San Jose State University
Fall 2017

Project Abstract

LAB 2 –Dropbox Application

CMPE 273

SUBMITTED BY –

Siddharth Suthar(011439636)

SUBMITTED TO-

Prof. Simon Shim

Enterprise Distributed System Lab 2 Report

Introduction

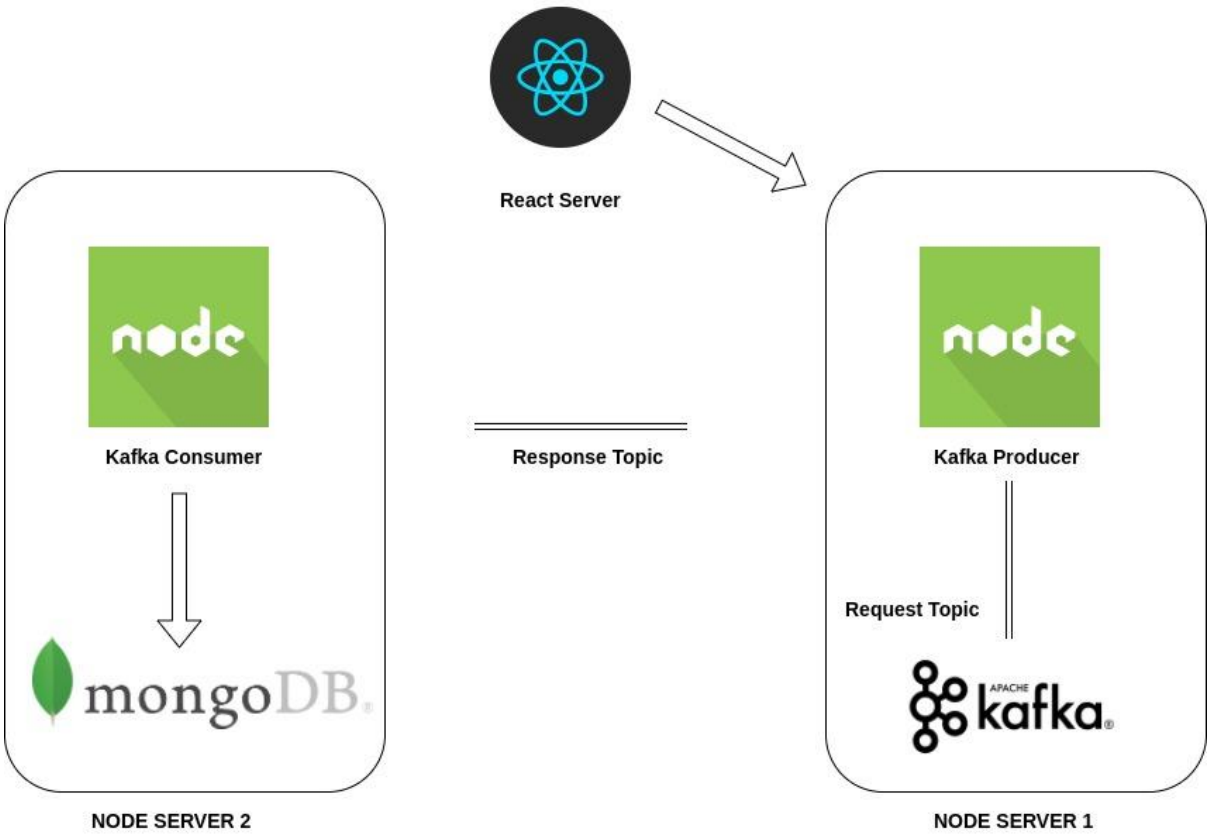
Goal :

The goal of this assignment is to understand and implement the basic concepts of distributed system. To understand how horizontal and vertical scaling works. And to implement the business logic of a Dropbox like application in MERN stack using a kafka queue as a middleware. And implement security designs to prevent it from cyber attacks and sql injections and cross site scripting.

