing is done by the developer and QA team. By using this tool it become easier to perform stress testing in order to deduct the cost of maintenance due to performance issue, also it allows both the detection and resolution of problems and the proactive prevention of problems caused by increased usage of the LMS.

6.1. Achievements

- ✓ Maintain URLs for future use such as save list of URL/Load URLs from a file/Delete URLs from the specific test case.
- ✓ 3-Types of Test Available depending on either single login credential or multiple user login credentials.
- ✓ The test results can be saved for future use.
- ✓ Simple and user friendliness.

6.2. Future Work

This system is developed using Object Oriented Programming Techniques therefore it is easy to enhance this application easily. This application can be further improved by using by executing a range of users from lesser number to a large number of users and application provides the optimal no of users which the server can handle.

Reference

[1] Sorin POPA, Associate Professor, PhD," Web Server Monitoring", University of Craiova

[2] Jonathan Barber, Rodolfo Matos, Susana Leitão," Moodle Monitoring Best Practices", University of Porto (PORTUGAL)

[3] Analyzing server response time using Testing Power Web Stress tool, [Online]. Available:https://ieeexplore.ieee.org/document/5773700/

[4] Apache Jmeter, [Online]. Available:

http://jmeter.apache.org/download_jmeter.cgi

[5] Locust, [Online]. Available:

https://locust.io/

[6] Webserver Stress Tool, [Online]. Available:

https://www.paessler.com

[7] Apache Log Analyzer, [Online]. Available:

https://www.manageengine.com/products/eventlog/apache-web-server-log-analyzer.html

[8] I teach with Moodle

http://www.iteachwithmoodle.com/2012/10/10/3-free-tools-to-test-if-your-moodle-server-can-cope-with-large-amounts-of-students/

[9] File System vs. Database

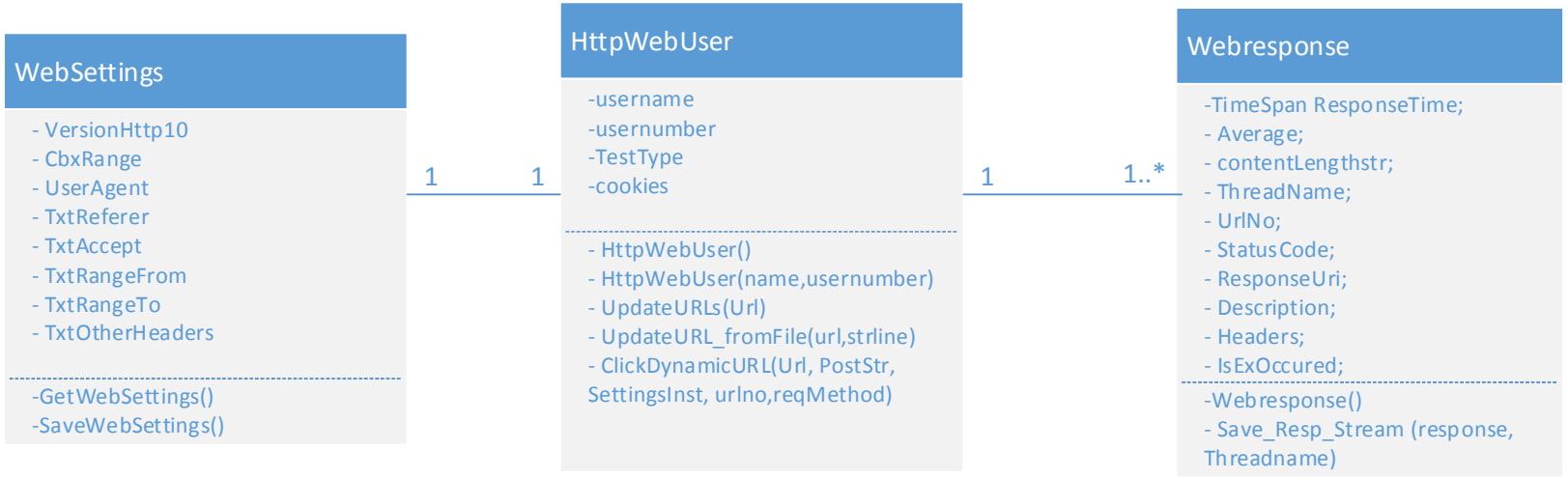
https://dzone.com/articles/which-is-better-saving-files-in-database-or-in-fil

[10] Background Worker Class

https://docs.microsoft.com/en-us/dotnet/api/system.componentmodel.backgroundworker?view=netframework-4.7.2

