**Introduction:**

**Goals:**

The goal of the system is to develop two clients and servers using Node.js and Express framework.

Server1:

To develop a “Calculator” to demonstrate stateless web services. This server should perform the basic calculator functions like addition, subtraction, multiplication and division of two numbers. It also should take care of exceptions.

Client1:

The client should provide the facility to test all the functions of calculator service.

Server2:

To develop a prototype of “LinkedIn Application” to demonstrate Restful web services.

The basic functionality of the server should include Sign up, Sign in and Sign out options.

Users account should provide at least user details, education, experience, skills and connections.

Should maintain time last logged in and should be returned back when user logs in.

To have own connection pooling.

Client2:

The client should provide the facility to test all the functions of “LinkedIn application” service up to 500 concurrent users.

**Purpose of the System:**

The purpose of the system is to test the developed application for a different number of concurrent of users and analyze the performance with the use of Node.js and Stateless web services.

**System Design:**

The chosen system design is the use of Node.js and Stateless web services. The architecture of the Node.js is the event driven, single threaded and asynchronous in nature. This architecture makes the application more scalable.

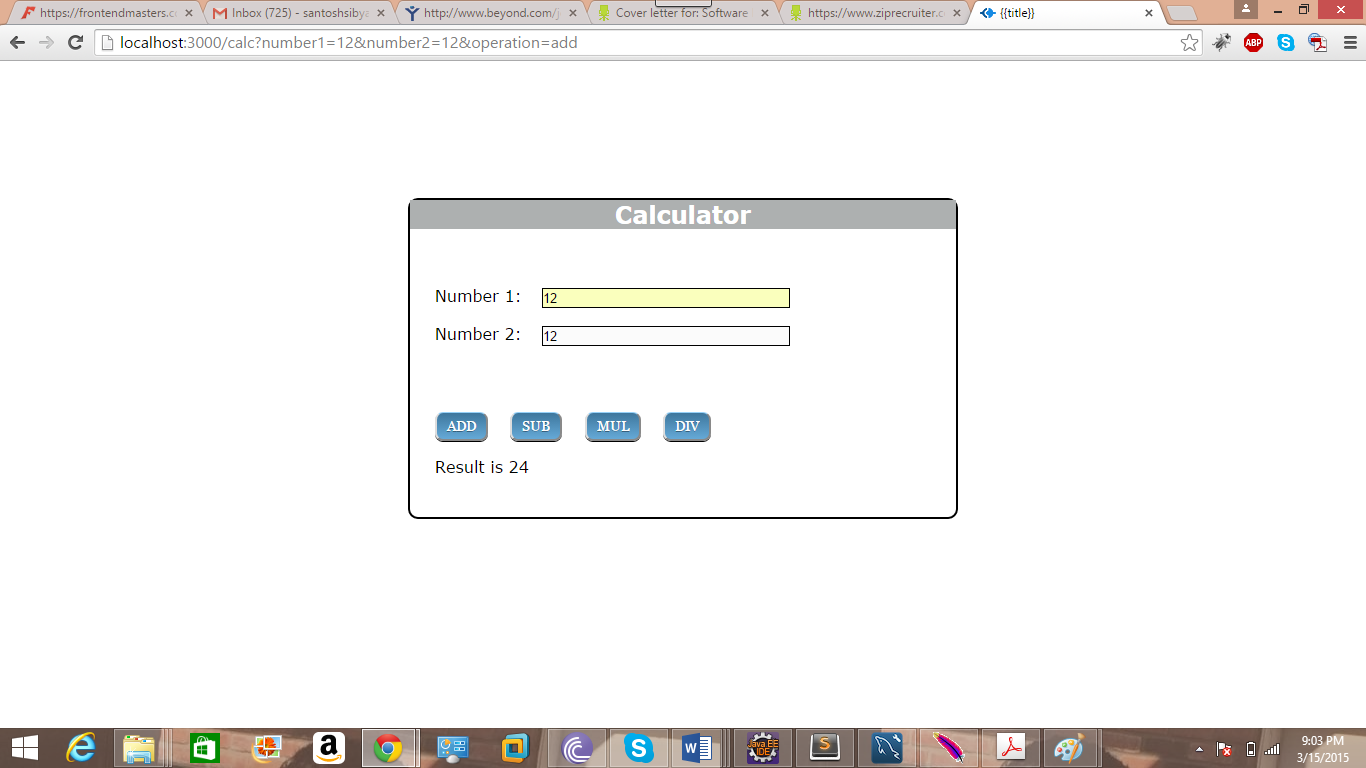
Results:

**Calculator Application:**

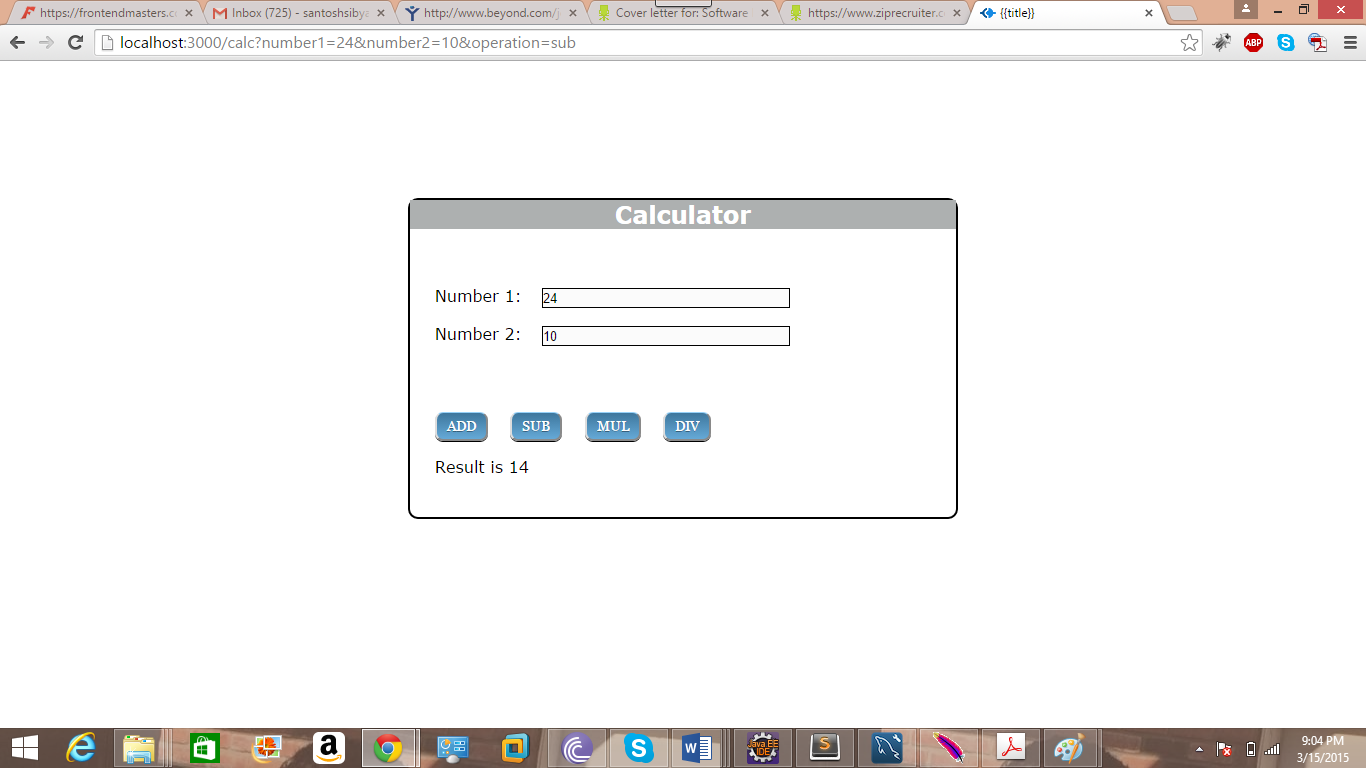
Home page:



Add operation:



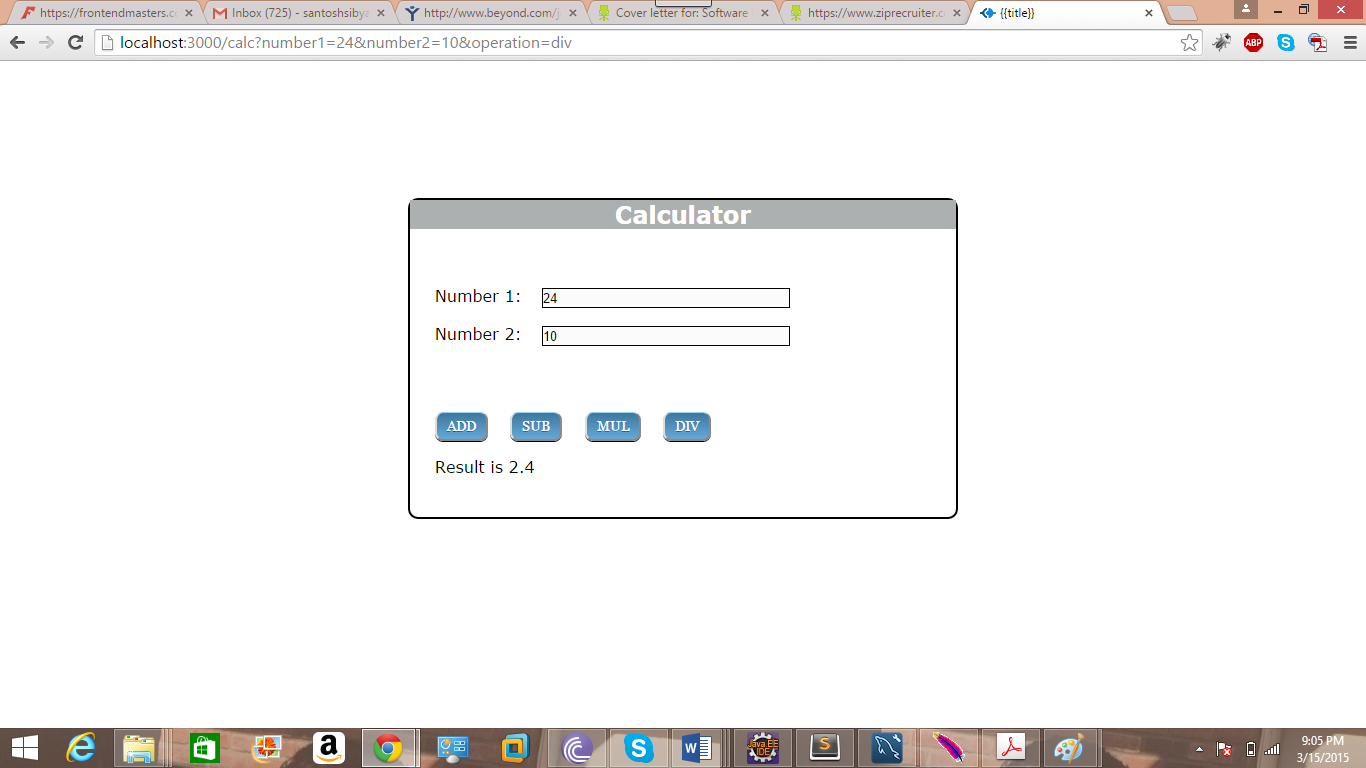
Subtraction:



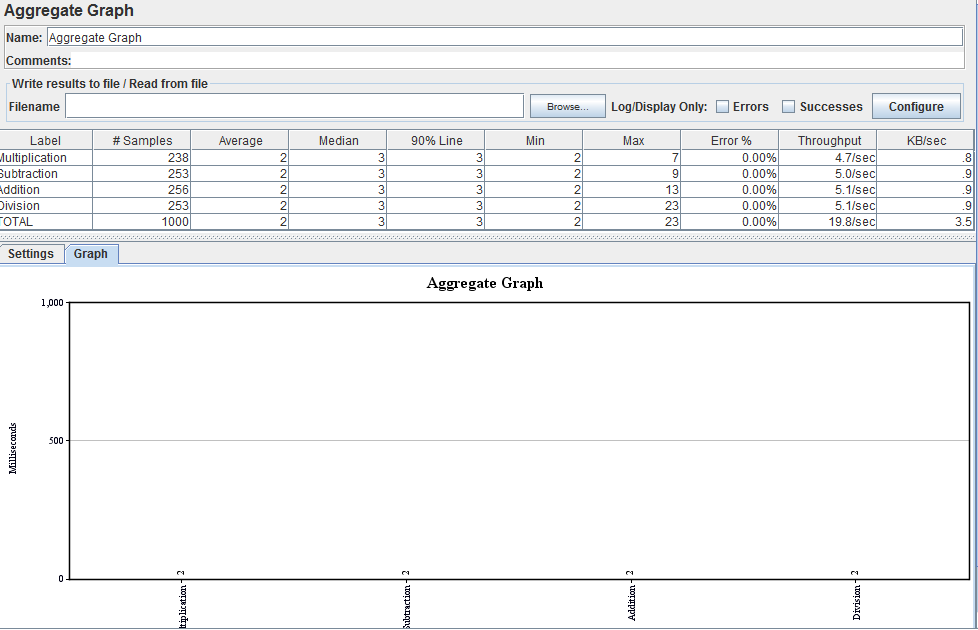
Multiplication:



Division:



**Calculator Server for 1000 concurrent users:**

****

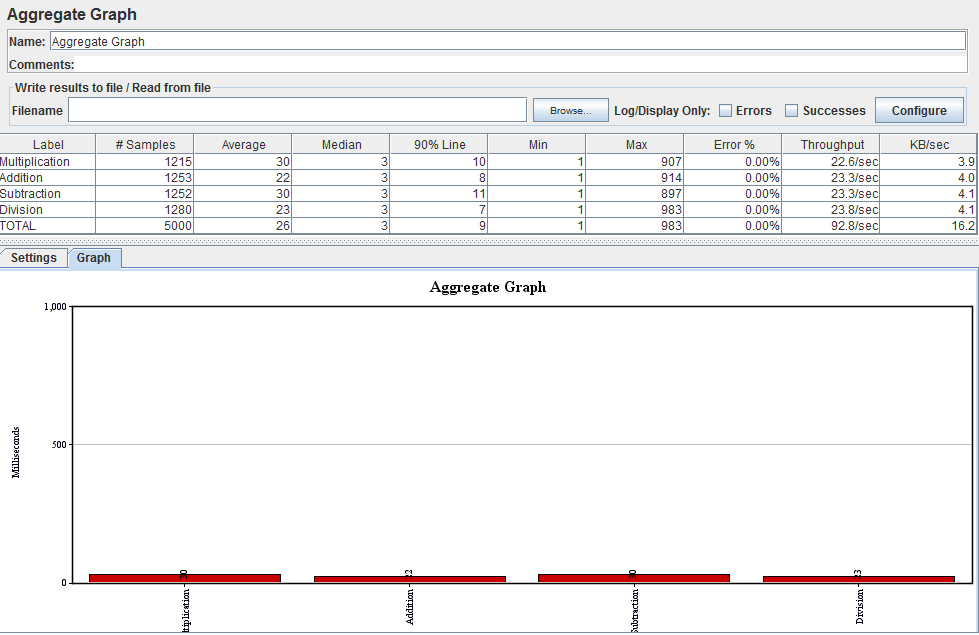
The average time taken for addition is 2 milliseconds.

The average time taken for subtraction is 2 milliseconds.

The average time taken for multiplication is 2 milliseconds.

The average time taken for division is 2 milliseconds.

**Calculator Server for 5000 concurrent users:**

****

The average time taken for Addition: 22 milliseconds

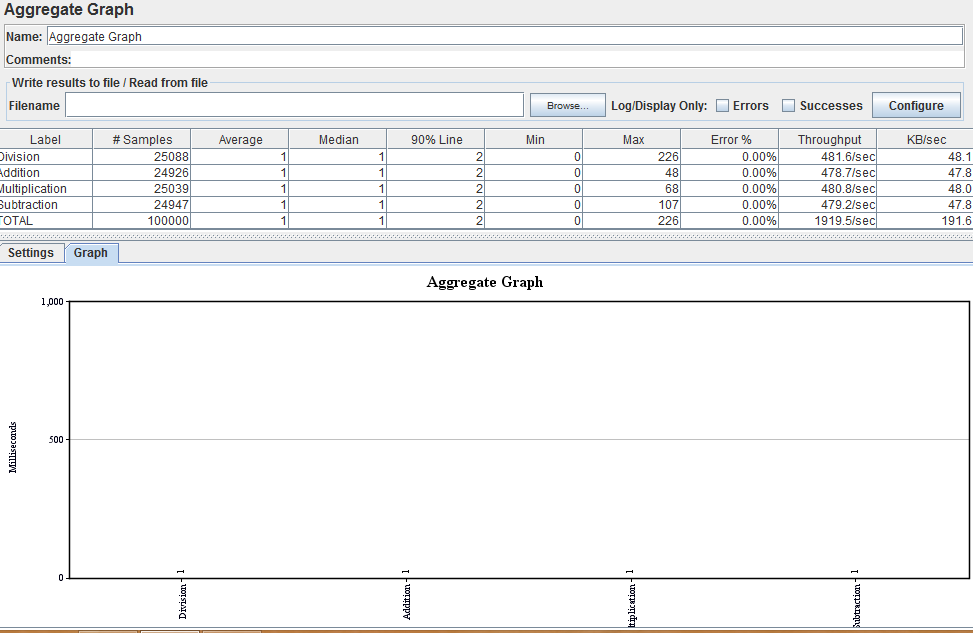
The average time taken for Subtraction: 30 milliseconds

The average time taken for Multiplication: 30 milliseconds

The average time taken for Division: 23 milliseconds.

Total average time taken for all operations: 26 milliseconds.

**Calculator Server for 100 concurrent users when each user makes 1000 calls to calculator:**

****

The average time taken for Addition: 1 millisecond.

The average time taken for Subtraction: 1 millisecond.

The average time taken for Multiplication: 1 millisecond.

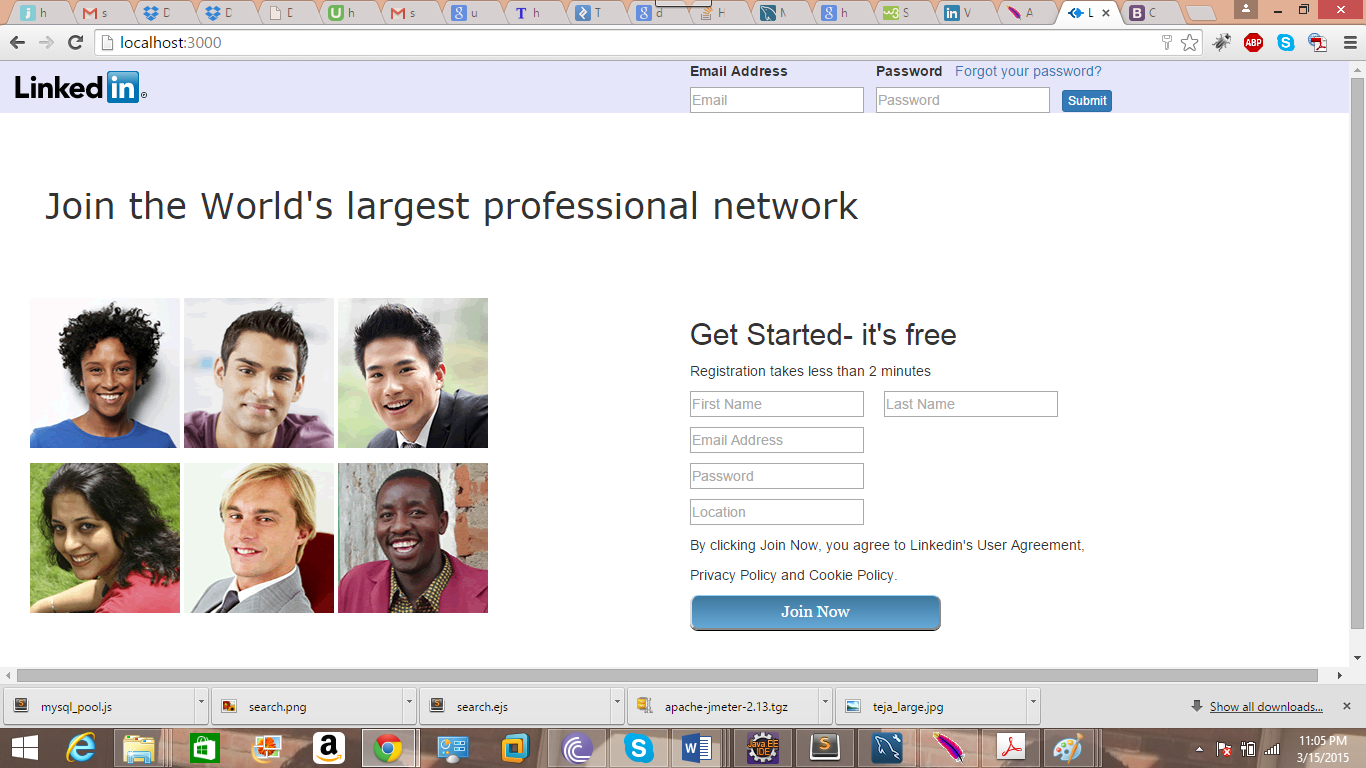
The average time taken for Division: 1 millisecond.