Purpose :

To develop a Dropbox like application that performs the basic functionalities of the Dropbox application like File Upload , File Download, File Share , User Creation , Group Creation Displaying user Info , user logs , maintaining the state , and encryption of Password using salt and hash. The idea is to understand and implement the concepts and architecture of the application in ReactJs , Redux , NodeJs, Kafka queue , and Mongodb. Kafka Queue is used to make the system distributed and allow it to be horizontally scalable.

System Design



As mentioned in the architecture , The front end part(Client side) of the application is developed using the combination of React and Redux, The Front end part sends an API request to the Node Server 1 where the request is handled by the node server ie (Kafka front end) which parses the request , handles the sessions and then generates a CorrelationId and assigns the message that ID and puts it on the Kafka queue(Topic). Once the message is on the Kafka queue the Kafka front end also known as Kafka producer can continue to serve other requests. Once the message is on the Kafka Queue , also known as a Topic, the designated consumer running on the backend server consumes the message on the topic which it listens to and processes it and then responds with the same CorrelationId , so now the backend or the Kafka Consumer acts as a Producer and puts the message on the Kafka response topic and the consumer running on the Node server 1 consumes the message and then further passes the message to the Front end part ie the React Server

This is how the basic communication occurs between front end and backend.
The distributed part of the system is that now since the message is passed on the Kafka Queue , the same code can be copied on multiple servers and the can act as the backend servers , ie it provides horizontal scaling.

Functionalities Implemented –

• Basic Users functionalities:
    - Signup new user
    - Sign in existing user
    - Sign out
    - Upload a file
    - List a File
    - Create a Folder
    - Star a File
    - Share a File
    - Share a Folder

- Users account should provide basic details such as:
    - About
    - Interests

- Provide Group Share Functionalities
    - Create Group
    - Add Member in the Group
    - Show Members
    - Delete Member from a Group
    - Delete Group

- Should Perform connection Pooling.

API's Created :

Method : POST

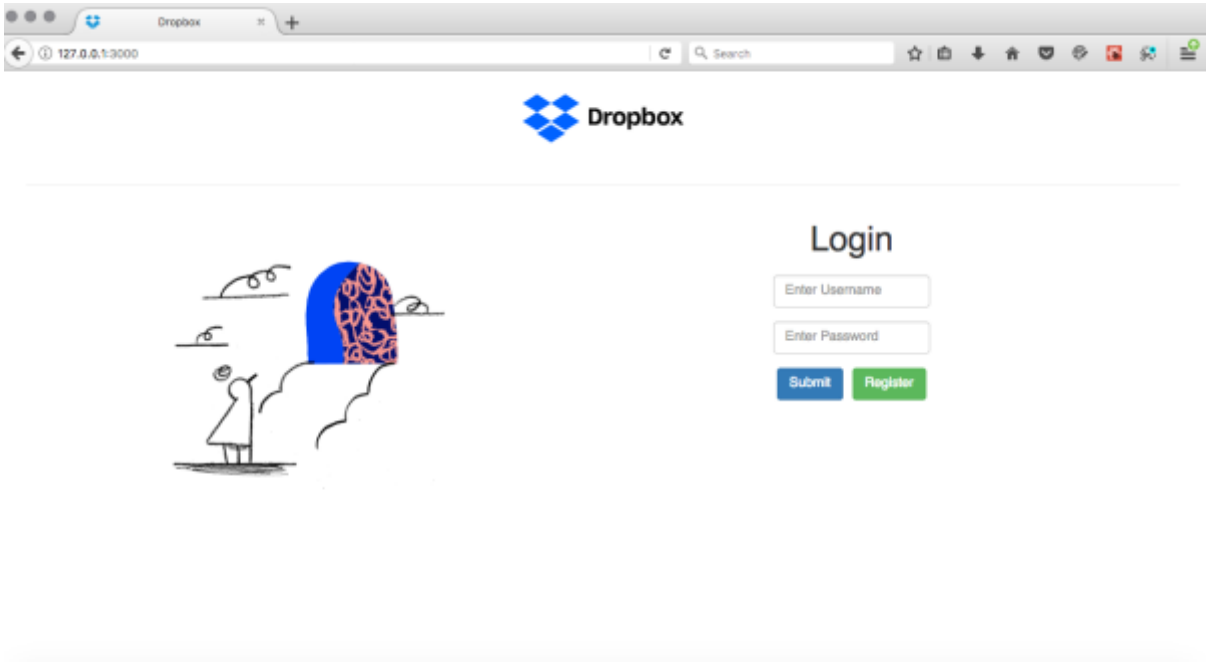
- Localhost:3001/users/login
- Localhost:3001/users/doLogout
- Localhost:3001/users/doRegister
- Localhost:3001/users/validateUser
- Localhost:3001/files/share
- Localhost:3001/files/upload
- Localhost:3001/files/delete
- Localhost:3001/files/uploadFolder
- Localhost:3001/files/deleteFolder
- Localhost:3001/users/createGroup
- Localhost:3001/users/addMember

