**Lab 1 Assignment**

Nodejs/Express FrameWork/My SQL/AngularJS

Hema Begur

**Abstract**

This lab assignment covers developing REST services using Node.js (Express) and Angular JS.

**Calculator**

**Server 1/Client 1:**

1. **Server** - **“Calculator” to demonstrate Stateless web services**

The first Node.js based server you need to develop is the “Calculator”. This server should perform the following tasks:

1. Addition

2. Subtraction

3. Multiplication

4. Division

1. **Client -** Client can interact with java script**.**

The client should provide the facility to test all the functions of calculator service. The client for Server 1 should behave in the following way:

Perform each operation of Server once. Print out the result in a systematic format.

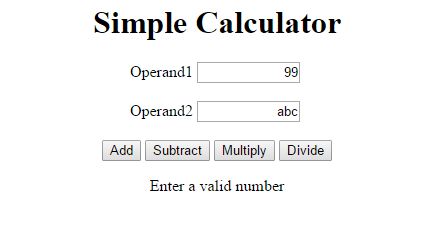
**System Design:** The calculator application is built in such a way that depending on the operator sent from the client, server would direct the call to appropriate mathematical operation. It checks for the different exceptions like the “Divide by Zero error” for division operation. Here the client and the server communicate using AJAX where an asynchronous ajax request to compute a specific operation is sent to the server.

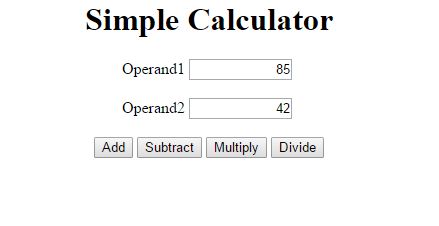
**Results:**

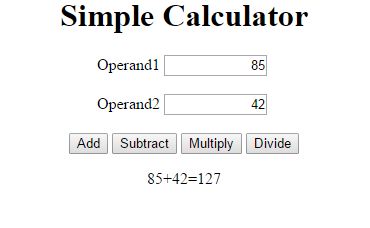
Server Result:

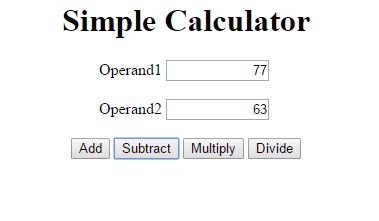


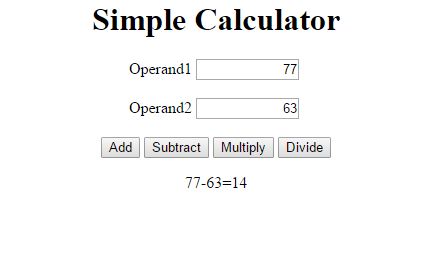
Client Result:

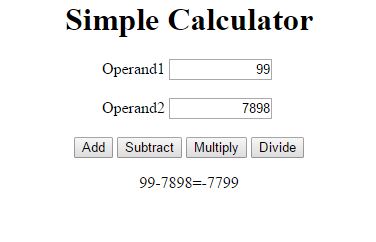


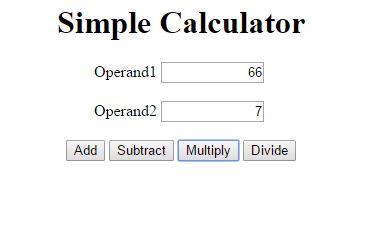


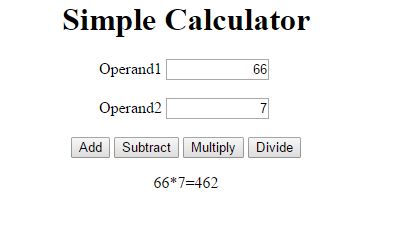


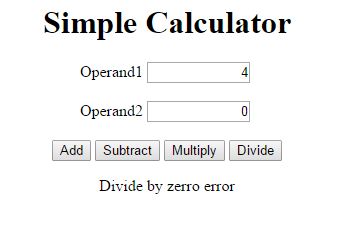


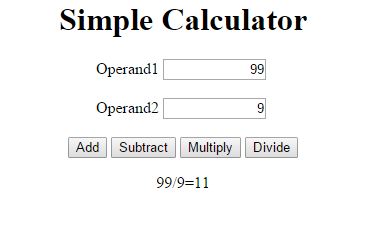
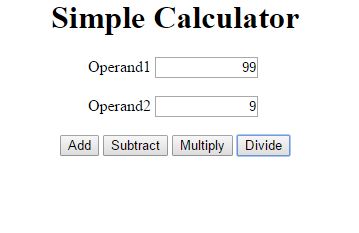










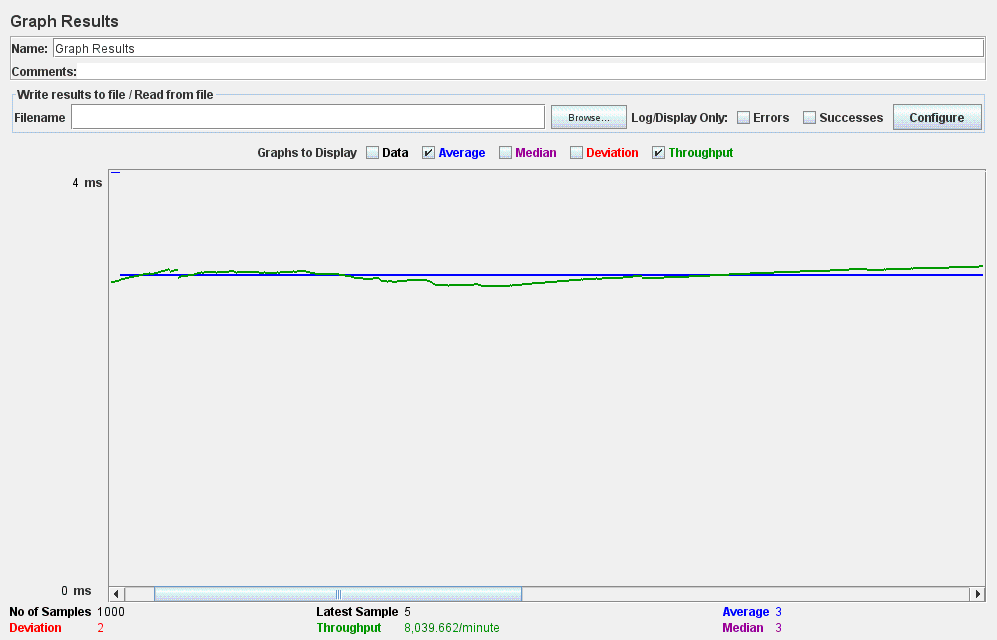


**Performance Testing:**

Tests Using Jmeter:

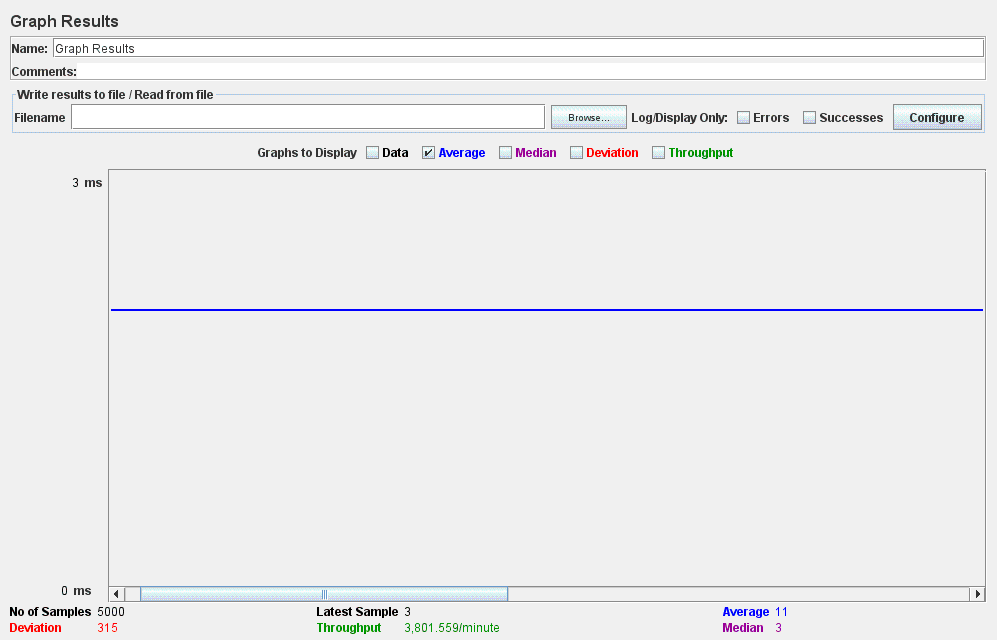
1. 1000 calculator calls on randomly selected tasks

* Addition

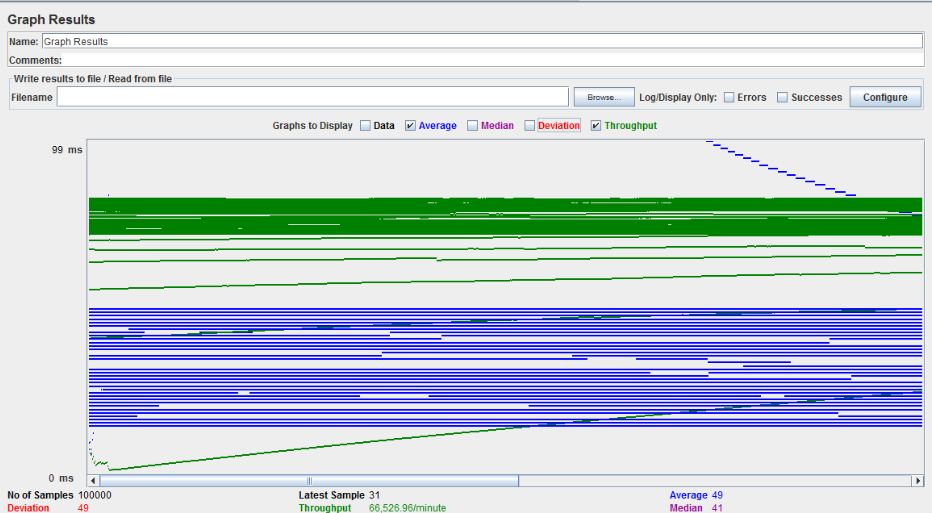


1. 5000 calculator calls on randomly selected tasks

* Multiplication



1. 100 concurrent calls for 1000 **addition** calls to the calculator



**Facebook Prototype**

**Server 2/ Client 2**

**Server**:

a) Basic **Users** functionalities:

1. Sign up new user (at least first name, last name, Email and password)

2. Sign in existing user

3. Sign out. Sign Up should have first name, last name, Email and password.

In order to use the system, a user must sign in first to the system.

b) **Users account** should provide basic details such as:

1. **About**: User overview, Work and education, contact info and life events.

**2. Friends list**: Send friend request, accept friend request.

3. **Interests** like music, shows and sports.

c) Provide **news feed** functionality.

d) Provide **Groups** functionalities:

1. Create group

2. Add member in a group

3. Show members in group

4. Delete member from a group

5. Delete group

e) Should perform **connection pooling** for database access.

**Client**:

Test client can interact with java script. The client should provide the facility to test all the functions of “Facebook application” service up to 500 concurrent users. Show the graph with 100, 200, 300, 400, 500 users without connection pooling and with connection pooling.

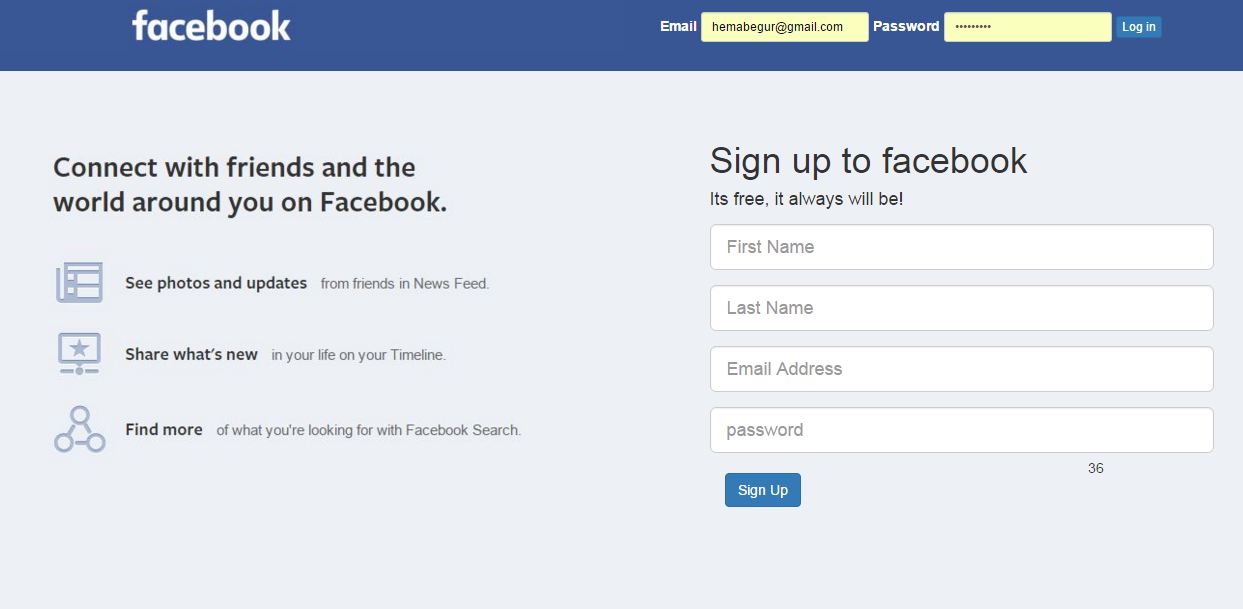
**System Design:** The system design for Facebook prototype is as follows:

* Front end developed using Angular JS, Bootstrap, HTML and CSS.
* **RESTFUL Web services** followed to develop the application.
* **Node Js** is used as the server side technology.
* **MYSQL DB** – Mysql database is used to persist all the data as restful services are stateless and don’t allow data persistence
* **AJAX** – Ajax is used making client server communication asynchronously for making the system efficient and good on performance.
* **Connection Pooling:** By