APPENDIX-A: Design Documentation

Class Diagram of Stress Testing Tool



APPENDIX-B: Code Listing

HttpWebUser Class

```
using System;
usingSystem.Text;
usingSystem.Windows.Forms;
using System.Net;
usingSystem.Web;
using System.IO;



namespace MOODLE_STRESS_TOOL
{

classHttpWebUser
    {
privatestring _Uname = "";
privateint _Usernumber=0;
staticintNo_of_users;

privateTestTypes _Testtype;
privateCookieContainer cookies = newCookieContainer();

publicHttpWebUser()
        {
            _Uname = "";
            _Usernumber = 0;
        }

publicHttpWebUser(stringUname, intUsernumber)
        {
            _Uname = Uname;
            _Usernumber = Usernumber;
        }

publicTestTypesTesttype
        {
set
            {
                _Testtype = value;
            }

get
            {

return _Testtype;
            }
        }
publicstringUname
        {
set
            {

                _Uname = value;
            }

get
```

```
            {

return _Uname;
            }
        }

publicintUsernumber
        {
set
            {
                _Usernumber = value;
            }

get
            {

return _Usernumber;
            }
       }

privatestringUpdateURL_fromFile(stringUrl, stringstrLine)
        {
if (strLine == null) returnnull;

intindx, startpos, arrPos;
string[] str1 = strLine.Split(',');
string newStr1 = "", newStr2 = "", newStr = "";
indx = Url.IndexOf("#?#", 0);
arrPos = 0;
newStr = Url;
if (indx> 0)
            {
startpos = 0;
while (indx> 0)
                {
                    newStr1 = newStr.Substring(0, indx);
                    newStr2 = newStr.Substring(indx + 3);
startpos = indx + 3;
newStr = newStr1 + str1[arrPos].ToString().Trim() + newStr2;
indx = newStr.IndexOf("#?#", startpos);
arrPos++;
                }
returnnewStr;
            }
else
            {
returnUrl;
            }
        }

publicstringUpdateURLs(StringUrl)
        {
intindx, startpos;
string newStr1 = "", newStr2 = "", newStr = "";
indx = Url.IndexOf("#i#", 0);

newStr = Url;
if (indx> 0)
            {
startpos = 0;
while (indx> 0)
                {
```

```
                    newStr1 = newStr.Substring(0, indx);
                    newStr2 = newStr.Substring(indx + 3);
startpos = indx + 3;
newStr = newStr1 + Usernumber.ToString().Trim() + newStr2;
indx = newStr.IndexOf("#i#", startpos);

                }
returnnewStr;
            }
else
            {
returnUrl;
            }
        }

publicstaticlongSave_Resp_Stream(HttpWebResponse response, stringtmpThreadName)
        {
Streamrcvstream = response.GetResponseStream();
byte[] respBytes = newbyte[256];
intbyteCount;


stringrespContent, TmpFilename;
TmpFilename = Path.GetTempPath() + tmpThreadName + ".tmp";

FileStream fs = newFileStream(TmpFilename, FileMode.Create, FileAccess.Write);
do
            {
byteCount = rcvStream.Read(respBytes, 0, 256);
fs.Write(respBytes, 0, byteCount);
fs.Flush();
            } while (byteCount> 0);


fs.Close();

response.Close();
rcvStream.Close();

FileInfo info = newFileInfo(TmpFilename);


if (response.ContentType.StartsWith("text/"))
            {
StreamReader reader = File.OpenText(TmpFilename);
respContent = reader.ReadToEnd();
reader.Close();
respContent = respContent.Replace("\r\n", "\n");
respContent = respContent.Replace("\n", "\r\n");

            }
else
            {
respContent = "Non-text response received. Right-click and choose 'Save' to save the
response content.";
            }

returninfo.Length;
        }

publicWebresponseClickDynamicURL(stringUrl, stringPostStr, WebsettingsSettingsInst,
inturlno, stringreqMethod = "GET", stringpostDataStr=null)
```

```
        {
Webresponseurlresponse = newWebresponse();

HttpWebResponse Response=null;

longcontentLength = 0;
doubleavg = -1;
DateTimerequestStart=DateTime.Now;

HttpWebRequest request = (HttpWebRequest)WebRequest.Create(Url);

request.KeepAlive = true;
request.Method = reqMethod;
request.AllowAutoRedirect = true;
request.CookieContainer = cookies;

try
            {
if (reqMethod == "POST")
                {
request.ContentType = "application/x-www-form-urlencoded";

if (Testtype==TestTypes.DYNAMIC)
PostStr=UpdateURLs(PostStr);
elseif (Testtype==TestTypes.MULTIPLE)
PostStr=UpdateURL_fromFile(PostStr,postDataStr);


byte[] postDataBytes = postDataBytes = Encoding.UTF8.GetBytes(PostStr);
request.ContentLength = postDataBytes.Length;

StreampostDataStream = request.GetRequestStream();
postDataStream.Write(postDataBytes, 0, postDataBytes.Length);
postDataStream.Close();
                }


if (SettingsInst.VersionHttp10)
request.ProtocolVersion = HttpVersion.Version10;

// request.AllowAutoRedirect = CbxAutoRedirect.Checked;
request.UserAgent = SettingsInst.UserAgent;
// request.CookieContainer = CookieCont;

if (SettingsInst.TxtReferer.Length> 0)
request.Referer = SettingsInst.TxtReferer;

if (SettingsInst.TxtAccept.Length> 0)
request.Accept = SettingsInst.TxtAccept;

if ((SettingsInst.CbxRange) && (SettingsInst.TxtRangeFrom + SettingsInst.TxtRangeTo>
0))
                {
if (SettingsInst.TxtRangeFrom == 0)
request.AddRange(SettingsInst.TxtRangeTo * -1);
elseif (SettingsInst.TxtRangeTo == 0)
request.AddRange(SettingsInst.TxtRangeFrom);
else
request.AddRange(SettingsInst.TxtRangeFrom, SettingsInst.TxtRangeTo);
                }
```

```
// custom additional headers
if (SettingsInst.TxtOtherHeaders.Length> 0)
                {
string[] lines = SettingsInst.TxtOtherHeaders.Split('\n');
foreach (string line in lines)
                    {
intpos = line.IndexOf(':');
if (pos != -1)
                        {
string header = line.Substring(0, pos).Trim();
stringheaderValue = line.Substring(pos + 1).Trim();
try
                            {
request.Headers.Add(header, headerValue);
                            }
catch (ArgumentException)
                            {
MessageBox.Show("ERROR: The header '" + header + "' has to be set explicitly using one
of the properties.    ");
                            }
                        }
                    }
                }


Response = (HttpWebResponse)request.GetResponse();
TimeSpandur = DateTime.Now - requestStart;
urlResponse.IsExOccured = false;
urlResponse.ThreadName = this.Uname + "-" + urlno.ToString();
contentLength = Save_Resp_Stream(Response, urlResponse.ThreadName);

urlResponse.contentLengthstr = contentLength.ToString("#,###,###") + " bytes";

if (dur.TotalSeconds> 0)
avg = Math.Round(((double)contentLength / (double)1024) / (double)dur.TotalSeconds,
2);

urlResponse.Average = avg;
urlResponse.ResponseTime = dur;

urlResponse.StatusCode = Response.StatusCode;
urlResponse.ResponseUri = Response.ResponseUri.ToString();
urlResponse.Description = Response.StatusDescription.ToString();
urlResponse.Headers = Response.Headers.ToString();

    }
catch (HttpException e1)
            {

TimeSpandur = DateTime.Now - requestStart;
urlResponse.IsExOccured = true;
urlResponse.ThreadName = this.Uname + "-" + urlno.ToString();
urlResponse.Description = e1.Message.ToString();
urlResponse.contentLengthstr = contentLength.ToString("#,###,###") + " bytes";
urlResponse.StatusCode = HttpStatusCode.ServiceUnavailable;
urlResponse.ResponseUri = Response.ResponseUri.ToString();
urlResponse.Description = e1.Message.ToString();

urlResponse.Average = avg;
urlResponse.ResponseTime = dur;

            }
```

```
catch (WebException e2)
            {
TimeSpandur = DateTime.Now - requestStart;
urlResponse.IsExOccured = true;
urlResponse.ThreadName = this.Uname + "-" + urlno.ToString();
urlResponse.Description = e2.Message.ToString();
urlResponse.contentLengthstr = contentLength.ToString("#,###,###") + " bytes";
urlResponse.StatusCode = HttpStatusCode.ServiceUnavailable;
urlResponse.ResponseUri = Url;
urlResponse.Description = e2.Message.ToString()+"["+e2.Message.ToString()+"]";
urlResponse.Average = avg;
urlResponse.ResponseTime = dur;
            }
returnurlResponse;
            }
        }
    }
```