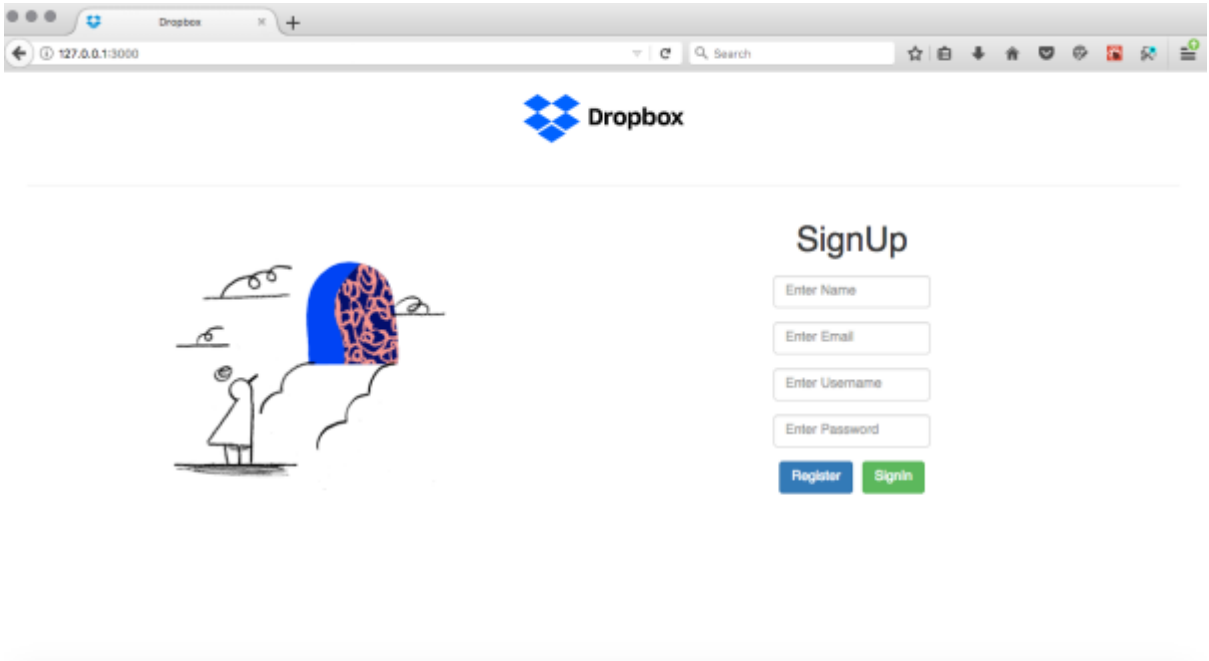
Method : GET

- Localhost:3001/users/getFiles