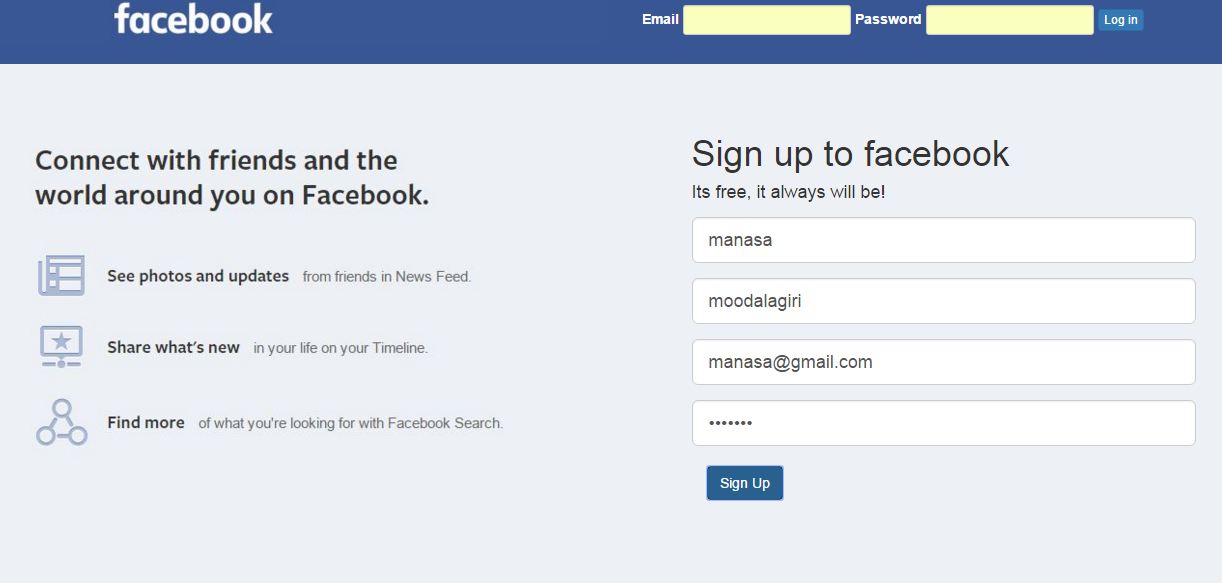
**Results**

**Login/ Sign Up Module/ Last Login Tracking**

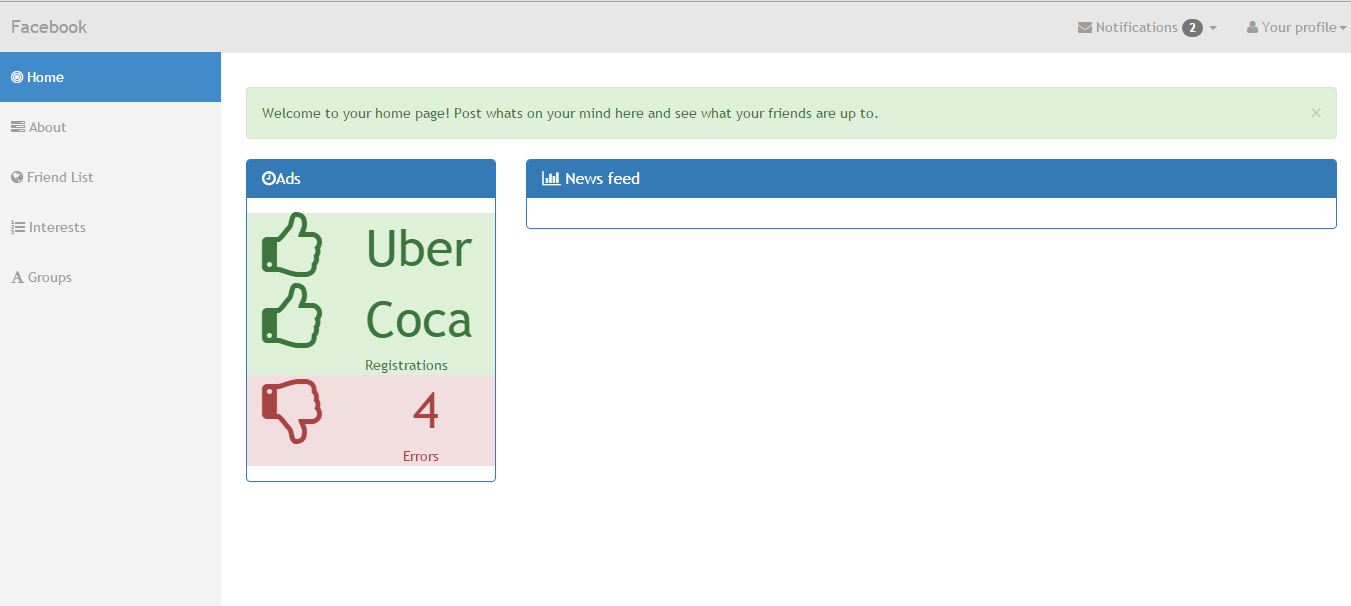
Login Page



**Signup page**

****

**Home page**



**QUESTIONS:**

1. **Data caching:** Caching in node for data can be effectively to save operation time with every request being made to server to get data user. We can implement caching in node-JS using node-cache/cache-manager very effectively.

Install cache-manager package from npm repository. It allows wrapping of functions in cache, tiered caches and consistence interface.

Why Data caching is required?

**Server Side:** heavy duty operations that compel to employ data caching Like - Streaming in static contents from file, static content of html, global values and DB calls.

**Client Side:** Context sensitive, client constants and resource bundles.

**Server Caching Strategy & SQL server caching strategy**

Test code :

function get\_cached\_user(id, cb) {

memory\_cache.wrap(id, function (cache\_callback) {

get\_user(id, cache\_callback);

}, cb);

}

Methods :

set(key, val, cb)

get(key, cb)

del(key, cb)

Size () - Returns the number of key value pairs present in the cache

Memsize () – Retrieves the number of key value entries taking up the space in cache