Webresponse Class

```
using System;
using System.Net;

namespace MOODLE_STRESS_TOOL
{
publicclassWebresponse
    {
publicTimeSpanResponseTime;
publicdouble Average;
publicstringcontentLengthstr = "";
publicstringThreadName="";
publicintUrlNo;
publicHttpStatusCodeStatusCode;
publicstringResponseUri = "";
publicstring Description = "";
publicstring Headers = "";
publicBooleanIsExOccured=false;

publicWebresponse()
        {
ResponseTime=TimeSpan.Parse("00:00:00");
            Average=0;
contentLengthstr = "";
ThreadName = "";
UrlNo=0;
StatusCode = HttpStatusCode.NotFound;
ResponseUri = "";
            Description = "";
            Headers = "";
IsExOccured = false;
        }
    }
}
```

Websetting Class

```
using System;

namespace MOODLE_STRESS_TOOL
```

```
{
classWebsettings
    {
publicBoolean VersionHttp10 = false;
publicBooleanCbxRange = false;
publicstringUserAgent="";
publicstringTxtReferer="";
publicstringTxtAccept="";
publicintTxtRangeFrom = 0;
publicintTxtRangeTo = 0;
publicstringTxtOtherHeaders = "";


    }
}
```

To Maintain Test Case following source codes are used

```
// To add URLs Manually to URL Data Grid View

privatevoidbtnAddURL_Click(object sender, EventArgs e)
        {
URLRowCount++;

if (LvwPostVars.Items.Count> 0)
            {

StringBuilderpostData = newStringBuilder();
foreach (ListViewItem item inLvwPostVars.Items)
                {
                     postData.AppendFormat("{0}={1}&",
                     HttpUtility.UrlEncode(item.SubItems[0].Text),
                     HttpUtility.UrlEncode(item.SubItems[1].Text));
                }
postData.Remove(postData.Length - 1, 1);

LvwPostVars.Clear();
GridViewURL.Rows.Add(URLRowCount, TextURL.Text, postData.ToString(), "POST");

            }
else
GridViewURL.Rows.Add(URLRowCount, TextURL.Text, "", "GET");

TextURL.Text = "";
        }

// To add post data as Key/Value Pair
privatevoidBtnAddPostVar_Click(object sender, EventArgs e)
        {

KeyValueFormdlg = newKeyValueForm();
if (dlg.ShowDialog() == DialogResult.OK)
            {
LvwPostVars.Items.Add(newListViewItem(newstring[2] { dlg.TxtKey.Text,
dlg.TxtValue.Text }));
LvwPostVars.Focus();
            }
        }
```

```
// Load Test Saved Test Case to GridviewURL
privatevoidBtnLoadPostVars_Click(object sender, EventArgs e)
        {
OpenFileDialogdlg = newOpenFileDialog();
dlg.Filter = "CSV Files (*.csv)|*.csv";
if (dlg.ShowDialog() == DialogResult.OK)
            {
StreamReader reader = File.OpenText(dlg.FileName);
string line;

GridViewURL.Rows.Clear();
while ((line = reader.ReadLine()) != null)
                {

string[] arr1 = line.Split(',');
if (arr1[0].ToString() != "URLNo")
GridViewURL.Rows.Add(arr1[0], arr1[1], arr1[2], arr1[3]);


                }
reader.Close();


            }
        }
```

// Load form event load the controls to its required positions

```
privatevoidMainfrm_Load(object sender, EventArgs e)
        {

dataGridView1.Columns[0].Width = 100;
dataGridView1.Columns[1].Width = 250;
dataGridView1.Columns[2].Width = 100;
dataGridView1.Columns[3].Width = 200;
dataGridView1.Columns[4].Width = 100;
dataGridView1.Columns[5].Width = 100;
dataGridView1.Columns[6].Width = 100;
dataGridView1.Columns[7].Visible = false;


btnTestTypes_Click(sender, e);

this.txtContent.Width = this.tabControl1.Width - this.treeView1.Width - 100;

this.panelURL.Height = this.Height- statusStrip1.Height;
this.panelURL.Left = this.panelMenu.Width;
this.panelURL.Top = 0;
this.panelURL.Width = this.Width - this.panelMenu.Width;

this.panelTestResult.Height = this.Height - statusStrip1.Height - 80;
this.panelTestResult.Left = this.panelMenu.Width;
this.panelTestResult.Top = 0;
this.panelTestResult.Width = this.Width - this.panelMenu.Width - 175;
            tabControl1.Width = this.panelTestResult.Width;
this.dataGridView1.Width = this.tabControl1.Width - 25;
this.dataGridView1.Height = this.tabControl1.Height - 25;

this.panelTestTypes.Height = this.Height- statusStrip1.Height-100;
this.panelTestTypes.Left = this.panelMenu.Width;
this.panelTestTypes.Top = 0;
this.panelTestTypes.Width = this.Width - this.panelMenu.Width;
```

```
       this.txtContent.Width = this.tabControl1.Width - this.treeView1.Width - 50;
this.txtHeader.Width = this.tabControl1.Width - this.treeView1.Width - 50;

this.PanelChart.Height = this.Height- statusStrip1.Height - 100;
this.PanelChart.Left = this.panelMenu.Width;
this.PanelChart.Top = 0;
this.PanelChart.Width = this.Width - this.panelMenu.Width;
this.chart1.Width = this.PanelChart.Width - 50;
this.chart1.Height = this.PanelChart.Height - 50;

        }
```