- Localhost:3001/users/groups

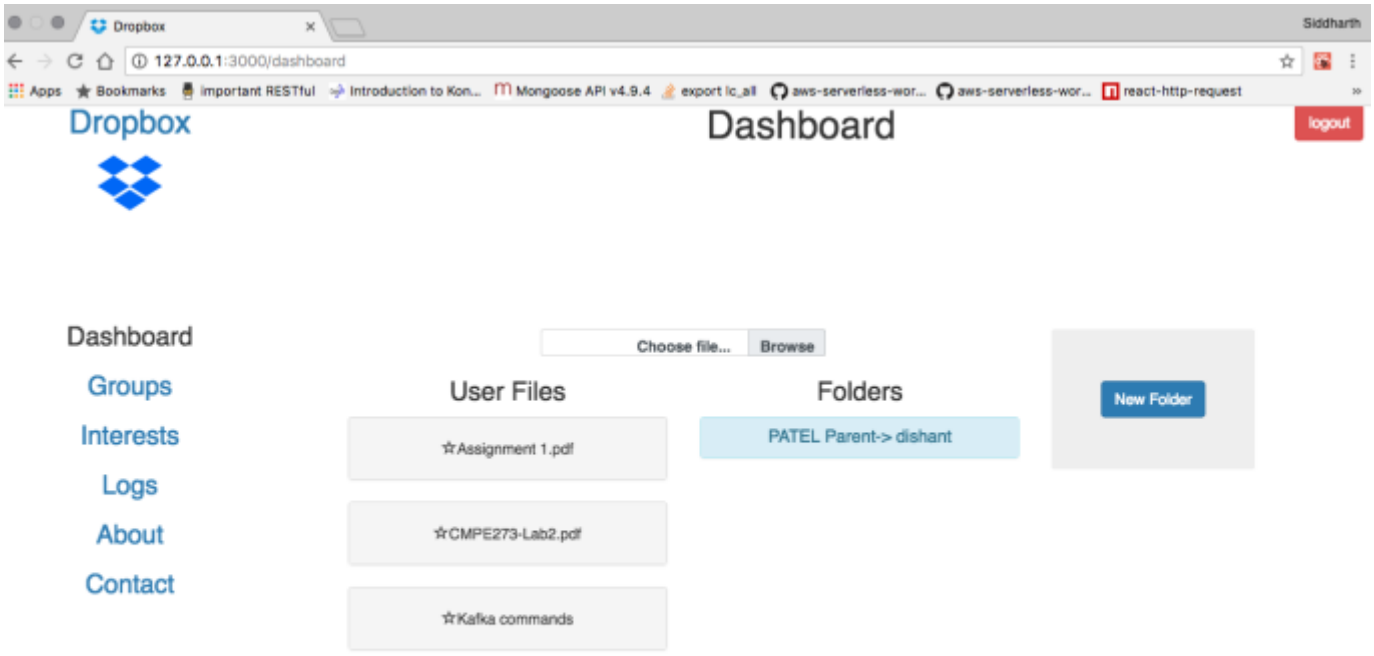
Screen Shots :

