Lab: Web Application & Performance

In this lab, we are to examine modern Web application development framework in Java. You will be asked to write a very simple Web application.

We will then use a benchmark tool to stress test the Web application you have developed to test the maximal load your application is capable of handling in face of real life Internet traffic.

A modern Java Web framework

The framework we have selected for this lab is **Java Spark**. Here are a few facts you need to know:

- Java Spark website is http://sparkjava.com/. You can find documentation, tutorial and sample code on the Web site.
- Java Spark and related library files (.jar) are given as part of this lab, so you don't need to go through the more involved Maven project setup as described on Java Spark website.
- Java Spark uses Java 8 syntax, so to compile Spark projects, you **must** use JDK 1.8, which is available at this link: http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

PART I: Setup for development

1. Read the Get Started section of Spark to familiarize yourself with the Spark API: http://sparkjava.com/documentation.html#getting-started
   Note: Spark uses some Java language features only available in Java 8. This is why you need to use JDK 1.8. In particular, Spark uses lambda expressions. Details can be find at https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html

   You don't need to understand the semantics behind lambda expressions for this lab.
2. Download the **lab-web.zip** file.
3. Uncompress the lab-web.zip file, and you will find a directory **lab-web** containing a number of Java library files (.jar), a Windows executable file (ab.exe) and a **sample** directory.
4. In the directory, you will find a sample application under **lab-web/sample**. Within the sample application directory, you will find two Windows command files (**compile.cmd** and **run.cmd**) for compiling and running the Web app using the Spark library. If you are on Linux, you can easily translate the **.cmd** files to the equivalent Linux version. Study the **cmd** files and **HelloWorld.java** to understand what it's doing.
5. While inside the **lab-web/sample** directory, run **compile.cmd** and **run.cmd**. You should see that the Web application is running and waiting for incoming traffic at **http://localhost:4567**
6. Use any browser, visit the following URLS:
   a. http://localhost:4567/hello
   b. http://localhost:4567/add?a=3.1415&b=10.0

c. http://localhost:4567

7. Can you explain the observations Step 6? Use the source code **lab-web/sample/HelloWorld.java** in your explanation.
8. Based on **HelloWorld**, write your own Web application, call it **LabWeb**, to support a few more URLs. You are free to design URL path name and their functionalities. Make sure at least one reads from the file system.

PART II: Benchmarking using ab.exe

You are given **ab.exe**, a program from Apache utility package. **Ab.exe** (stands for Apache Benchmark) performs exhaustive stress testing on a target Web site. It is available on Ubuntu under the **apache2-utils** package, and can be installed using *sudo apt-get install apache2-utils*.

Details on how to use **ab.exe** can be found here: http://en.wikipedia.org/wiki/ApacheBench

1. Use **ab** to test against every URL paths of your own Web application. Make sure you issue at least 500 requests (ab –n 500 "http://localhost:4567/...").
2. Rank the response times of each of your URL paths of your Web application from the slowest to the fastest.
3. Explain the performance differences you see.
4. Experiment with concurrent stress testing. Say, you issue 10 concurrent request sessions: ab –n 100 –c 10 "http://localhost:4567/..."
5. What does concurrent sessions do to the response time?
6. By increasing the concurrency 1, 2, 3, 4, 5, 6, …, if we want to achieve a response time of 5 sec or less, what is the **maximum** number of concurrent sessions your Web application can support?
7. Using ideas of distributed systems, offer solutions to scale up your Web applications to support increasing traffic demand while still maintaining responsiveness of key URL paths of the application.