Execute Dynamic URL test

```
privatevoidbtn_Run_Dynamic_Click(object sender, EventArgs e)
        {
Testtype = TestTypes.DYNAMIC;

            toolStripProgressBar2.Visible = true;
            toolStripStatusLabel2.Visible = true;

// delete temporary files

if (!this.backgroundDeleteTmp.IsBusy)
            {
this.backgroundDeleteTmp.RunWorkerAsync();

            }

Thread.Sleep(10000);

numWorkItems = Int32.Parse(this.TextThreads.Text);

            toolStripProgressBar1.Visible = true;
            toolStripStatusLabel1.Visible = true;

Calcrequests = Int32.Parse(this.TextThreads.Text) * (GridViewURL.RowCount - 1);
this.labelCalclValueRequests.Text = Calcrequests.ToString();

dataGridView1.Rows.Clear();
            dataGridView1.RowCount = Calcrequests + 1;
ClearformControls();

btnTestResult_Click(sender, e);
timer.Start();
Application.DoEvents();
ButtonRun.Enabled = false;
ButtonStop.Enabled = true;
buttonReset.Enabled = true;
            _shouldStop = false;

treeView1.Nodes.Clear();
TreeNodetNode = newTreeNode("Virtual Users");
tNode.ImageIndex = 0;
treeView1.Nodes.Add(tNode);
treeView1.ExpandAll();

if (!this.backgroundRun.IsBusy)
            {
backgroundRun.RunWorkerAsync();
            }
```

```
        }

// execute URL click (With Single user credentials)

privatevoidButtonRun_Click(object sender, EventArgs e)
        {
Testtype = TestTypes.SINGLE;

            toolStripProgressBar2.Visible = true;
            toolStripStatusLabel2.Visible = true;

if (!this.backgroundDeleteTmp.IsBusy)
            {
this.backgroundDeleteTmp.RunWorkerAsync();

            }

Thread.Sleep(10000);

numWorkItems = Int32.Parse(this.TextThreads.Text);

            toolStripProgressBar1.Visible = true;
            toolStripStatusLabel1.Visible = true;

Calcrequests = Int32.Parse(this.TextThreads.Text) * (GridViewURL.RowCount - 1);
this.labelCalclValueRequests.Text = Calcrequests.ToString();

dataGridView1.Rows.Clear();
            dataGridView1.RowCount = Calcrequests + 1;
ClearformControls();

btnTestResult_Click(sender, e);
timer.Start();
Application.DoEvents();
ButtonRun.Enabled = false;
ButtonStop.Enabled = true;
buttonReset.Enabled = true;
            _shouldStop = false;

treeView1.Nodes.Clear();
TreeNodetNode = newTreeNode("Virtual Users");
tNode.ImageIndex = 0;
treeView1.Nodes.Add(tNode);


if (!this.backgroundRun.IsBusy)
            {
backgroundRun.RunWorkerAsync();
            }

        }

privatevoidbackgroundRun_DoWork(object sender, DoWorkEventArgs e)
        {


WebsettingsWSetting = newWebsettings();

if (SettingsFormInst.CbxUseHttp10.Checked)
                WSetting.VersionHttp10 = true;

WSetting.UserAgent = SettingsFormInst.TxtUserAgent.Text;
```

```
// request.AllowAutoRedirect = CbxAutoRedirect.Checked;
WSetting.UserAgent = SettingsFormInst.TxtUserAgent.Text;
// request.CookieContainer = CookieCont;

if (SettingsFormInst.TxtReferer.Text.Length> 0)
WSetting.TxtReferer = SettingsFormInst.TxtReferer.Text;

if (SettingsFormInst.TxtAccept.Text.Length> 0)
WSetting.TxtAccept = SettingsFormInst.TxtAccept.Text;

if ((SettingsFormInst.CbxRange.Checked) && (SettingsFormInst.TxtRangeFrom.Text.Length
+ SettingsFormInst.TxtRangeTo.Text.Length> 0))
            {
if (SettingsFormInst.TxtRangeFrom.Text.Length == 0)
WSetting.TxtRangeTo = (Int32.Parse(SettingsFormInst.TxtRangeTo.Text) * -1);
elseif (SettingsFormInst.TxtRangeTo.Text.Length == 0)
WSetting.TxtRangeFrom = (Int32.Parse(SettingsFormInst.TxtRangeFrom.Text));

            }

List<string>UrlListT = GetUrlList(this.GridViewURL);

txtClickDelayval = Int32.Parse(txtClickDelay.Text);
Application.DoEvents();

for (inti =1; i<= Int32.Parse(TextThreads.Text); i++)
            {
HttpWebUser user = newHttpWebUser();
user.Uname= "USER_" + i.ToString();
user.Usernumber = i;

if (Testtype == TestTypes.DYNAMIC)
user.Testtype = TestTypes.DYNAMIC;
else
user.Testtype = TestTypes.SINGLE;


Threadworker Thread = newThread(() =>VirtualUserHandler(user,UrlListT, WSetting));
workerThread.Start();


Application.DoEvents();
this.backgroundRun.ReportProgress(i);

if (backgroundRun.CancellationPending)
                {
e.Cancel = true;
backgroundRun.ReportProgress(0);
return;
                }
             }
        }

// execute URL click (With Multiple user credentials from CSV)

privatevoidbtnGOMultiple_Click(object sender, EventArgs e)
        {
Testtype = TestTypes.MULTIPLE;

            toolStripProgressBar2.Visible = true;
            toolStripStatusLabel2.Visible = true;
```

```
if (!this.backgroundDeleteTmp.IsBusy)
            {
this.backgroundDeleteTmp.RunWorkerAsync();

            }

Thread.Sleep(10000);

numWorkItems = Int32.Parse(this.TextThreads.Text);

            toolStripProgressBar1.Visible = true;
            toolStripStatusLabel1.Visible = true;

dataGridView1.Rows.Clear();

Calcrequests = Int32.Parse(this.TextThreads.Text) * (GridViewURL.RowCount - 1);
this.labelCalclValueRequests.Text = Calcrequests.ToString();

            dataGridView1.RowCount = Calcrequests + 1;


ClearformControls();

btnTestResult_Click(sender, e);
timer.Start();
Application.DoEvents();
ButtonRun.Enabled = false;
ButtonStop.Enabled = true;
buttonReset.Enabled = true;
            _shouldStop = false;

treeView1.Nodes.Clear();
TreeNodetNode = newTreeNode("Virtual Users");
tNode.ImageIndex = 0;
treeView1.Nodes.Add(tNode);


if (!this.backGRunMultiple.IsBusy)
            {
backGRunMultiple.RunWorkerAsync();
            }
        }
```

Virtual Web User Creation

```
privatevoid backgroundRun_DoWork(object sender, DoWorkEventArgs e)
        {

Websettings WSetting = newWebsettings();

List<string> UrlListT = GetUrlList(this.GridViewURL);

            txtClickDelayval = Int32.Parse(txtClickDelay.Text);
Application.DoEvents();

for (int i =1; i <= Int32.Parse(TextThreads.Text); i++)
            {
HttpWebUser user = newHttpWebUser();
                user.Uname= "USER_" + i.ToString();
                user.Usernumber = i;
```

```
if (Testtype == TestTypes.DYNAMIC)
                    user.Testtype = TestTypes.DYNAMIC;
else
                    user.Testtype = TestTypes.SINGLE;


Thread workerThread = newThread(() => VirtualUserHandler(user,UrlListT,
               WSetting));
                workerThread.Start();


Application.DoEvents();
this.backgroundRun.ReportProgress(i);

if (backgroundRun.CancellationPending)
                {
                    e.Cancel = true;
                    backgroundRun.ReportProgress(0);
return;
                }


            }
        }

privatevoid backgroundRun_ProgressChanged(object sender, ProgressChangedEventArgs
       e)
        {
int thread = int.Parse(this.TextThreads.Text);

int val = ((e.ProgressPercentage * 100) / thread);

            toolStripProgressBar1.Value = val;
            toolStripStatusLabel1.Text = val.ToString() + "%";
        }

privatevoid backgroundRun_RunWorkerCompleted(object sender,
       RunWorkerCompletedEventArgs e)
        {
            toolStripProgressBar1.Visible = false;
if (e.Cancelled)
            {

this.toolStripStatusLabel1.Text = "Processing cancelled";
            }
elseif (e.Error != null)
            {
this.toolStripStatusLabel1.Text = e.Error.ToString();
            }
else
            {

this.toolStripStatusLabel1.Text = "Completed";
            }
        }
```