Total average time taken for all operations: 1millisecond.

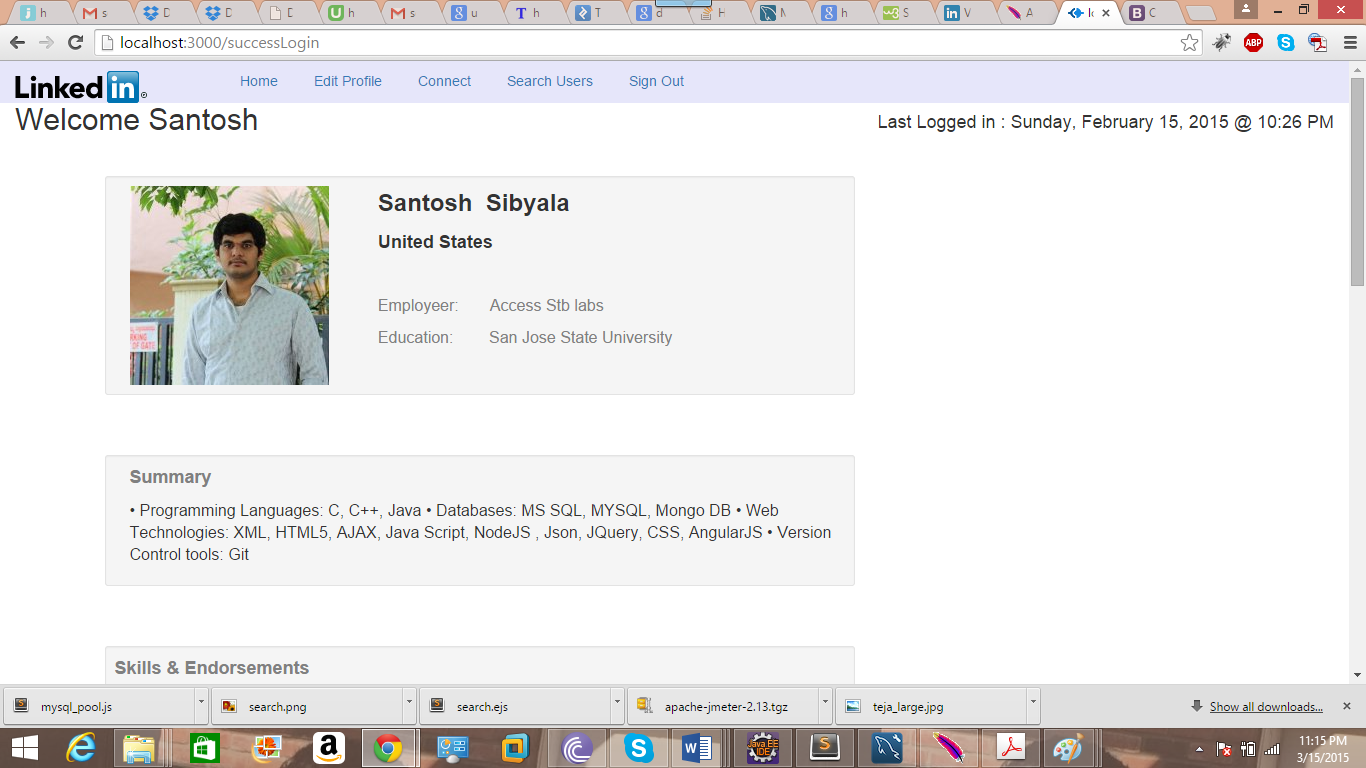
Average time taken is 1 millisecond for each operation. The average time taken for each operation in the third case is less than that of first and second cases because the browser might have stored the details of the operations using cookies in first and second cases and might have used those results.

LinkedIn Application:

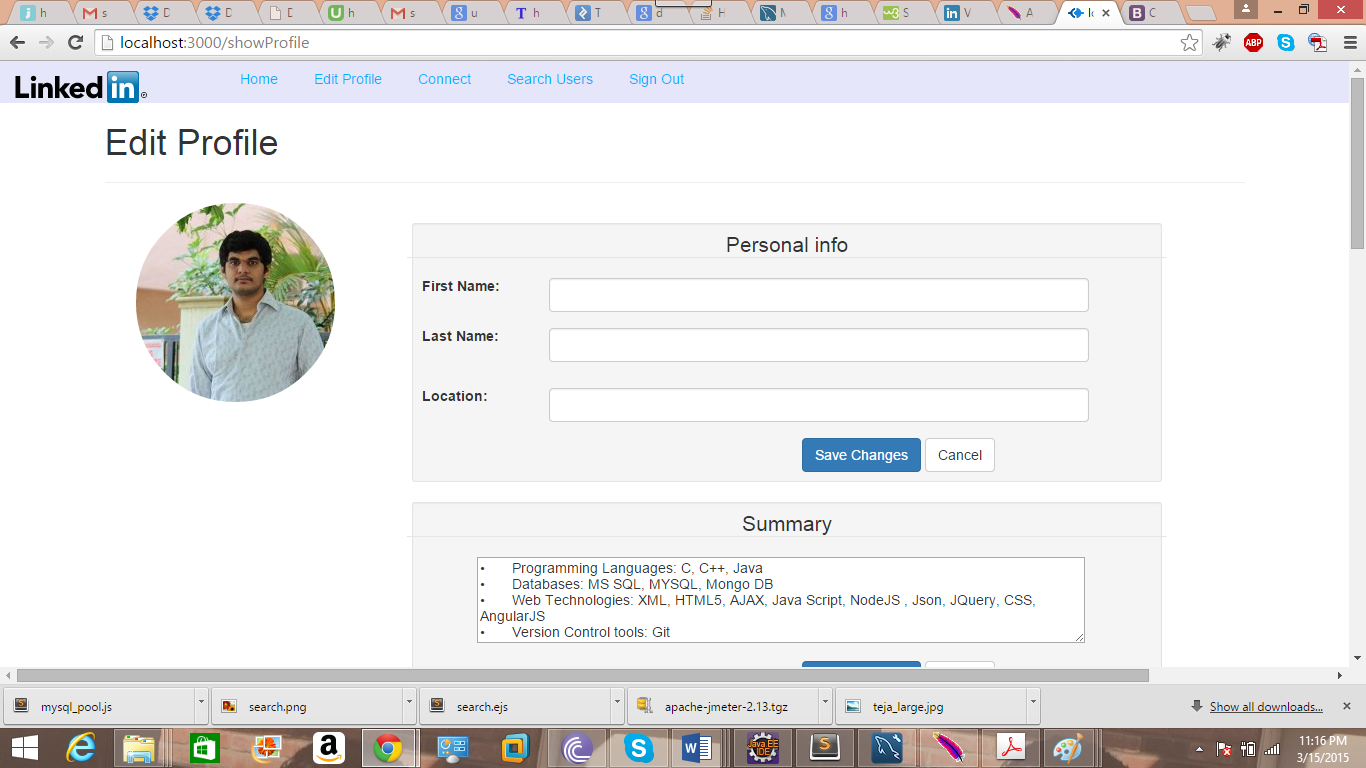
Signin and Signup page:



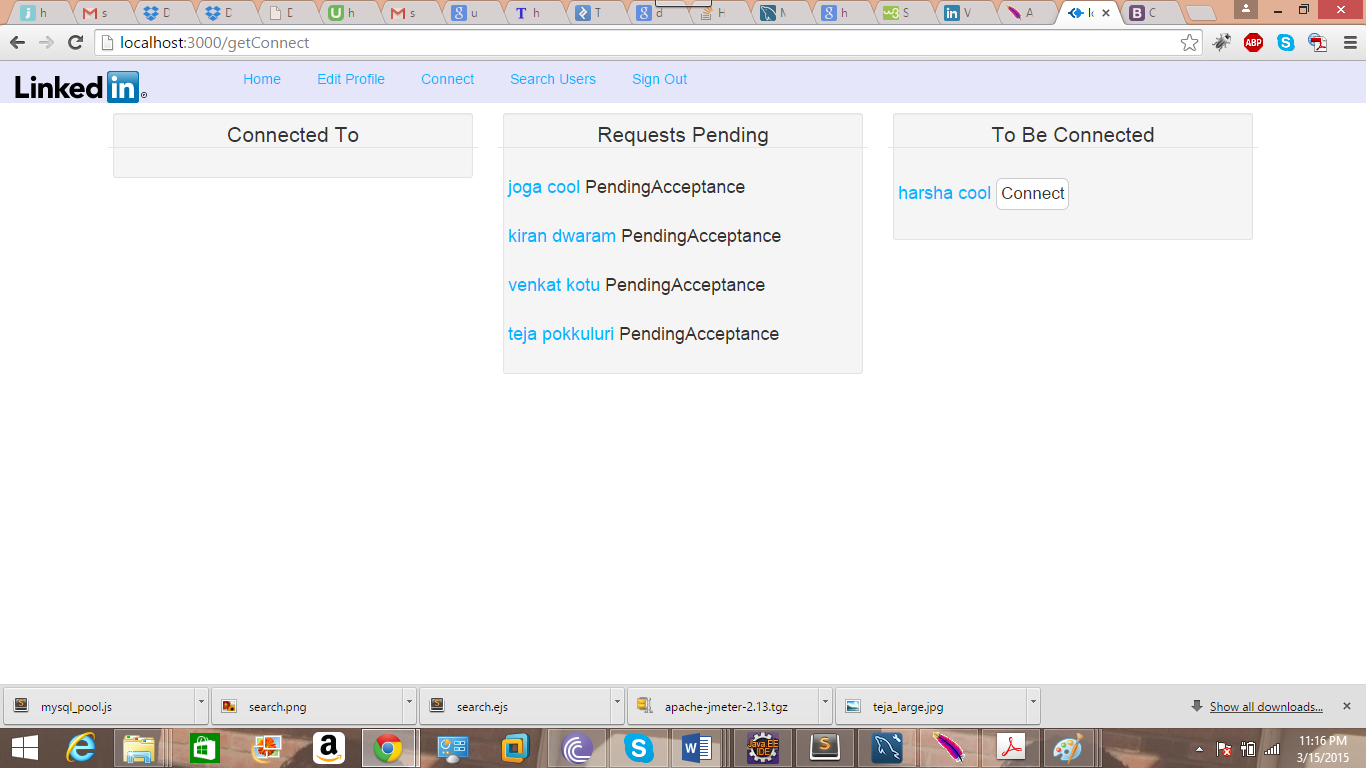
Home Page:



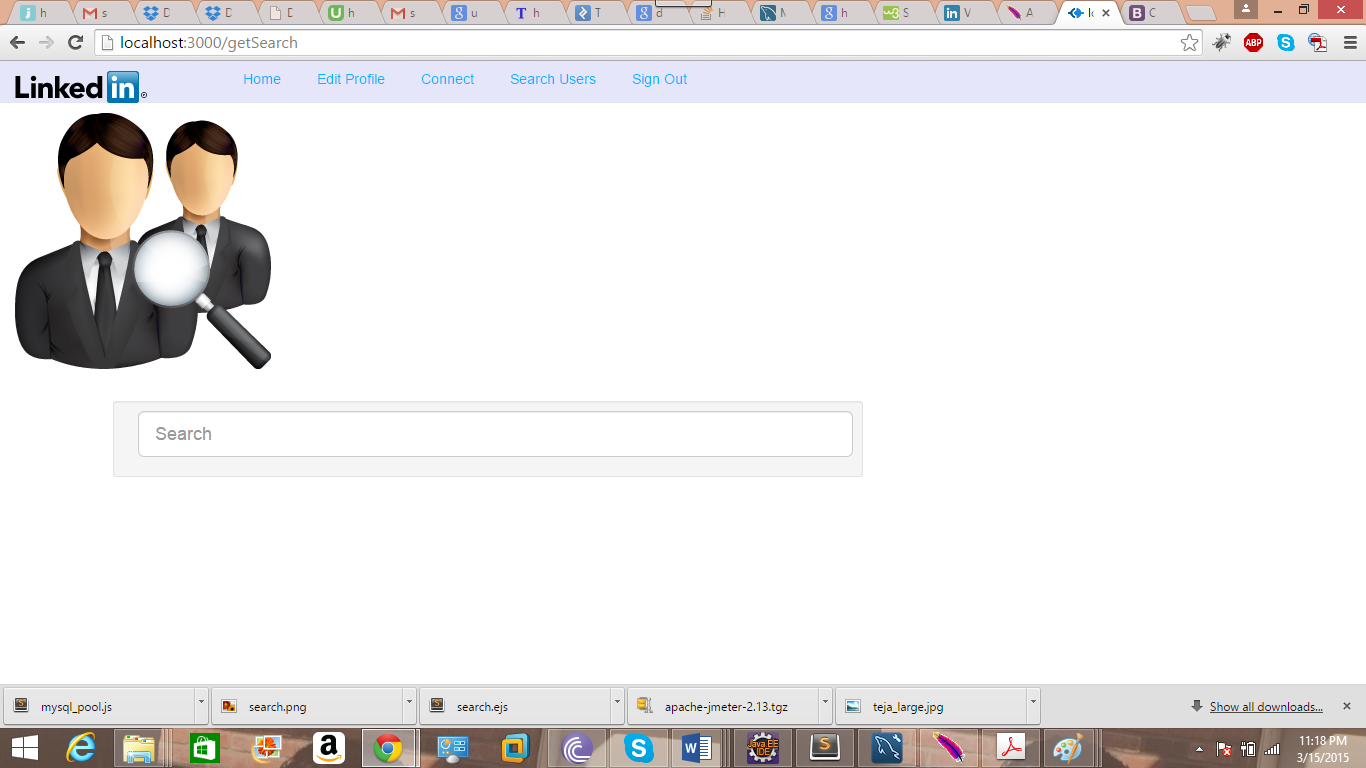
Edit Profile Page:



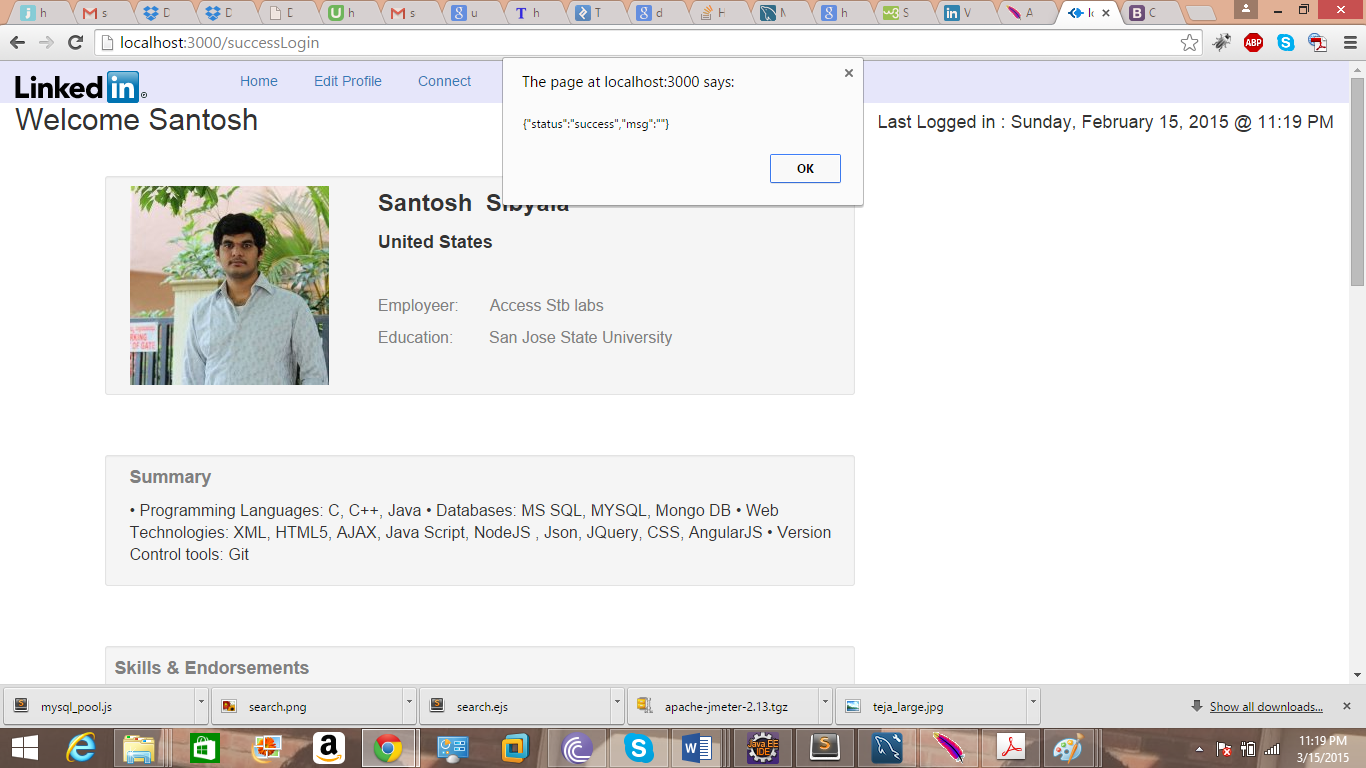
Connections Page:



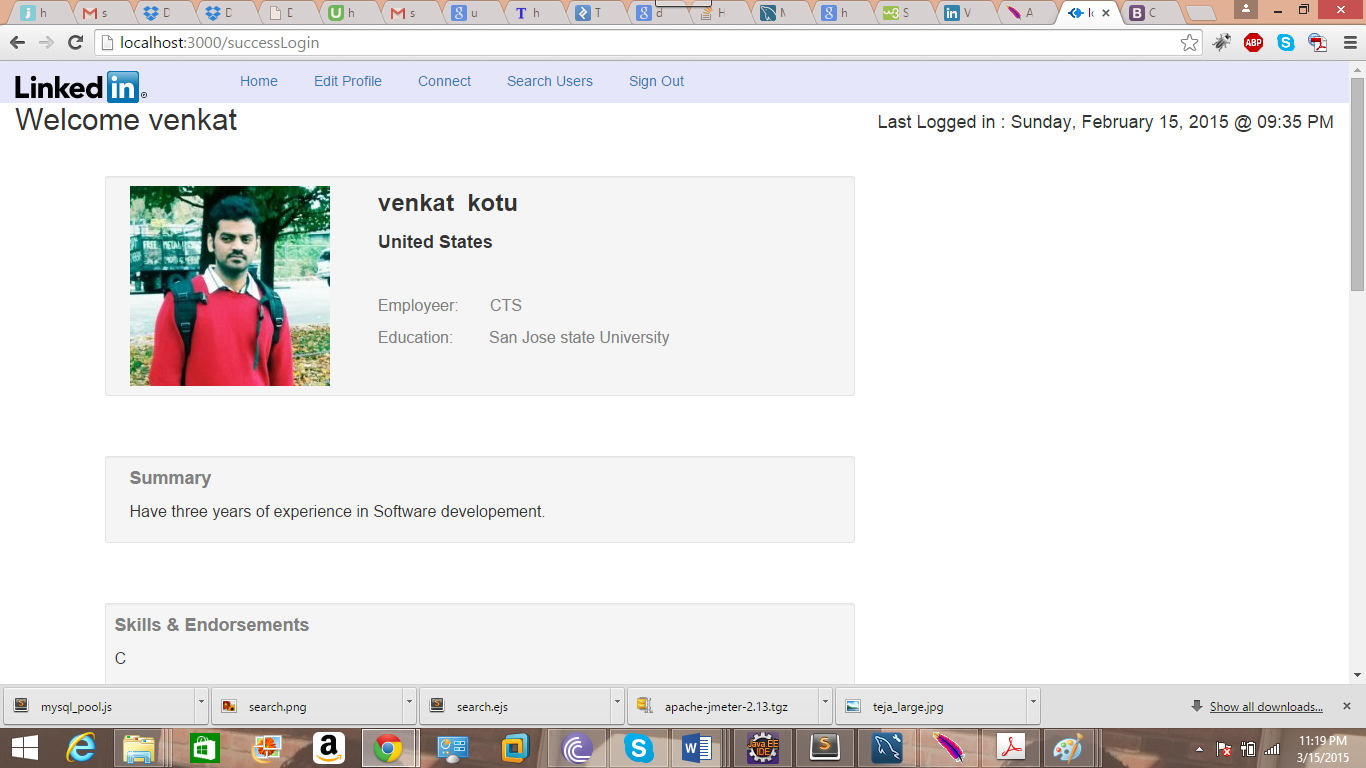
Search Page:



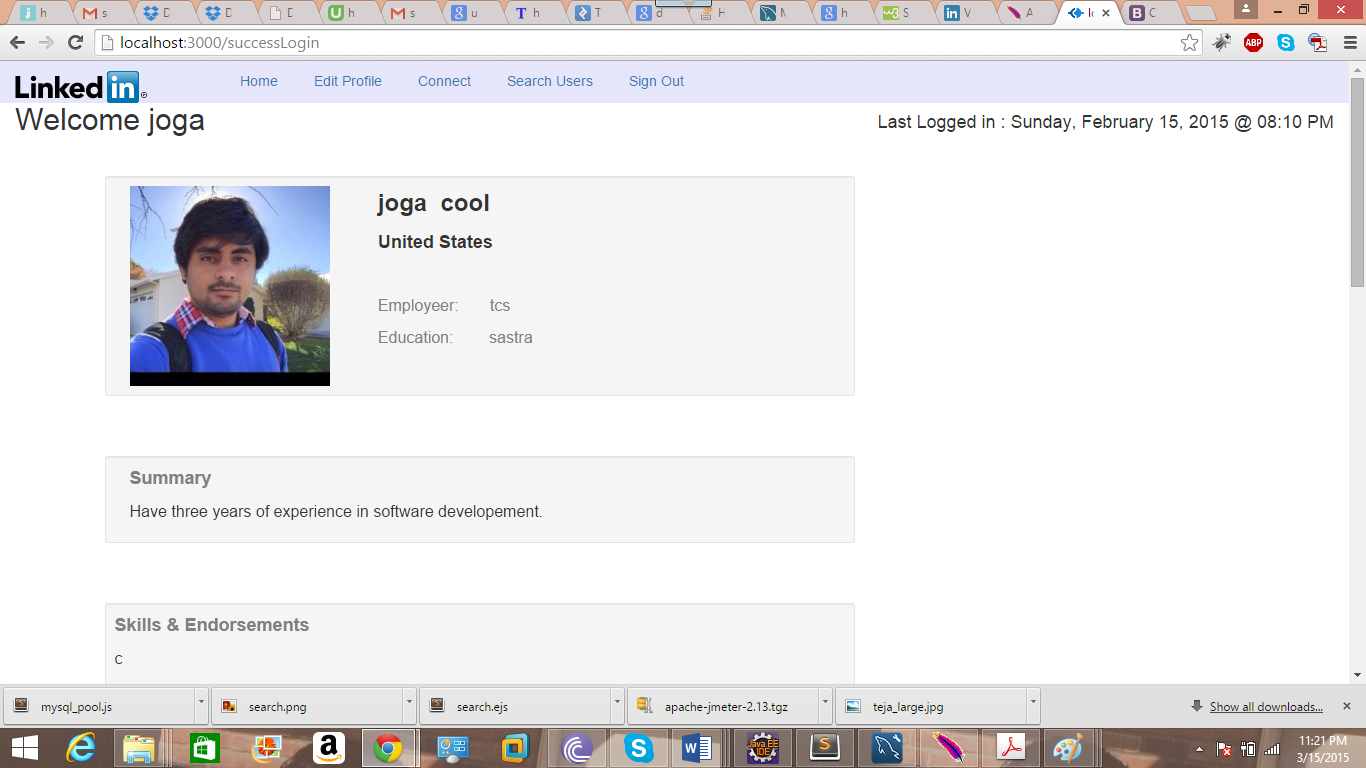
Signout Page:



User 2 home page:



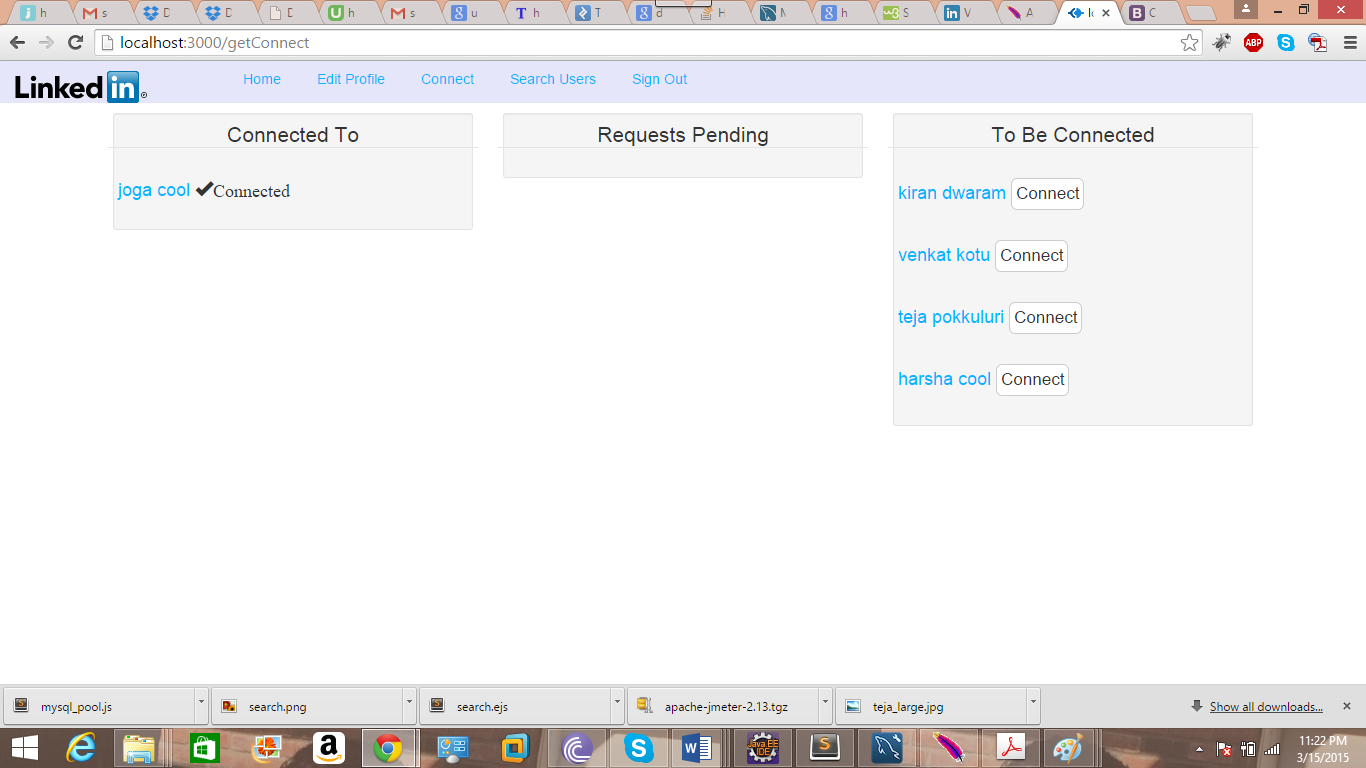
User3 home page:



Accepting request in connection page:



After connection is accepted:



**Questions:**

1. **Algorithm for Data Caching:**

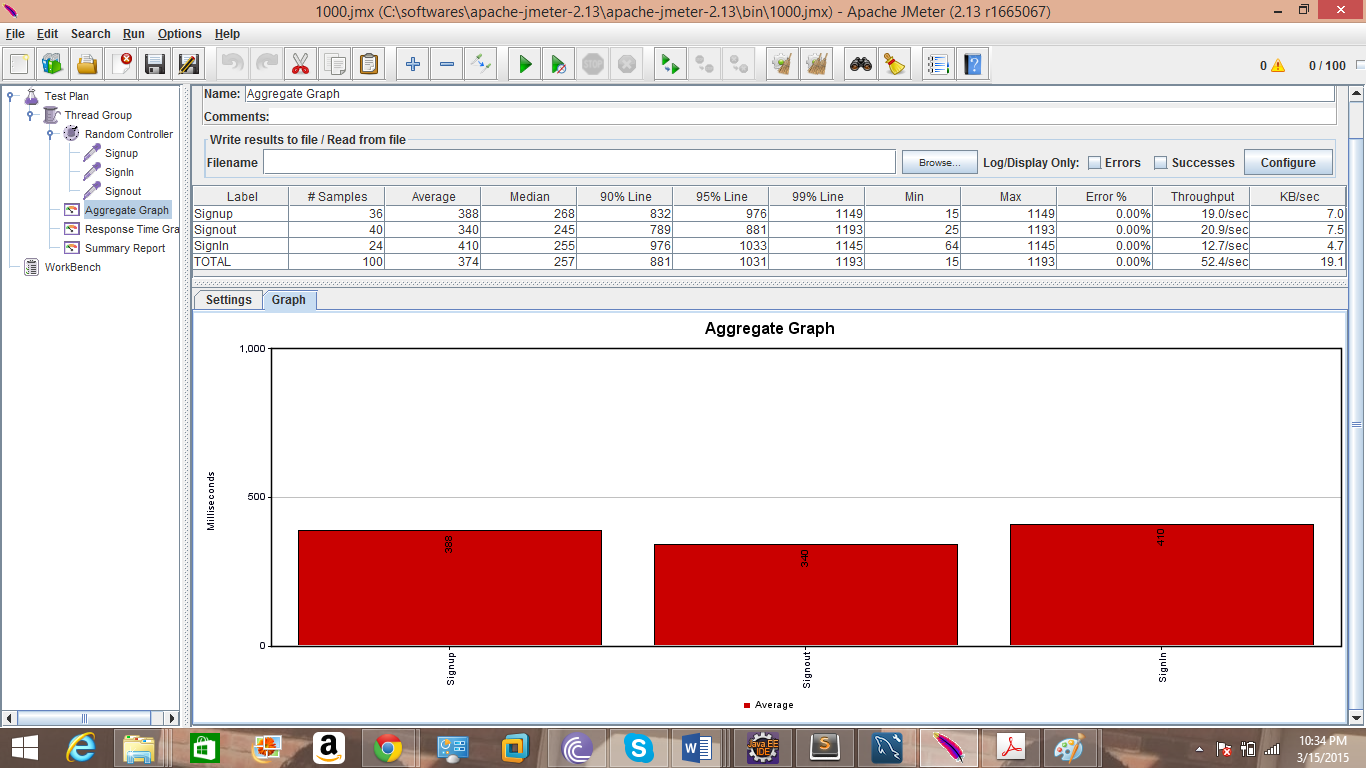
The algorithm and strategy I chose for data caching is LRU (Least Recently Used) strategy.