1) Sign In and Sign Up Page : User can Login or Sign Up on this page

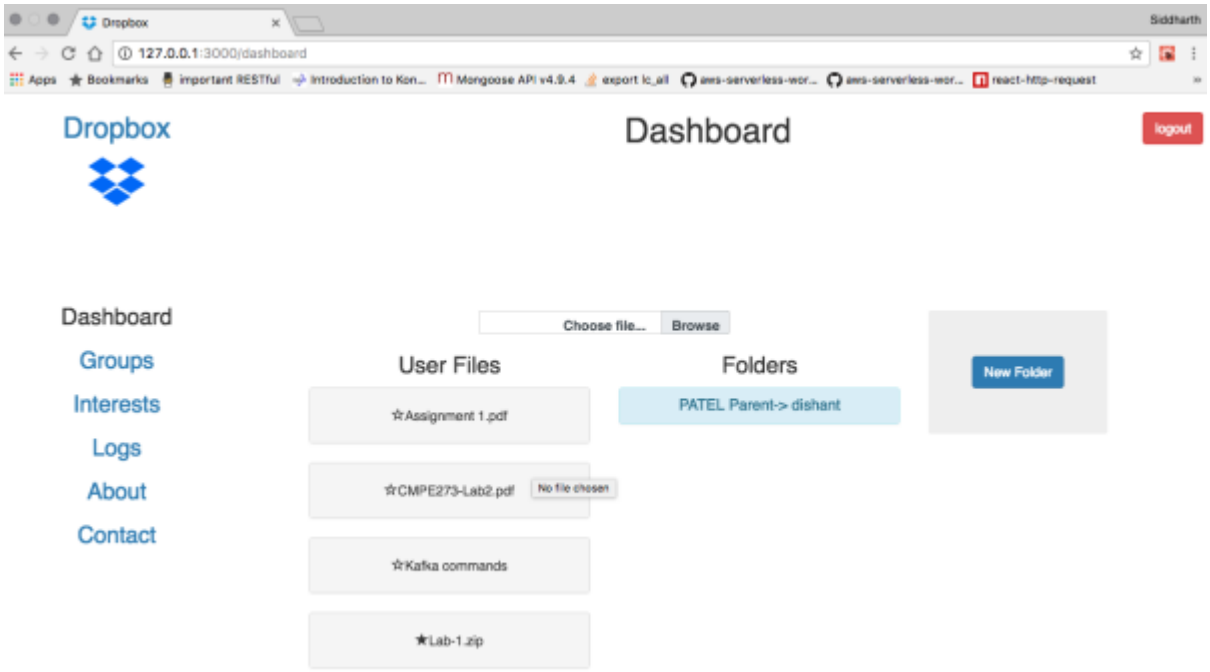




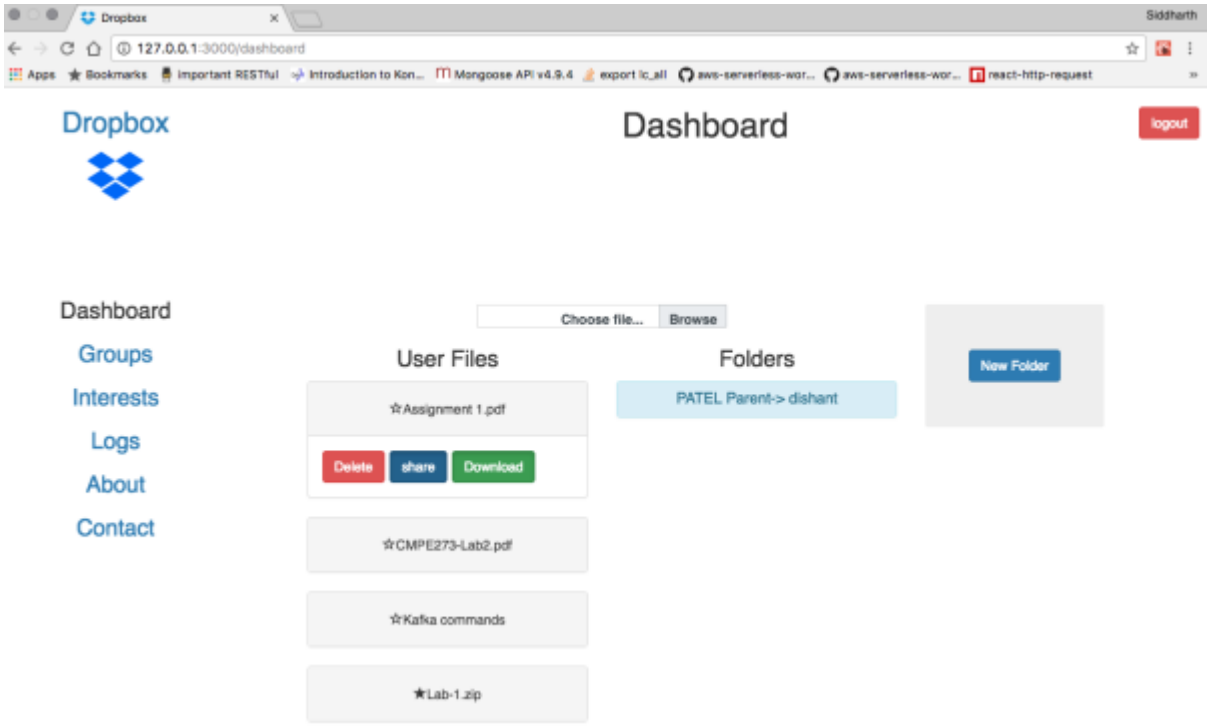
2) Dashboard : This is the dashboard of the application where the user can create new folders, upload files and share files. And mark his files as Starred or not.