Virtual User Handler

```
privatevoid VirtualUserHandler(HttpWebUser User, List<string> UrlList, Websettings
WSetting, string postDataStr=null)
        {
int Urlno = 1;
TreeNode tNode2;
foreach (string url in UrlList)
            {
string[] arr = url.Split(',');
Webresponse Wresponse = newWebresponse();


              Wresponse = User.ClickDynamicURL(arr[1], arr[2], WSetting, Urlno,
              arr[3],postDataStr);


Interlocked.Increment(ref requests);
Interlocked.Increment(ref requestsLastSec);


if (Wresponse.StatusCode == HttpStatusCode.OK) Interlocked.Increment(ref
                count200);
                elseif (Wresponse.StatusCode == HttpStatusCode.Unauthorized)
                Interlocked.Increment(ref count401);
elseif (Wresponse.StatusCode == HttpStatusCode.NotFound) Interlocked.Increment(ref
               count404);
elseif (Wresponse.StatusCode == HttpStatusCode.NotModified)
                Interlocked.Increment(ref count304);

if (treeView1.InvokeRequired)
                {
                    treeView1.Invoke(newAction(() =>
                    {

                        tNode2 = newTreeNode(Wresponse.ThreadName);

if (Wresponse.IsExOccured == false&& Wresponse.StatusCode == HttpStatusCode.OK)
                        {
                            tNode2.ImageIndex = 1;

                        }
else
                        {
                            tNode2.ImageIndex = 2;
Interlocked.Increment(ref countFailures);
                        }
                        treeView1.Nodes[0].Nodes.Add(tNode2);

                    }));
                }

if (dataGridView1.InvokeRequired)
                {
                    dataGridView1.Invoke(newAction(() =>
                    {

                        dataGridView1.Rows[RowCount].Cells[0].Value =
                        Wresponse.ThreadName;
```

```
                        dataGridView1.Rows[RowCount].Cells[1].Value =
                        Wresponse.ResponseUri.ToString();
                        dataGridView1.Rows[RowCount].Cells[2].Value =
                        ((int)Wresponse.StatusCode) + "";
                        dataGridView1.Rows[RowCount].Cells[3].Value =
                        Wresponse.Description.ToString();


                        dataGridView1.Rows[RowCount].Cells[4].Value =
                        Wresponse.contentLengthstr;
                        dataGridView1.Rows[RowCount].Cells[5].Value =
                        Wresponse.ResponseTime.TotalSeconds;

                        dataGridView1.Rows[RowCount].Cells[6].Value =
                        Wresponse.Average;

                        dataGridView1.Rows[RowCount].Cells[7].Value =
                        Wresponse.Headers.ToString();

Interlocked.Increment(ref RowCount);
Interlocked.Increment(ref Urlno);

                    }));
                }

            }

Interlocked.Decrement(ref numWorkItems);
        }
    }
```

Response Time Graph

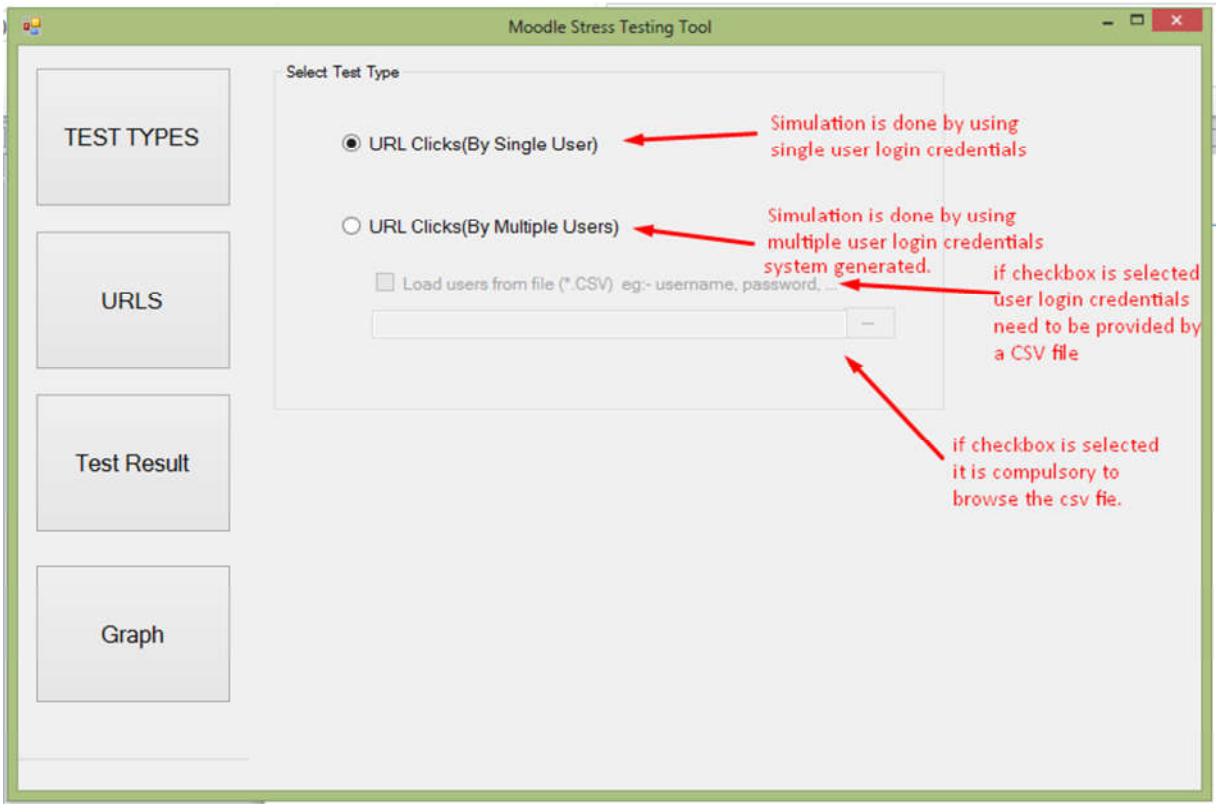
All the response data loaded in to the grid using VirtualUserHandler is used to Draw the graph. Below is the souce code of the response time graph.

```
private void Chart_Click(object sender, EventArgs e)
        {
            chart1.Series.Clear();
            this.PanelChart.Visible = true;
            this.panelTestResult.Visible = false;
            this.panelTestTypes.Visible = false;
            this.panelURL.Visible = false;
            int URLRowCount = this.GridViewURL.RowCount - 1;
            int arrySize = Int32.Parse(this.TextThreads.Text);
            double[,] z = new double[URLRowCount, arrySize];
            double[] y = new double[arrySize];
            string[] x = new string[arrySize];


            int RowCount = (Int32.Parse(this.TextThreads.Text) * URLRowCount);

            string user = "";
            for (int k=0; k<RowCount;k++)
            {
                user = dataGridView1.Rows[k].Cells[0].Value.ToString();
                string[] words = user.Split('-');
```

```
                int a = Int32.Parse(words[0].ToString().Substring(5));
                int b = Int32.Parse(words[1].ToString());
                string s = dataGridView1.Rows[k].Cells[5].Value.ToString();
                z[b-1,a-1] =
double.Parse(dataGridView1.Rows[k].Cells[5].Value.ToString());

            }
            for (int NoURL = 0; NoURL < URLRowCount; NoURL++)
            {
                for (int i = 0; i < Int32.Parse(this.TextThreads.Text); i++)
                {
                    x[i] = "user_" + i.ToString();
                    y[i] = z[NoURL, i];

                }


                chart1.Series.Add(new Series(NoURL.ToString()));
                chart1.Series[NoURL].IsValueShownAsLabel =false;
                chart1.Series[NoURL].BorderWidth = 2;
                chart1.Series[NoURL].ChartType = SeriesChartType.Line;
                chart1.Series[NoURL].Points.DataBindXY(x, y);
            }
```

