EAST WEST UNIVERSITY

Department of Computer Science and Engineering

CSE430 - Software Testing and Quality Assurance

Lab Topic: Load Testing with Apache JMeter

1. Objective

In this lab, students will learn how to use Apache JMeter for performance testing, focusing on load testing of web applications. Students will create test plans, simulate multiple virtual users, analyze results, and gain insights into web server performance under varying load conditions.

2. Learning Outcomes

1. After completing this lab, students will be able to:
2. Install and configure Apache JMeter for load testing.
3. Create Test Plans simulating real-world traffic using multiple virtual users.
4. Perform response validation using assertions.
5. Analyze performance results using JMeter's listeners and metrics.
6. Execute advanced configurations such as timers, CSV data, and post requests.

3. System Requirements and Setup

3.1 Prerequisites

- **Java Development Kit (JDK)** 8 or above.
- **Operating System:** Windows, macOS, or Linux.
- **Internet Connection** (for downloading dependencies).

3.2 Installing Java

1. Download JDK from Oracle Java SE Downloads.
2. Install and set environment variables:
    - **Windows:** Add JAVA_HOME to Environment Variables.
    - **macOS/Linux:** Set JAVA_HOME in .bashrc or .zshrc.
3. Verify installation:
   Run the command java -version in your terminal/command prompt.

3.3 Installing JMeter

1. Download JMeter from Apache JMeter Downloads.
2. Extract the archive to a directory.
3. Navigate to the **bin** folder.
4. Launch JMeter:
    - **Windows:** Double-click jmeter.bat.
    - **Linux/macOS:** Run ./jmeter from the terminal.

4. Creating Your First Test Plan

4.1 Step 1: Create a Thread Group

A **Thread Group** simulates concurrent users:

1. Right-click on **Test Plan → Add → Threads (Users) → Thread Group**.
2. Set parameters:

    - **Number of Threads:** 10 (10 virtual users).
    - **Ramp-Up Period:** 10 (1 user per second).
    - **Loop Count:** 5 (each user runs 5 times).

4.2 Step 2: Add HTTP Request Defaults

1. Right-click on **Thread Group → Add → Config Element → HTTP Request Defaults**.
2. Set:
    - **Server Name:** httpbin.org.
    - **Protocol:** https.
    - **Port:**
        - 80 for HTTP.
        - 443 for HTTPS.

4.3 Step 3: Add an HTTP Request Sampler

1. Right-click on **Thread Group → Add → Sampler → HTTP Request**.
2. Configure:
    - **Method:** GET.
    - **Path:** /.

4.4 Step 4: Add Response Assertion

To validate the response:

1. Right-click on the **HTTP Request → Add → Assertions → Response Assertion**.

2. Set:
    - **Field to Test:** Text Response.
    - **Pattern to Test:** Example Domain.

4.5 Step 5: Add Listeners

1. Right-click on **Thread Group → Add → Listener**.
2. Suggested listeners:
    - **View Results Tree**
    - **Summary Report**
    - **Aggregate Report**
    - **Graph Results**

5. Advanced Configurations

5.1 Timers

To simulate **user think time**:

1. Add → **Timer → Constant Timer**.
2. Set delay (e.g., 2000 ms).

5.2 CSV Data Set Config

For **parameterization**:

1. Add → **Config Element → CSV Data Set Config**.

2. Set:
    - **Filename:** users.csv.
    - **Variable Names:** username,password.

5.3 Using Post Requests

1. Change **Method** to POST.
2. Add parameters in **HTTP Request**:
    - **Name:** username, **Value:** ${username}.
    - **Name:** password, **Value:** ${password}.

6. Executing the Test Plan

1. Save your Test Plan as .jmx.
2. Click **Start** (green play button).
3. Monitor progress via Listeners (e.g., **Summary Report**, **View Results Tree**).

7. Analyzing Results

7.1 Summary Report

- **Average:** Mean response time.
- **Min/Max:** Shortest/longest request time.
- **Error %:** Failures.

- **Throughput:** Requests per second.

7.2 Example Interpretation

If 10 users send 5 requests each:

- **Total requests:** 50.
- **Ideal throughput:** Close to 5 requests/sec if the server is stable.
- **Errors:** Should be 0% under normal conditions.

8. Best Practices

- Use **Assertions** to validate correctness.
- Keep separate environments for **testing** and **production**.
- Use tools like **Grafana** or **New Relic** to monitor server health.
- Run tests multiple times for **consistency**.

9. Practice Problems

9.1 Problem 1: Load Test a Public Website

- Test the homepage of a public website like https://httpbin.org or https://example.com.
- Use 25 virtual users, ramp-up time of 15 seconds, and loop count of 3.
- **Goal:** Measure response time and server stability under moderate load.

9.2 Problem 2: Simulate Login using Fake Store API

- Test the login functionality on https://fakestoreapi.com.
- Use 5 virtual users with a loop count of 2.
- **Goal:** Validate login success and token response.

9.3 Problem 3: Search Feature Parameterized Test

- Use **GitHub API** to test parameterized search queries.
- Use CSV Data Set Config with keywords like jmeter, java, testing.
- **Goal:** Practice dynamic input data and validate search results.

9.4 Problem 4: Stress Test with High Load

- Stress test a static website like https://example.com with 100 virtual users.
- Ramp-up: 30 seconds, duration: 2 minutes (using scheduler).
- **Goal:** Observe system behavior and throughput under stress.

9.5 Problem 5: Analyze Test Results

- Analyze error rate, average response time, and throughput for any previous test scenario.
- **Goal:** Identify potential bottlenecks and write a report summarizing findings.

10. Conclusion

This lab introduced students to **Apache JMeter**, a critical tool in performance testing. By following the guide, students gain practical experience in simulating real-world traffic, validating responses, and deriving actionable insights into the performance of web servers and APIs.