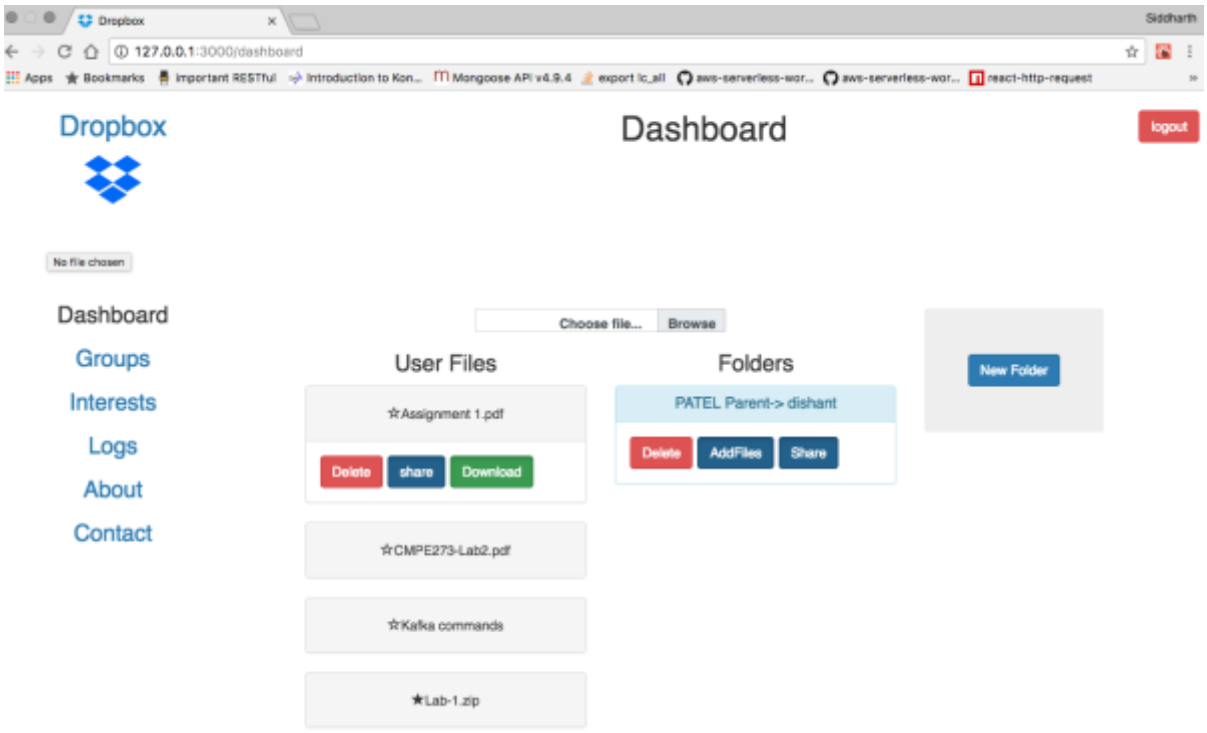
Starred Files-



File Share , Download , Delete Functionalities-