APPENDIX-C: User Documentation

✓ Test Type Selection



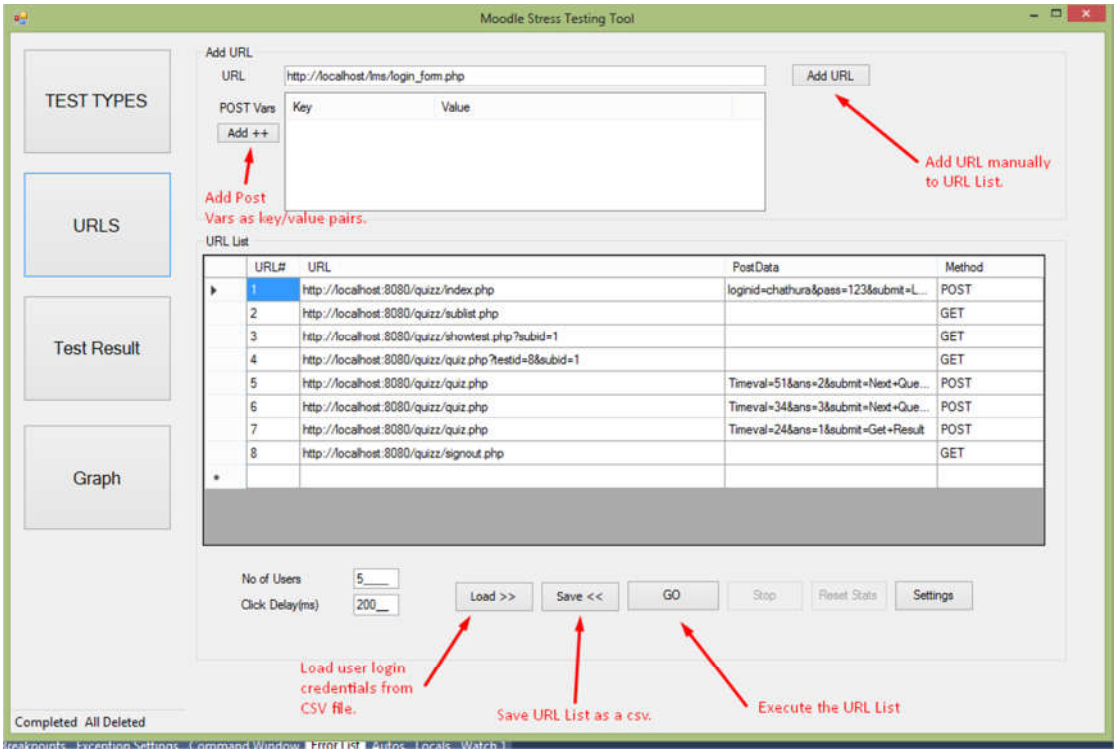
Option-1: URL Clicks (By Single User)

Here LMS is simulated by single user login credentials. By this method LMS can be simulated for simple task such as user login, assignment downloads etc. can be done. Cannot be used for online exam, quiz and assignment uploads etc. once the task is consumed by the user it is no longer available for the particular user.

Option-2: URL Clicks (By Multiple Users)

Here LMS is simulated by multiple user login credentials. By this method LMS can be simulated for online exam, quiz and assignment uploads etc. This can be done in two ways (1). By system generated user credentials (2). By user login credentials provided by using a CSV file.

✓ URLS Interface

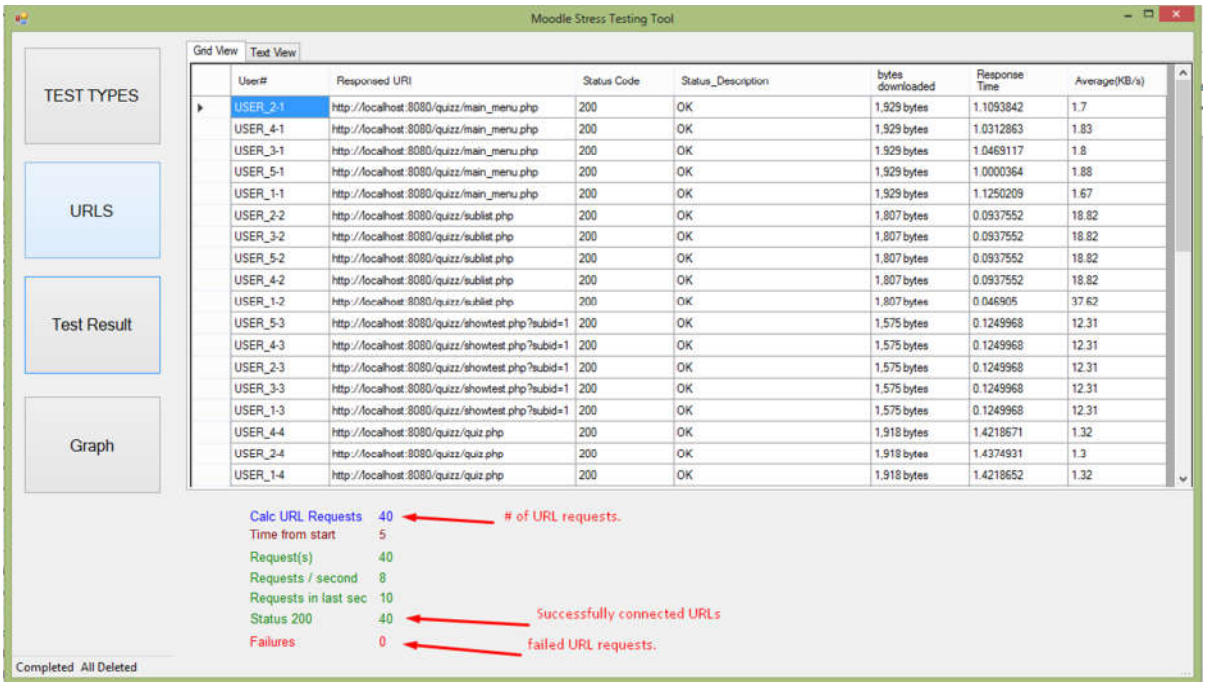


URLs can be added either manually or using CSV file. When POST URLs added manually post variables need to be sent with POST request. Add++ button helps to add PostData as Key/Value Pairs with the URL.