Hits () – Returns the cache hits

Misses () – Returns the number of cache misses

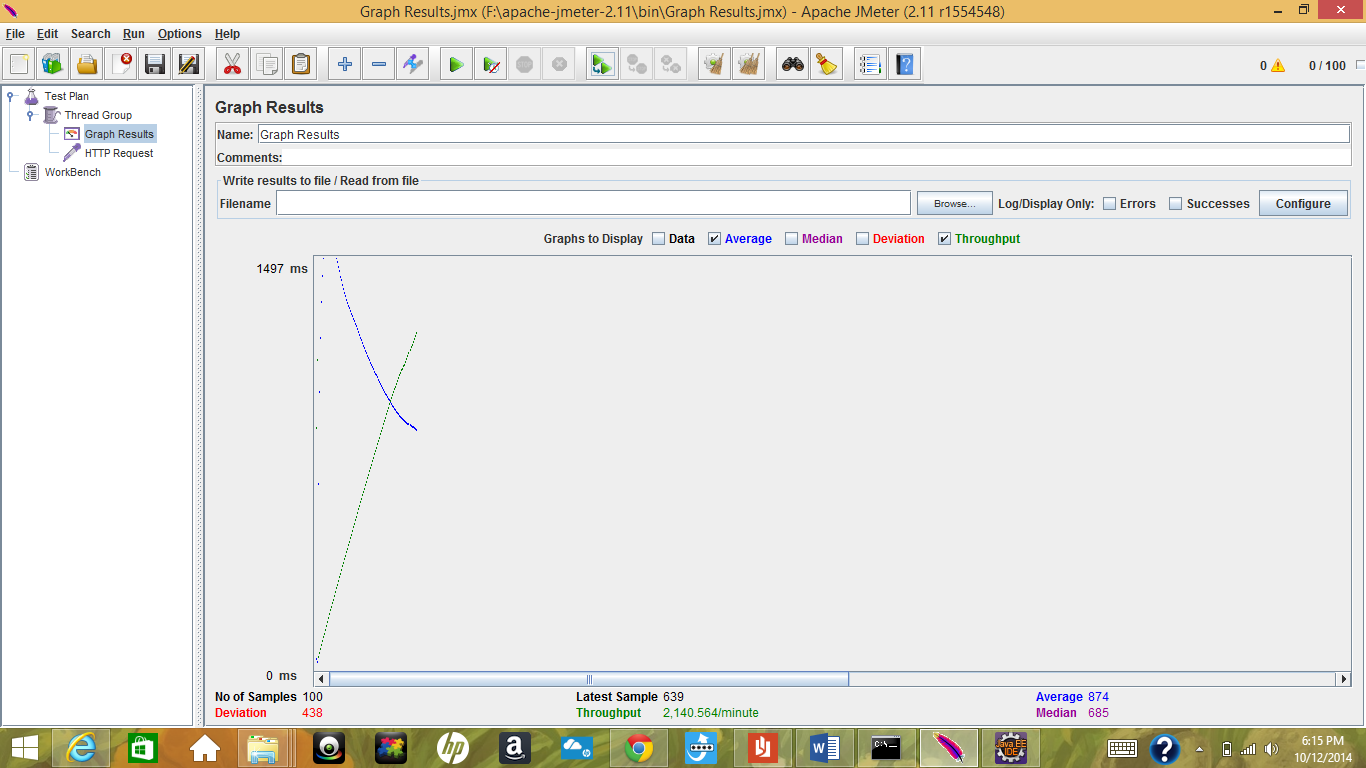
You can use single store or multiple store as you require. Also we can create custom store as required.

**Client Caching Strategy**

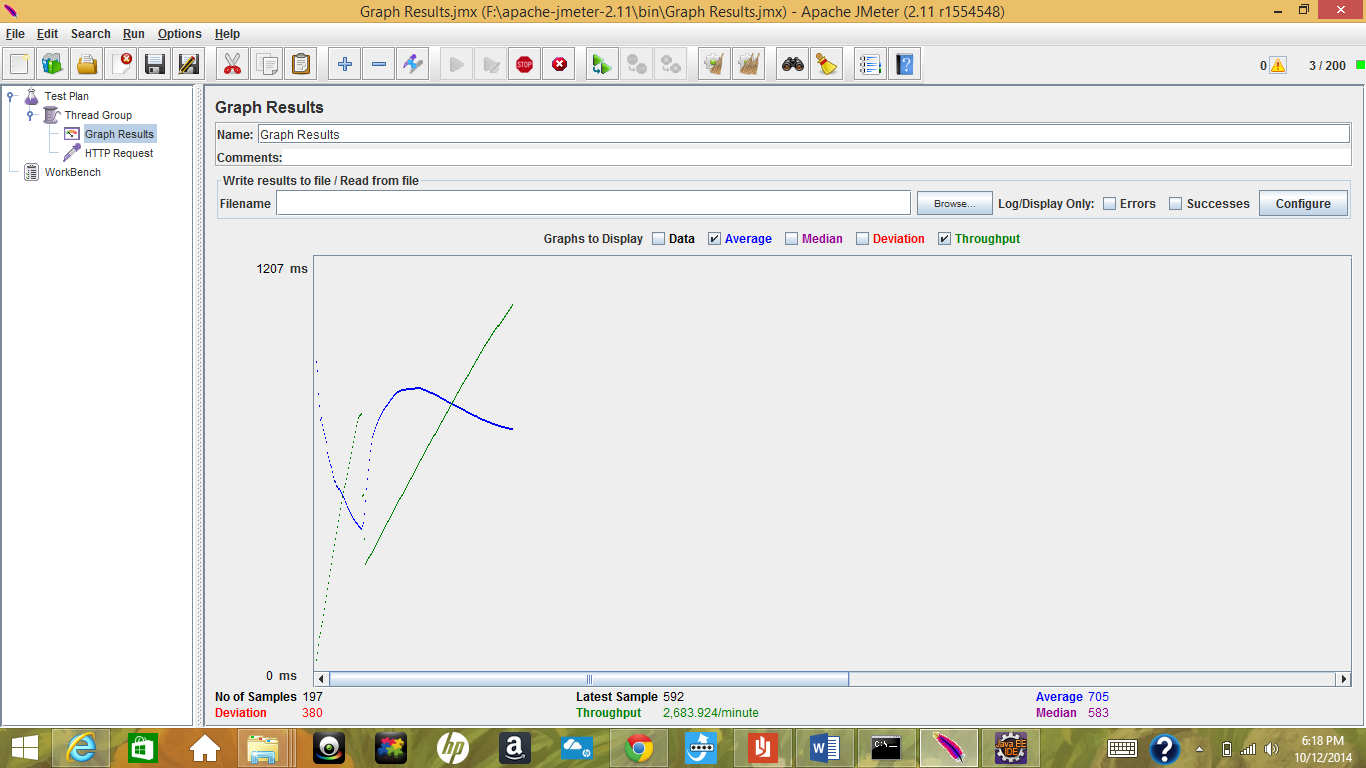
* Local storage – Data is stored and persisted over different pages and also on closing of pages
* Session Management – It helps maintain session and the user data in the cookies and it is stored and persisted over every top level browsing context.