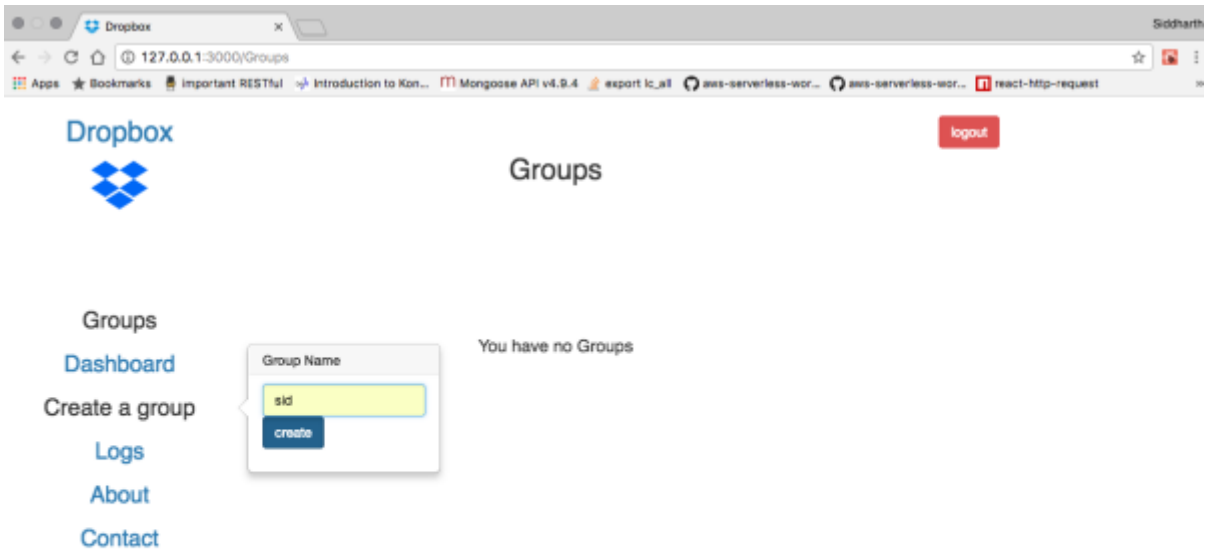


Folder Share, Create, Add Files and Delete Functionality-

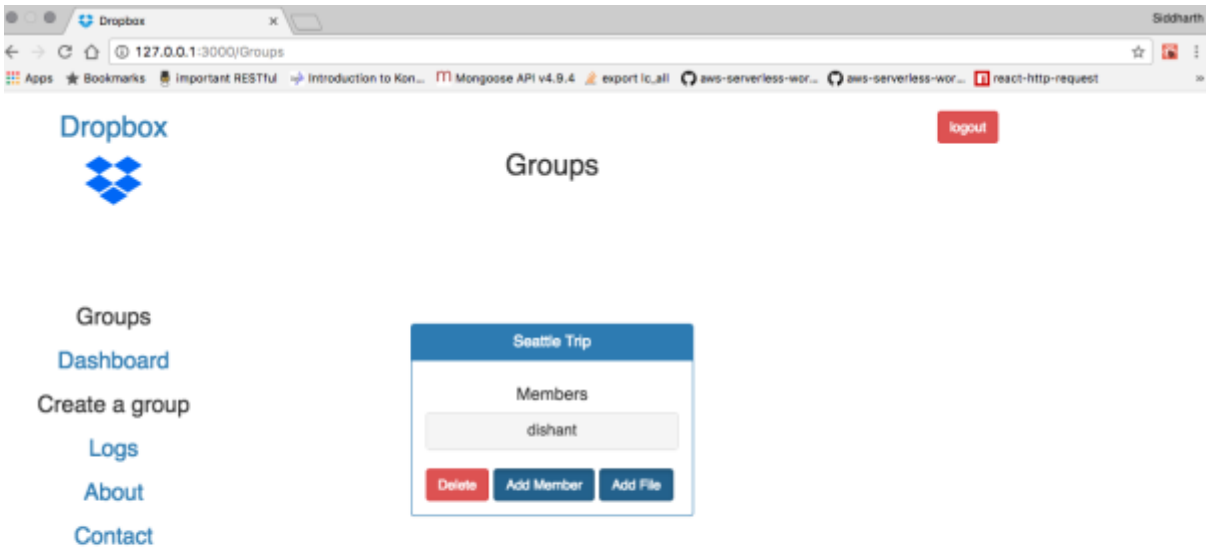