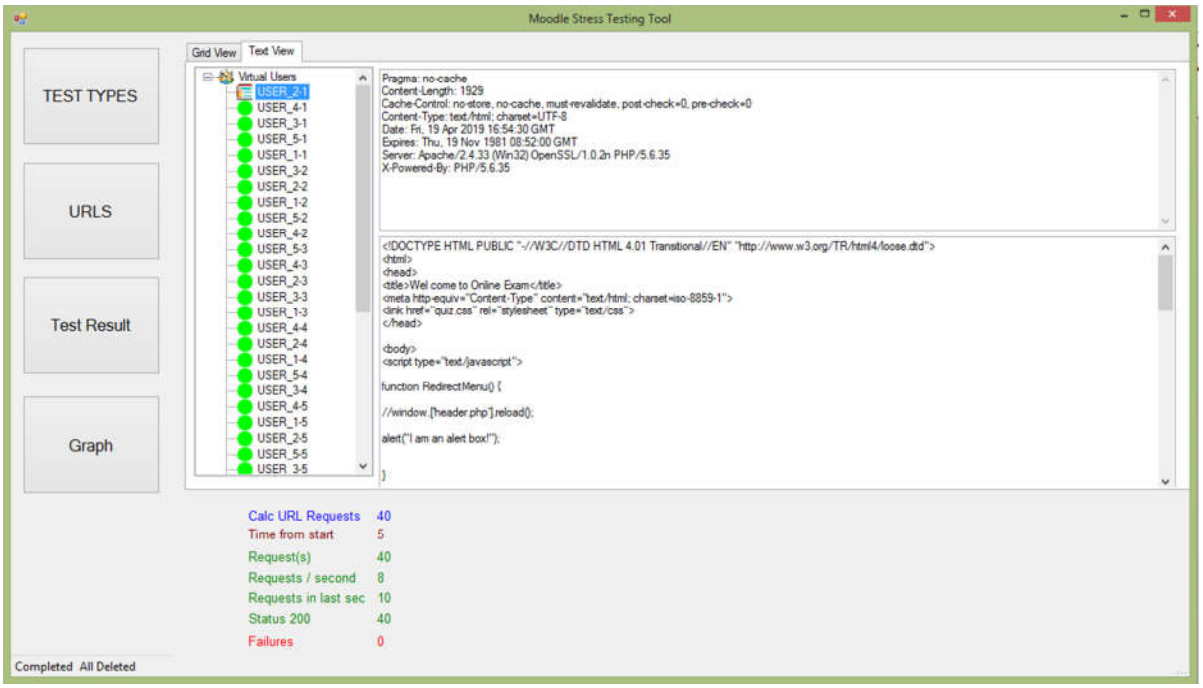
After Loading URLs to URL List No of Users need to be specified otherwise it take Number of users as 5. Click delay is the no milliseconds clasps between each URL click.

✓ Test Results Interface

Test Results interface display status of the Load Test. It provides details of the each user thread running. Test results can be viewed in a data grid view or a tree view. Following figure shows the test results as data grid view.