**Performance characteristics WITH & WITHOUT Connection Pooling**

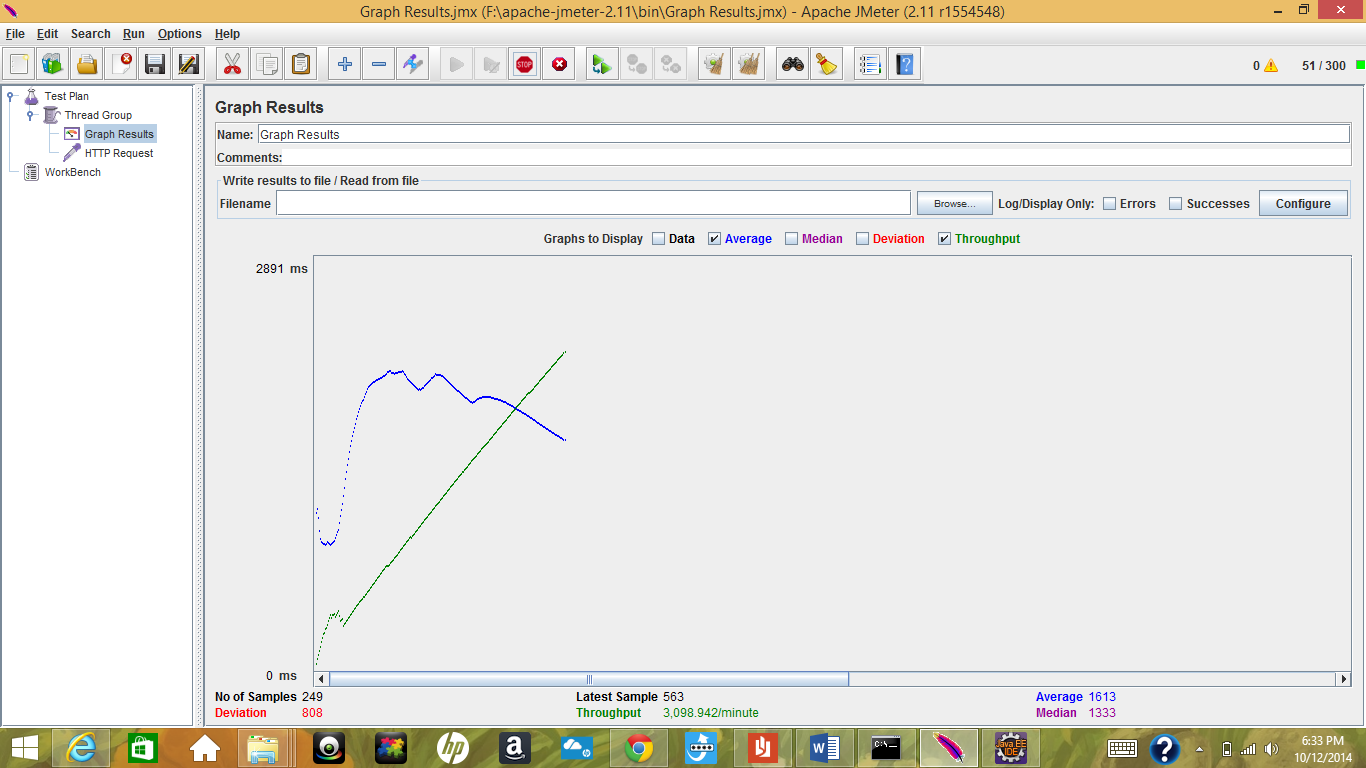
**Without Connection Pooling**: ￼

Users : 100

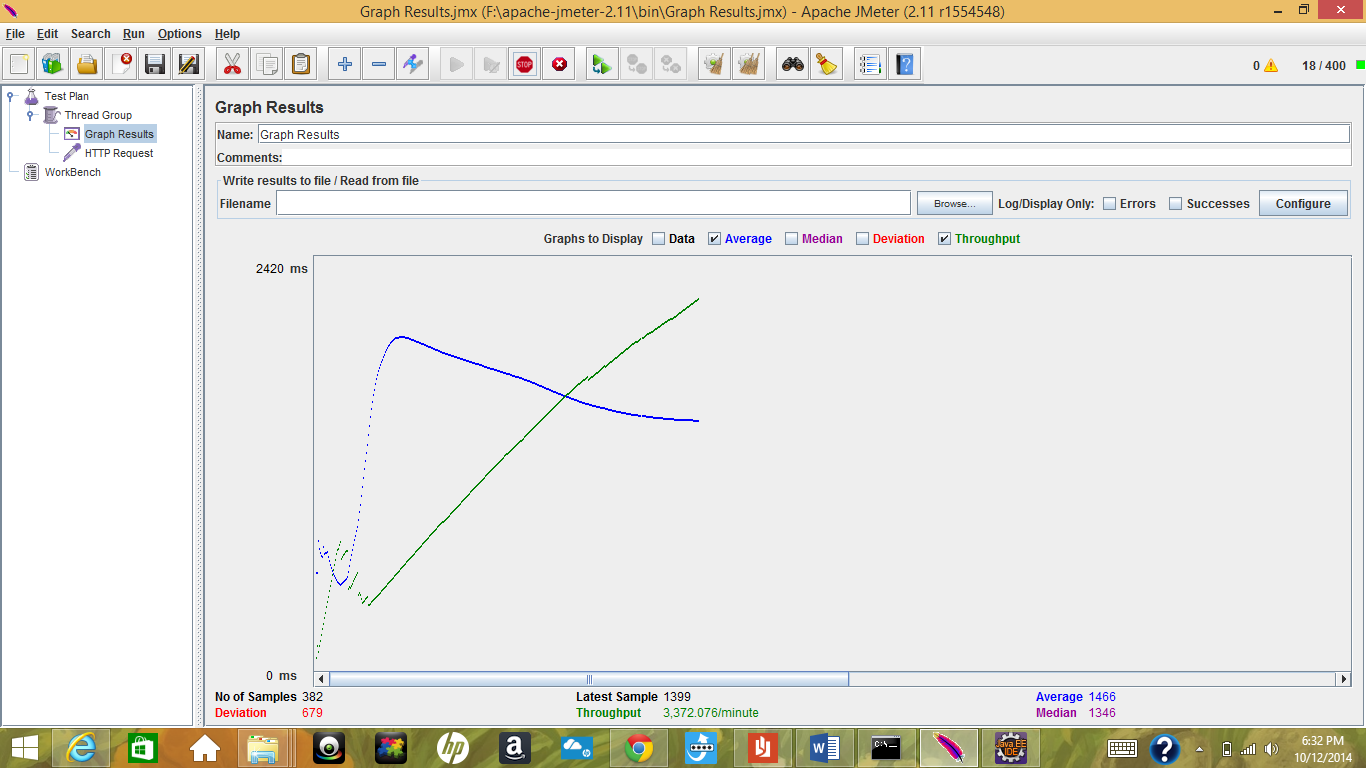
Users : 200



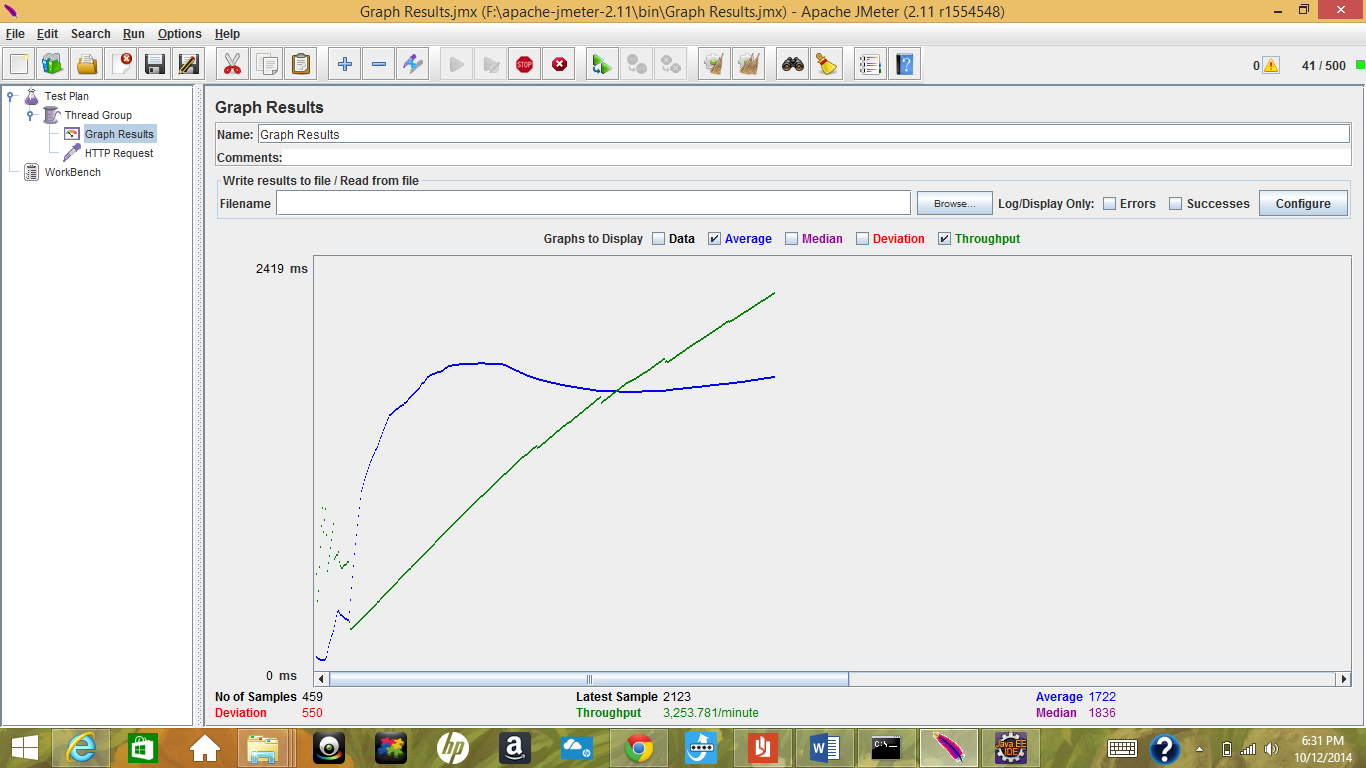
Users : 300



Users : 400

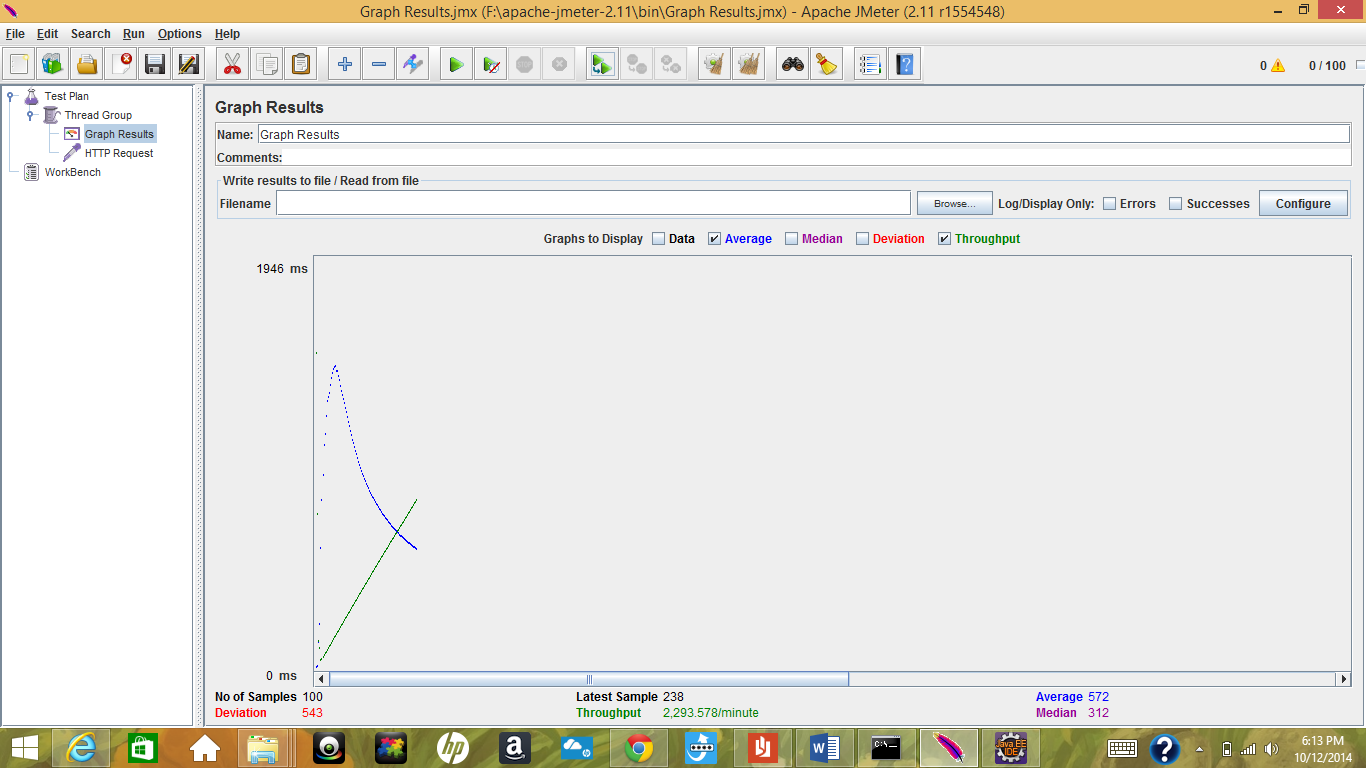


Users : 500

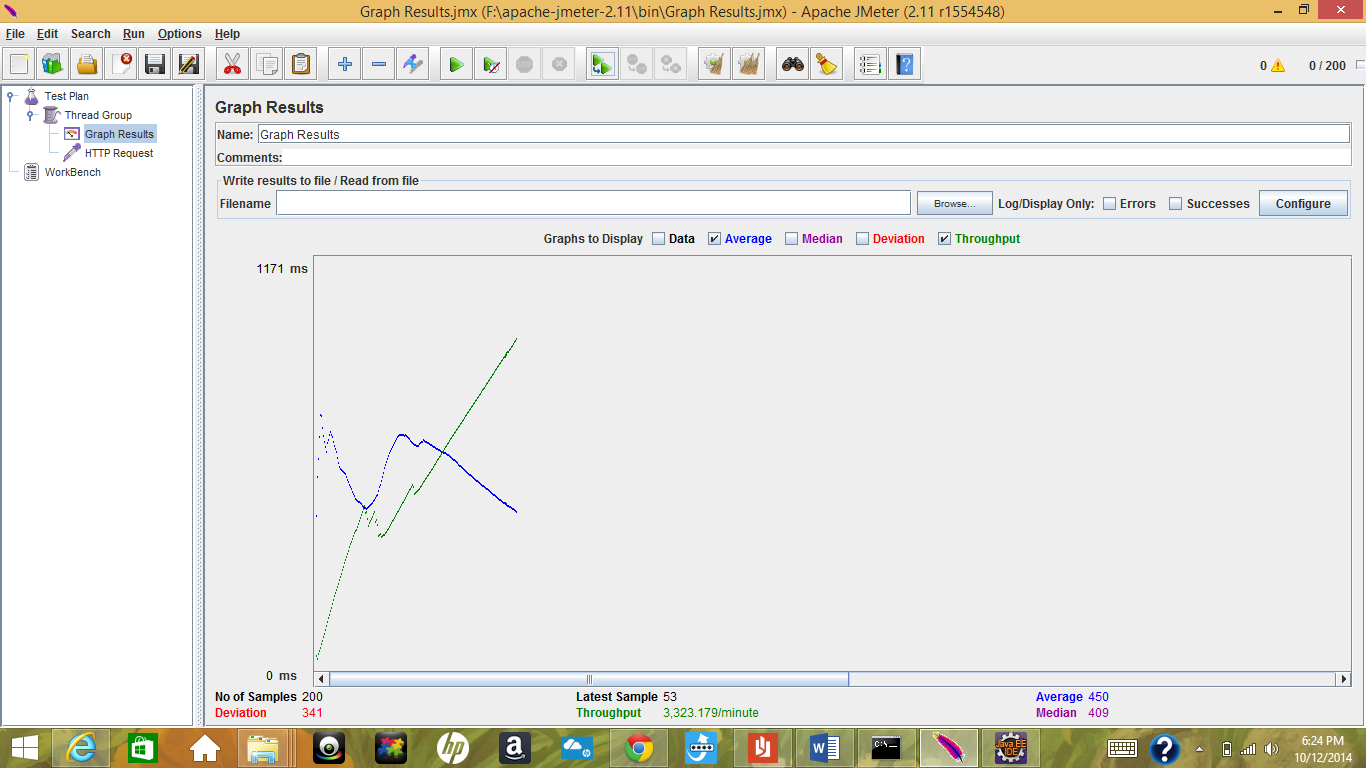


**With Connection Pooling**

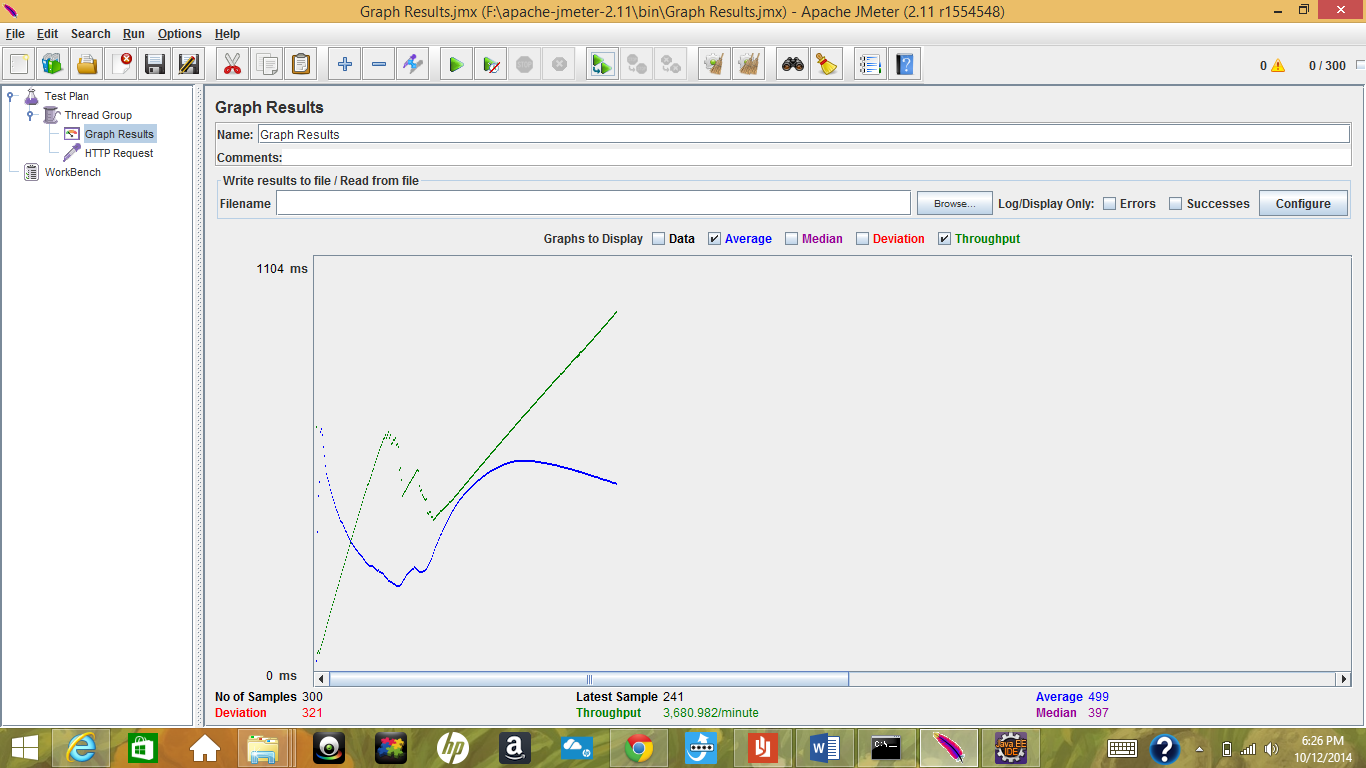
Users: 100



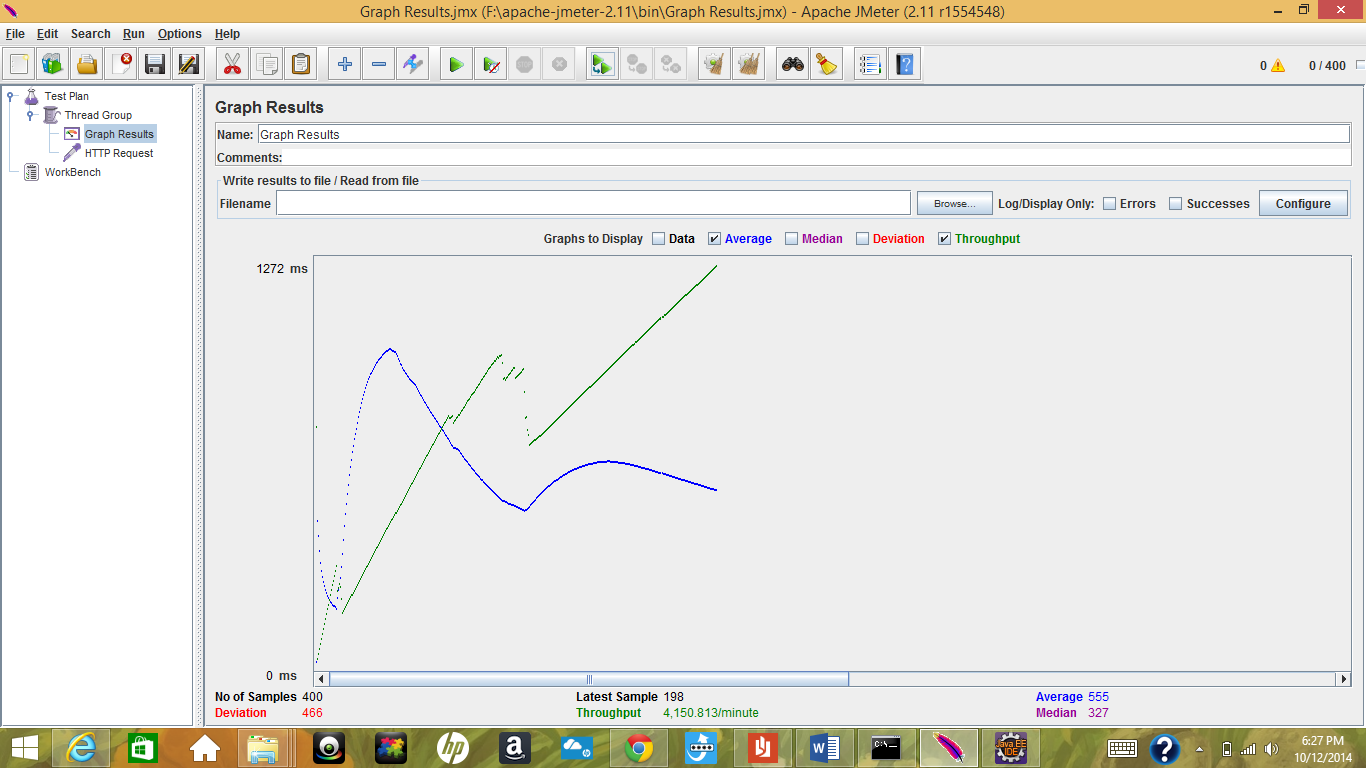
Users: 200



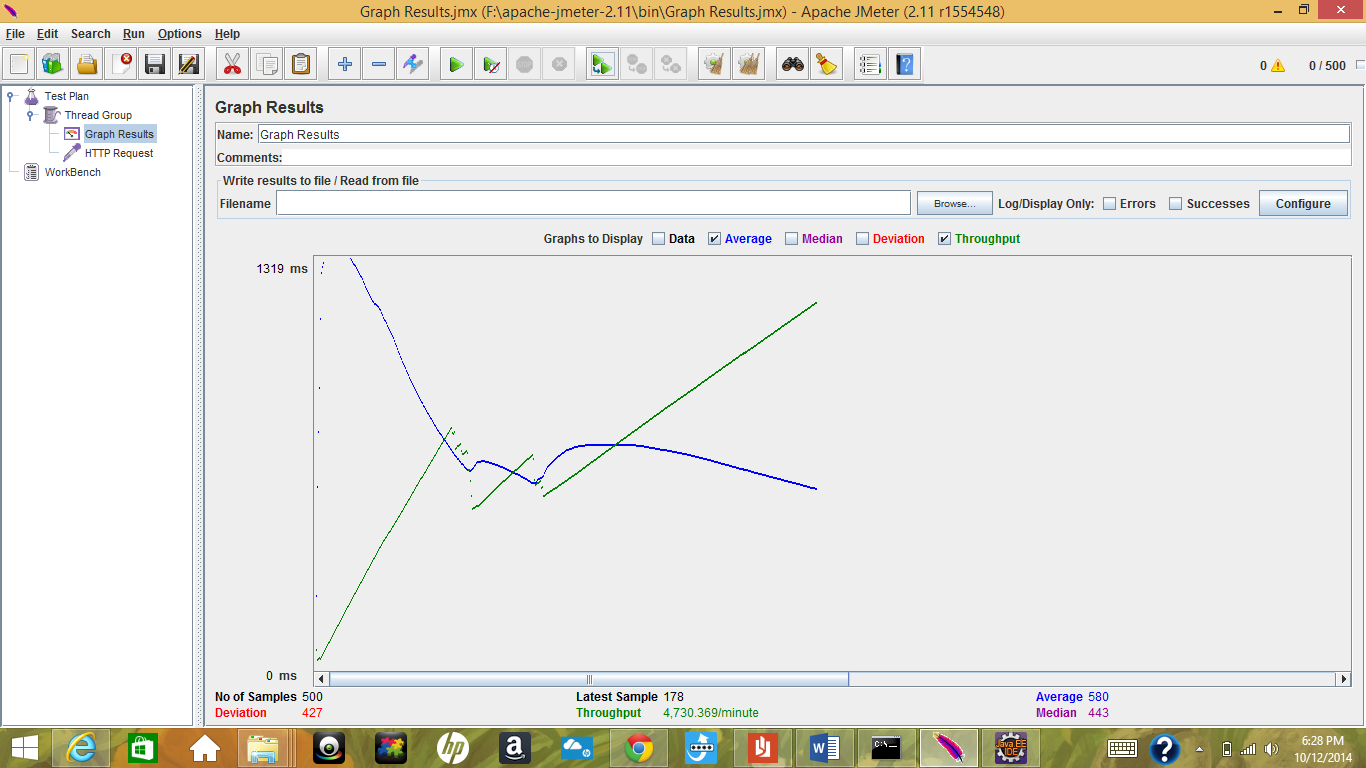
Users: 300



Users : 400



Users : 500



Following are the advantages for the node-pool:

* It reuses the database connection
* Limits the number of concurrent connections
* Does not keep a connection open indefinitely. It is immediately released to the connection pool once the demand is over. This enables recycling and re-usage of connections