Because while performing caching the objects that are frequently used should be cached. If the cache memory is full then discard those objects from the cache which are least recently used. This process of discarding Least Recently used objects does not affect the performance of the system as those objects are less frequently used.

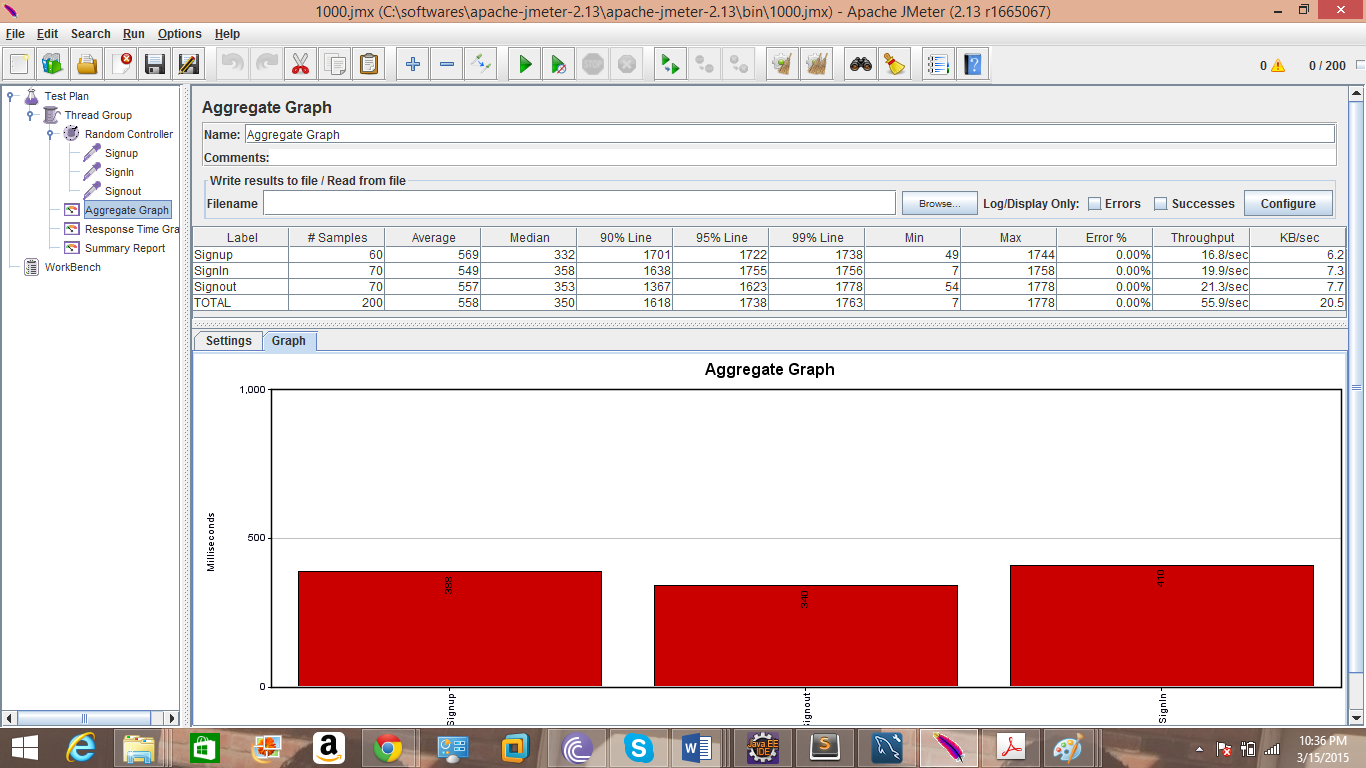
1. **Connection Pooling:**

Without Connection Pooling:

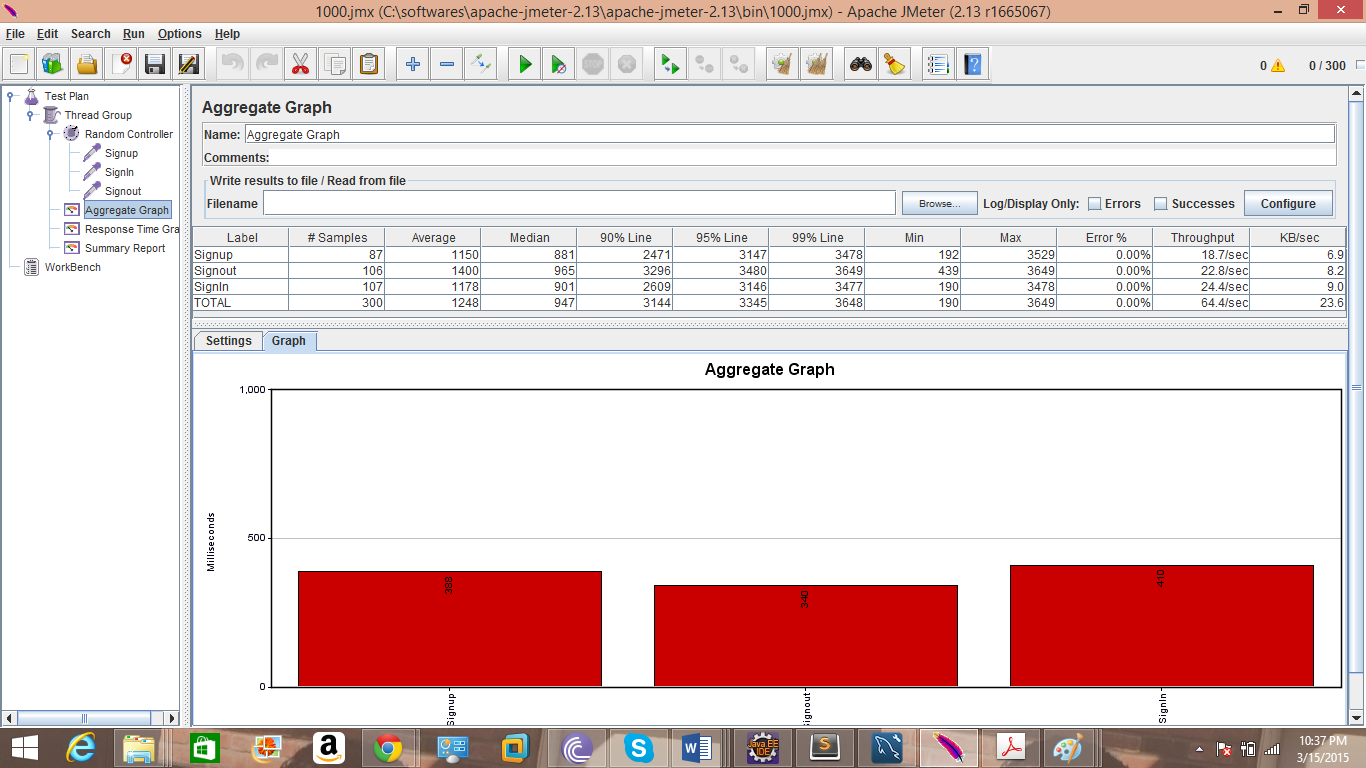
100 users:



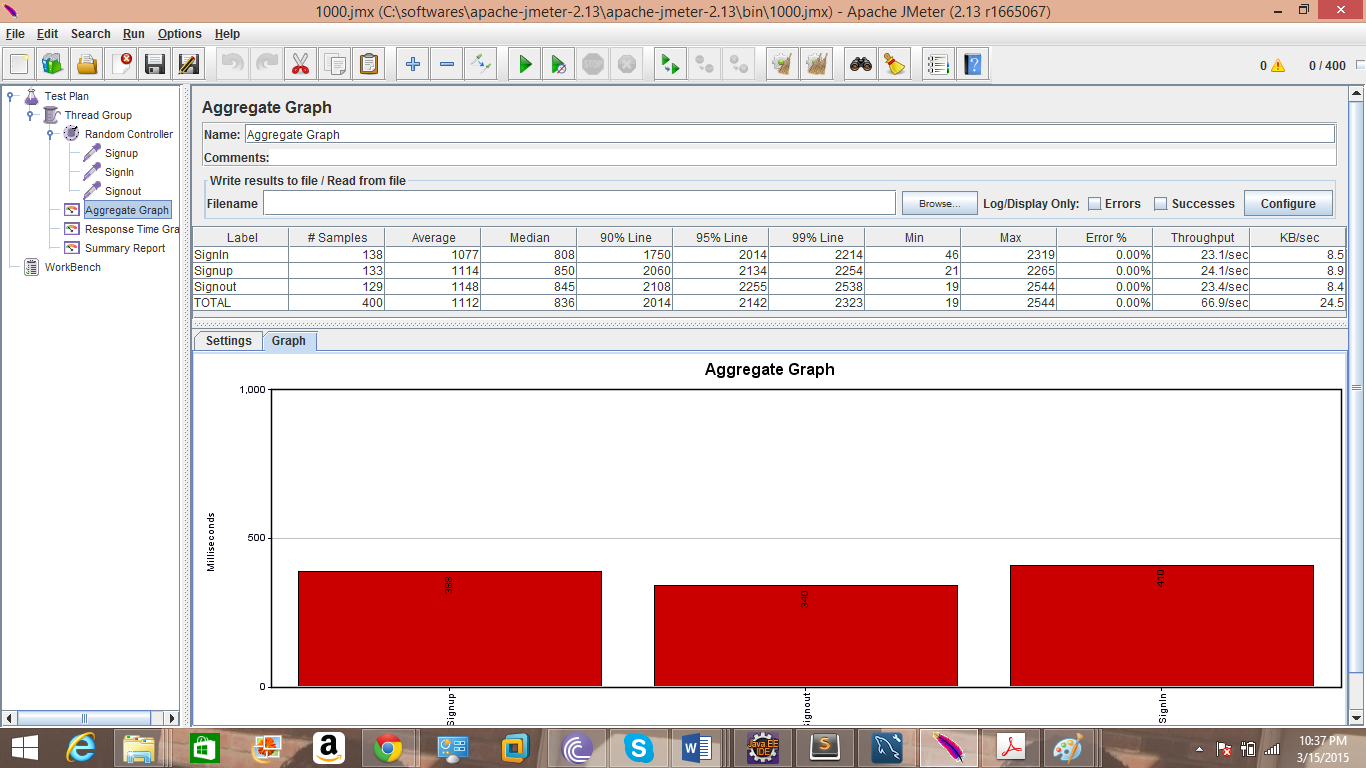
200 Users:



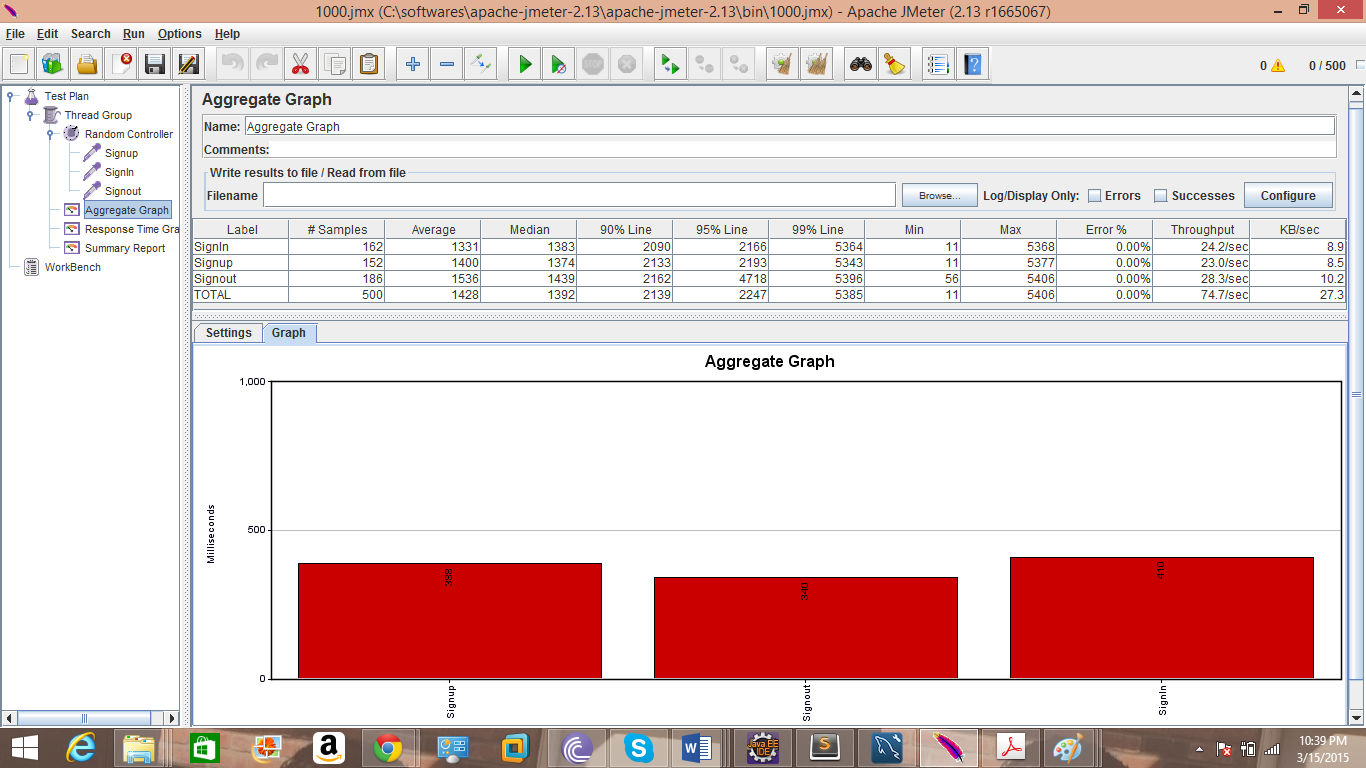
300 Users:



400 Users:

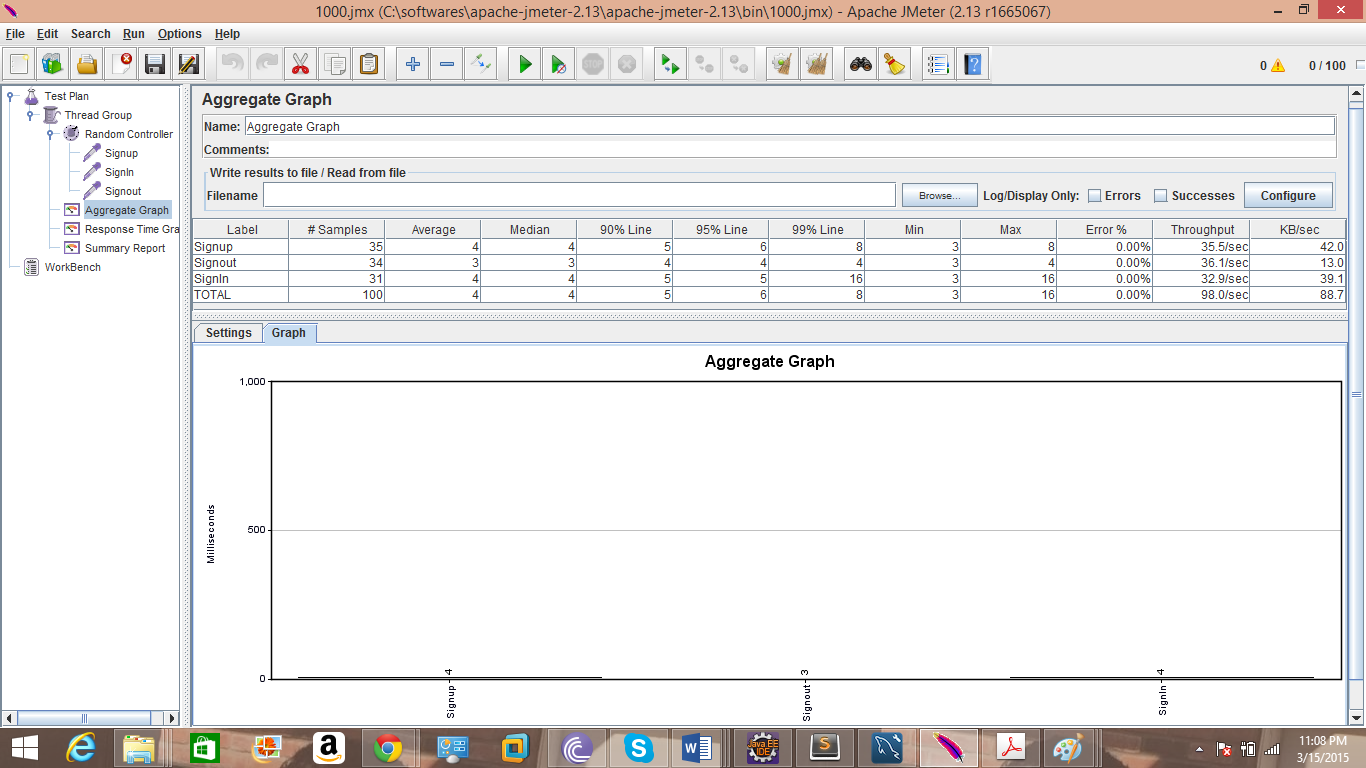


500 Users:

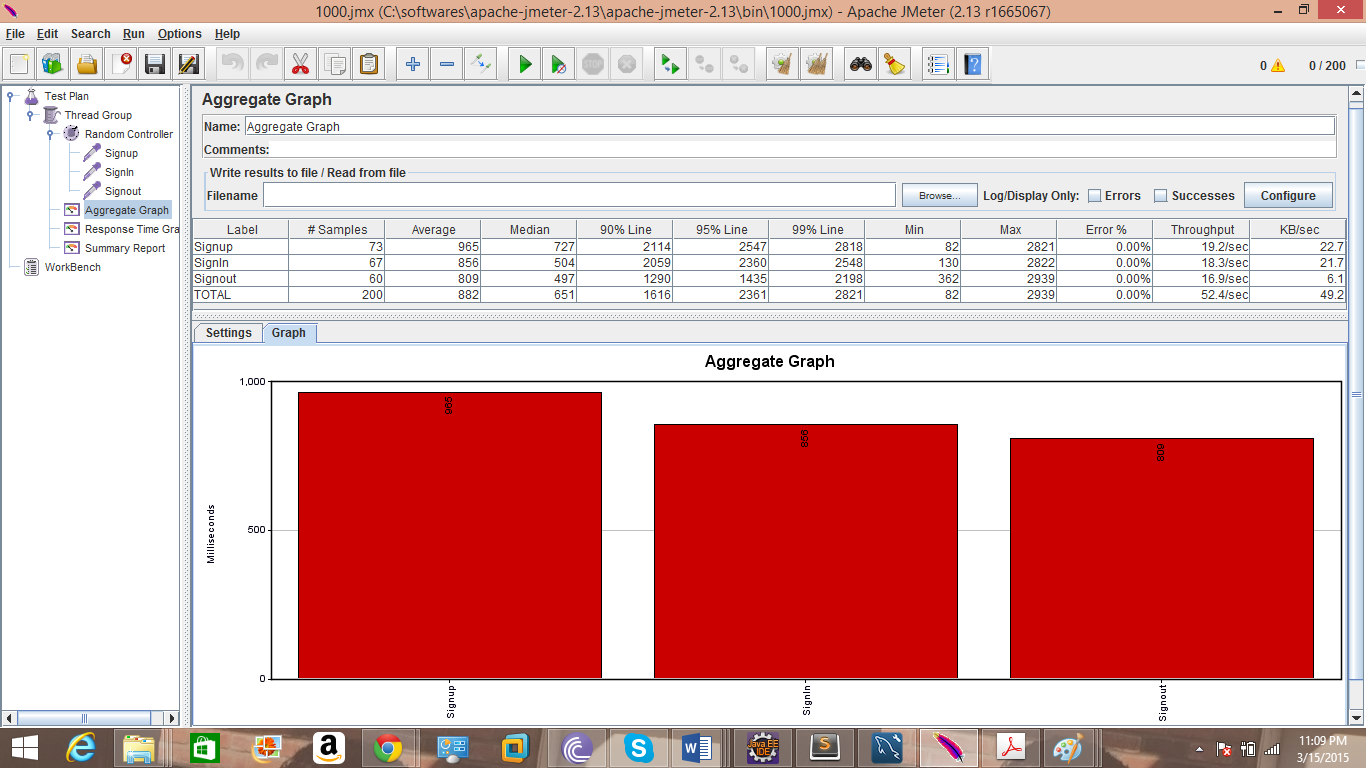


Connection Pooling:

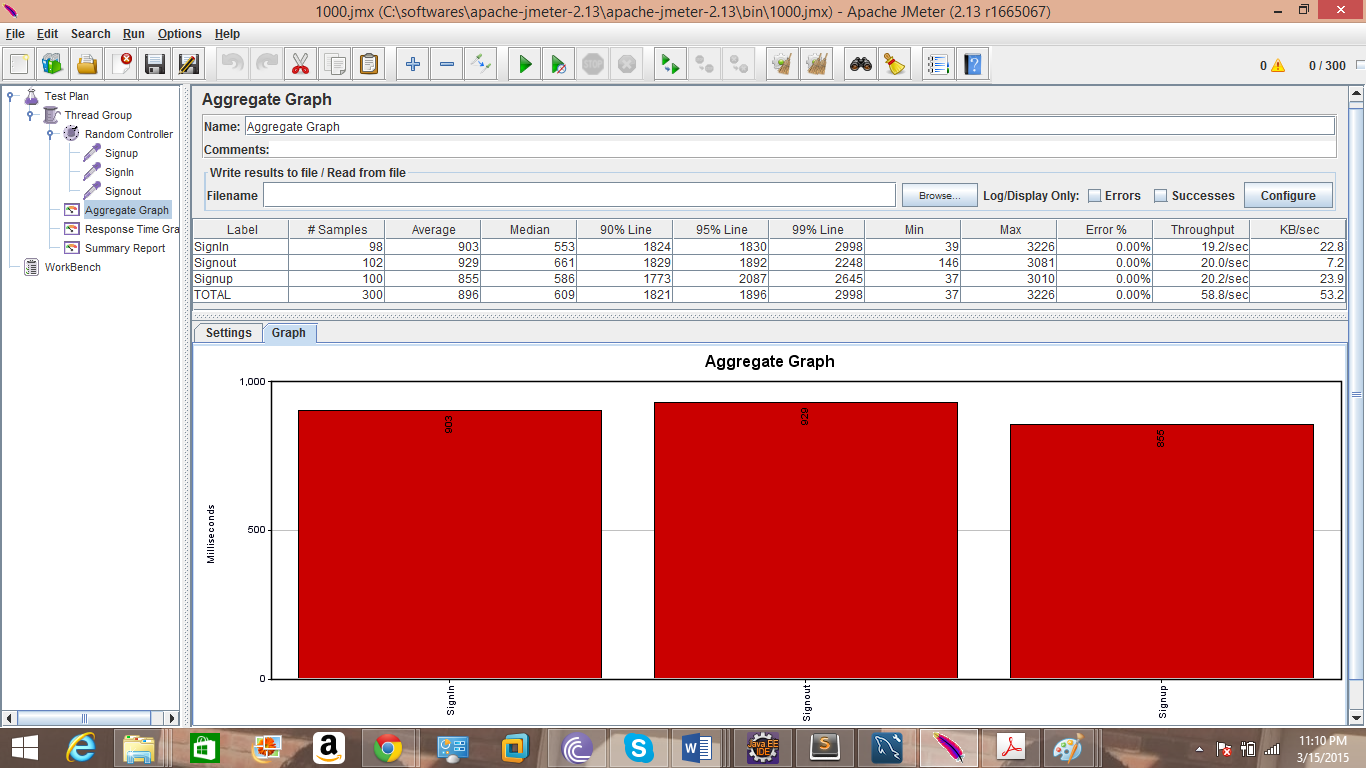
100 users :



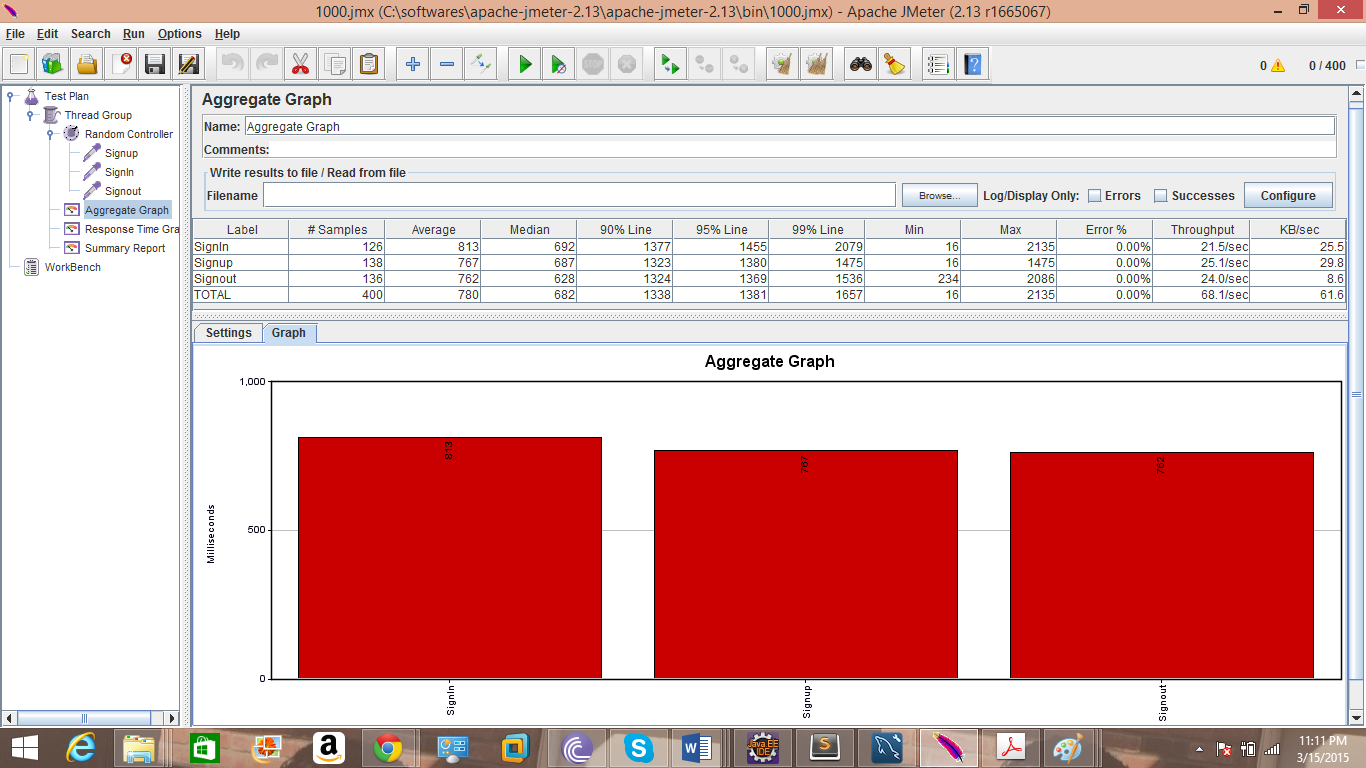
200 users:



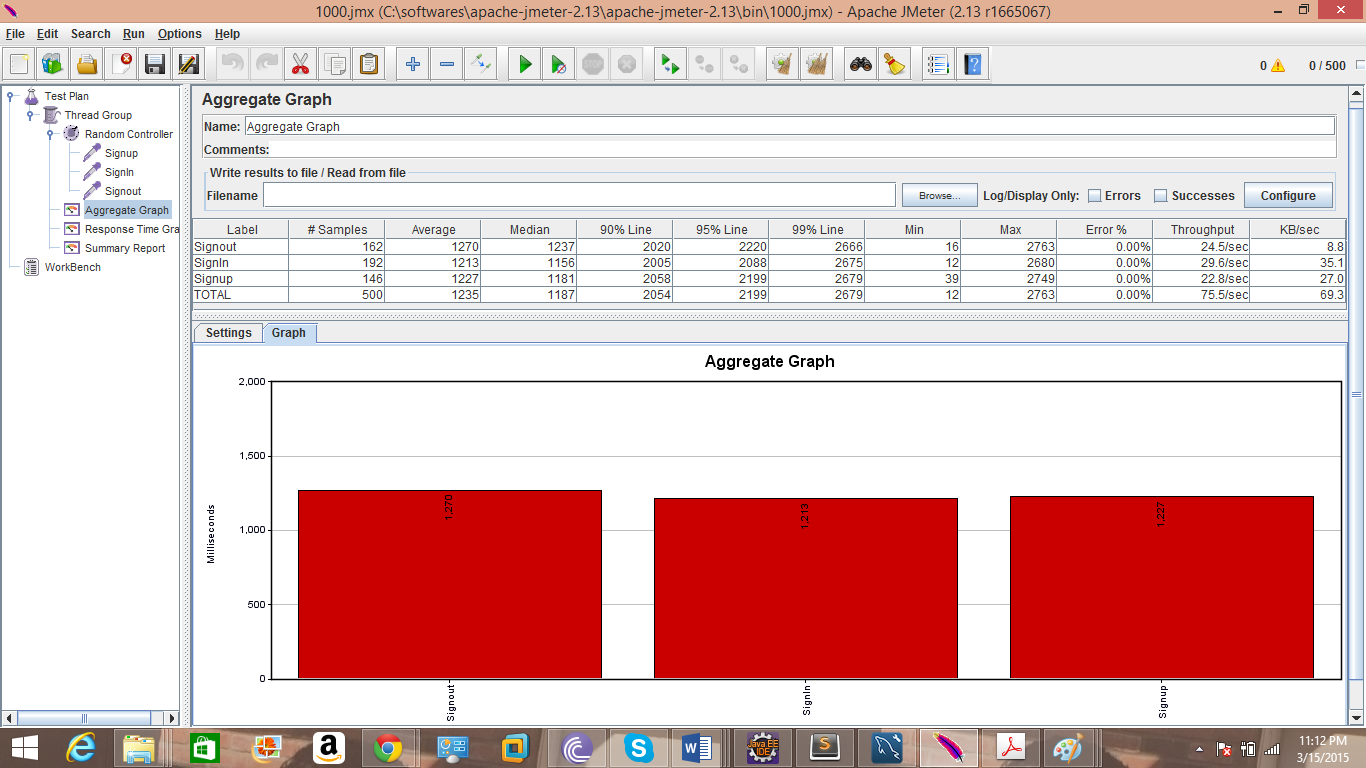
300 users:



400 Users:



1. Users:



**Without Connection Polling** **in milliseconds**

100 users-374

200 users-558

300 users-1248

400 users-1112

500 users-1428

**With Connection Polling in milliseconds**

100 users-4

200 users-882

300 users-896

400 users-780

500 users-1235

1. **Implementation of SQL Request Caching :**
2. Identify the objects to be cached. Cache those objects which are used frequently.
3. Decide on the place where caching should be done like MemCached, Browser Cache etc…
4. Keep track of the objects cached.

If the object is updated in the memory then make sure that it is updated in the Cache simultaneously.

Else If the new object is created or old object is removed from database etc…make sure that those changes are done in cache also.

4. If the Cache is full then keep those objects in the cache that are most recently used than the others.