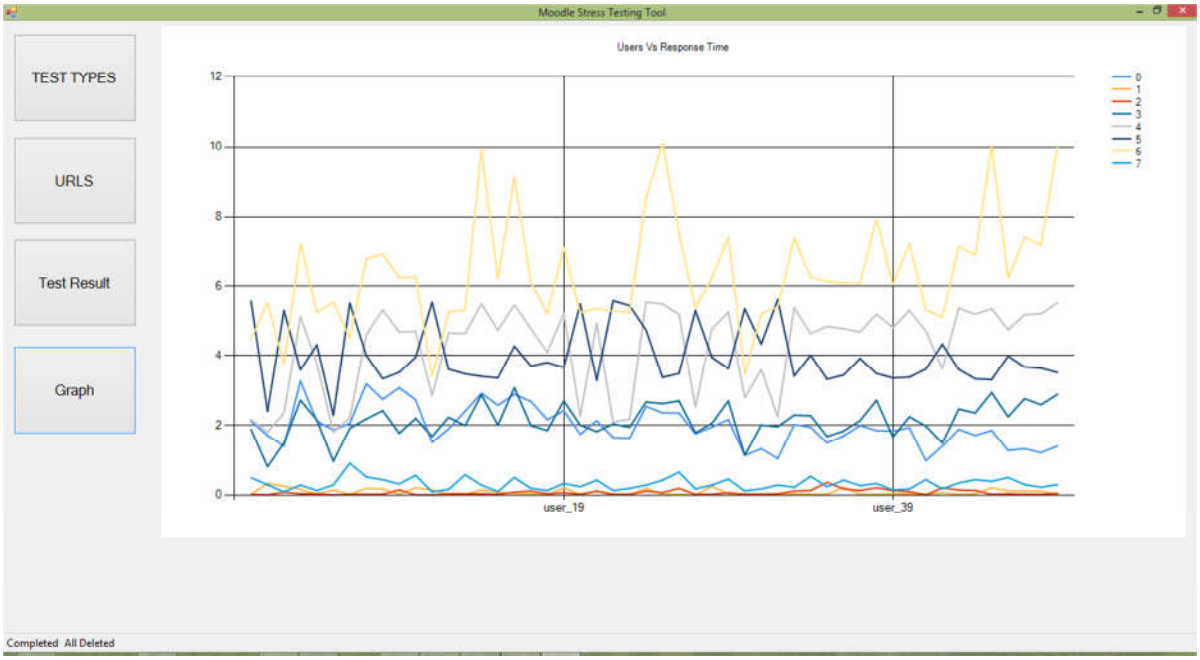


Following figure shows the test results as tree view



✓ Graph Interface

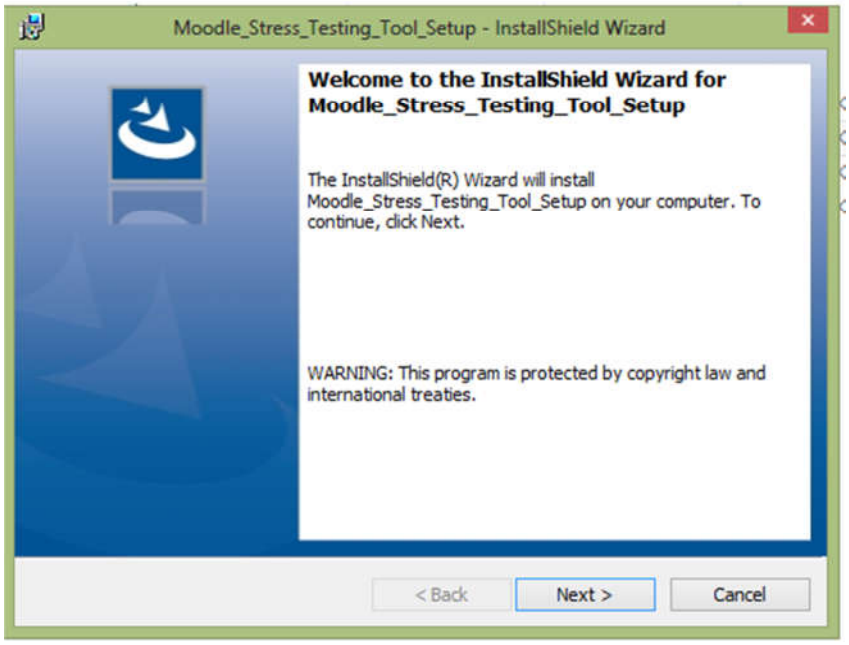
Graph interface display the displays Users Vs Response Time graph. Each line displays the response time of the each URLs.



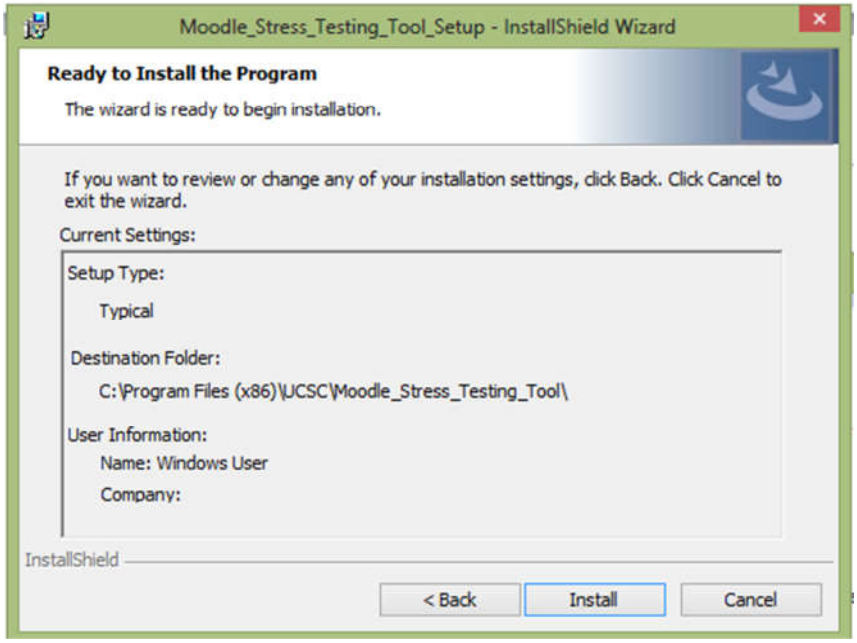
APPENDIX-D: System Documentation

✓ Installation of Moodle Stress Testing Tool

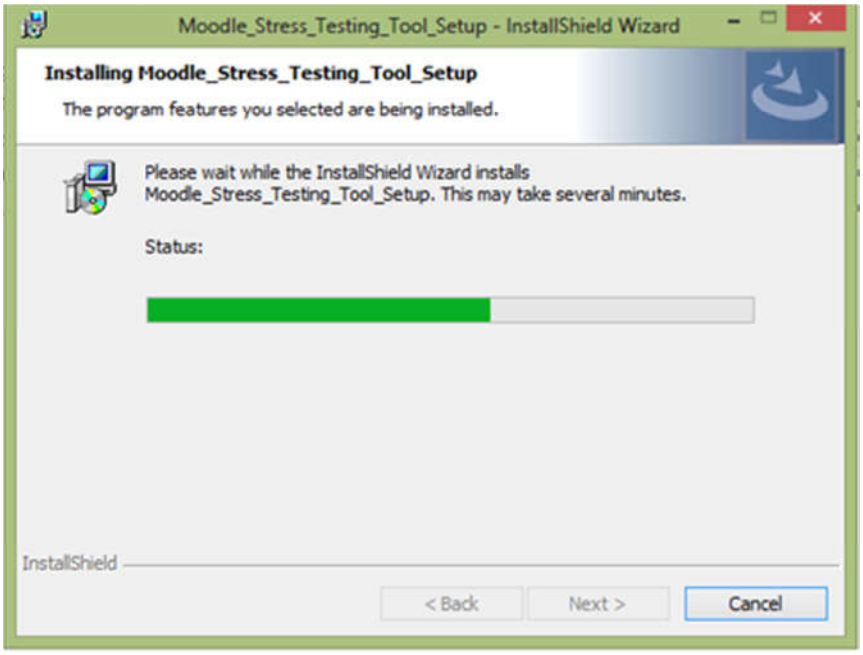
1) Double click on Moodle stress Testing Setup and go next.



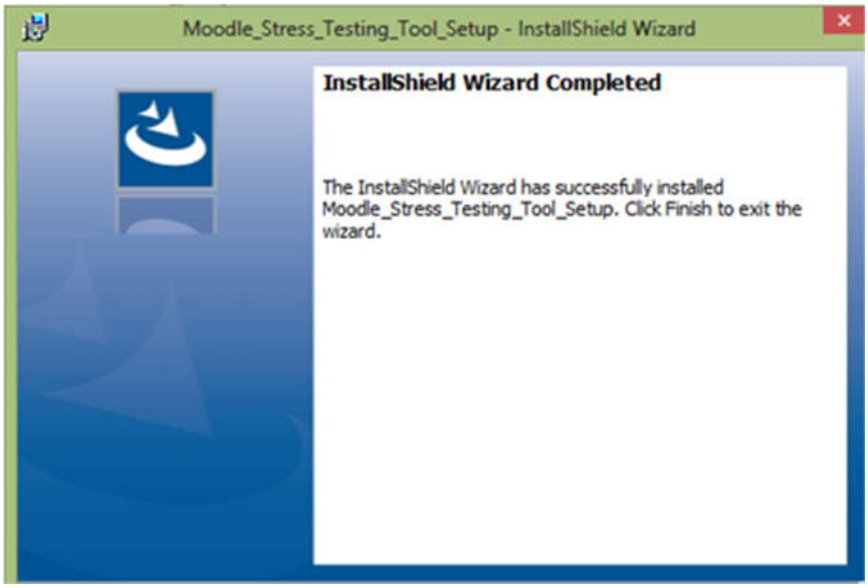
2) Click on Install.



3) Installation process will get stated.



4) Installation is completed successfully and now Moodle Stress Testing Tool can be used.



✓ Installation of NetData

Please refer below link
https://docs.netdata.cloud/packaging/installer/