* No worry of closing connections explicitly and thus taking care of open connections and exception pertaining to the same.

**SQL Request Caching** :

Query cache stores the text of the SEECT statement along with the result associated with it. If an identical or same statement is received, server gets the output from the query cache rather than executing again. Query cache is shared among sessions. Query cache does not returned slate data. When data is modified the caches are flushed.

To implement it if data caching is done then we need to store SQL Statement with select option in appropriate cache objects with the data set responses from DB. Further if any of the request made by client is identical to saved query then it will respond to that request instead of sending it to DB server.

1. Is your session strategy horizontally scalable?

No, It is not handled as horizontally scalable. If more number of additional servers are added, then you need to store sessions across server .Before your application routes can use session data in a request, the data needs to be looked up. This is done by running each request through session middleware. Since sessions depend on cookies, the request first needs to run through cookie parsing middleware

1. Explain how node.js handles multiple client requests simultaneously even though it is single threaded. Explain using block diagram.



Node.js is event driven, handling all requests asynchronously from single thread.

When A and B are received on apache, two threads are created which handle requests. Each handling the query separately, each waiting for the query results before serving the page. The page is only served until query is finished. The query fetch is blocking because server cannot execute the rest of thread until it receives the result.

In node, request is handled asynchronously, which means while it fetches the results for A, it jumps to handle another request, and when the results arrive for A arrive it sends back the results to callback which sends the response. Node.js knows to execute callback when fetch finishes.

Actually the node server does exactly that for you all the time. To make switches, (the asynchronous behavior) most functions that you would use will have callbacks.

Node JS Web Server internally has a Component, known as “Event Loop” which uses single thread. It checks if any Client Request is placed in Event Queue. If no, then it waits for incoming requests.If yes, then it picks up one Client Request from Event Queue and starts the process which client requests.