Groups Page :

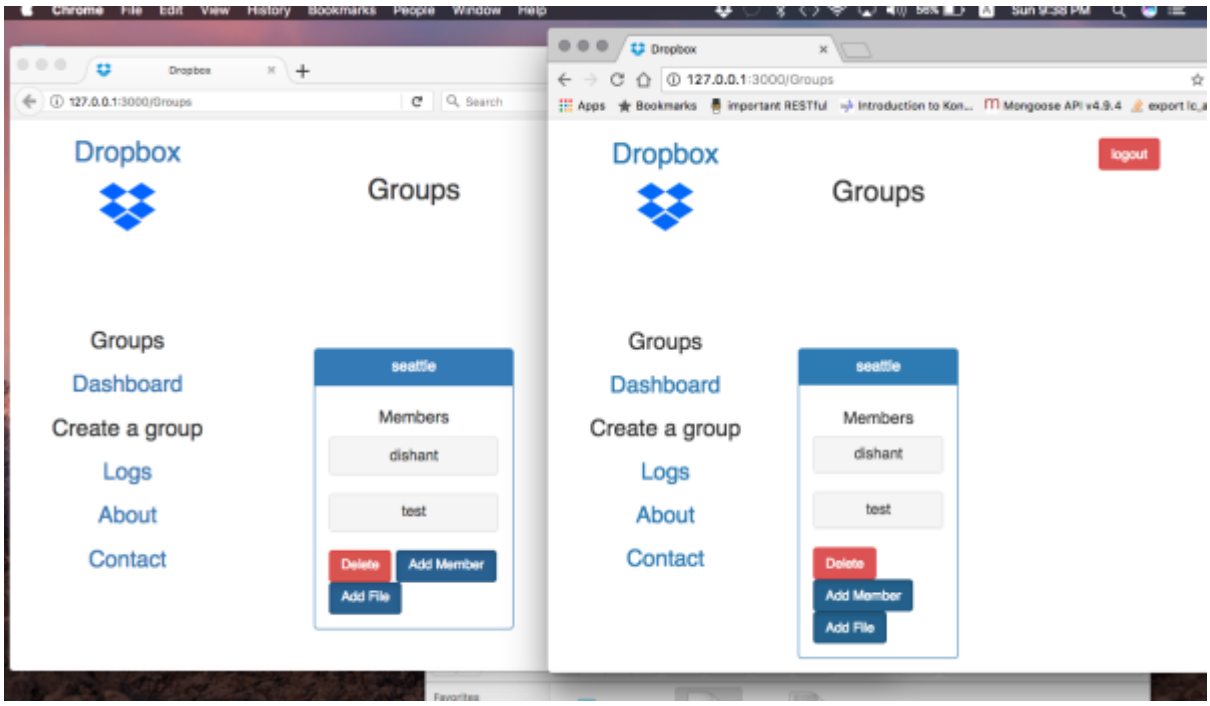
Creating a group (Owner is by default member of the Group) –



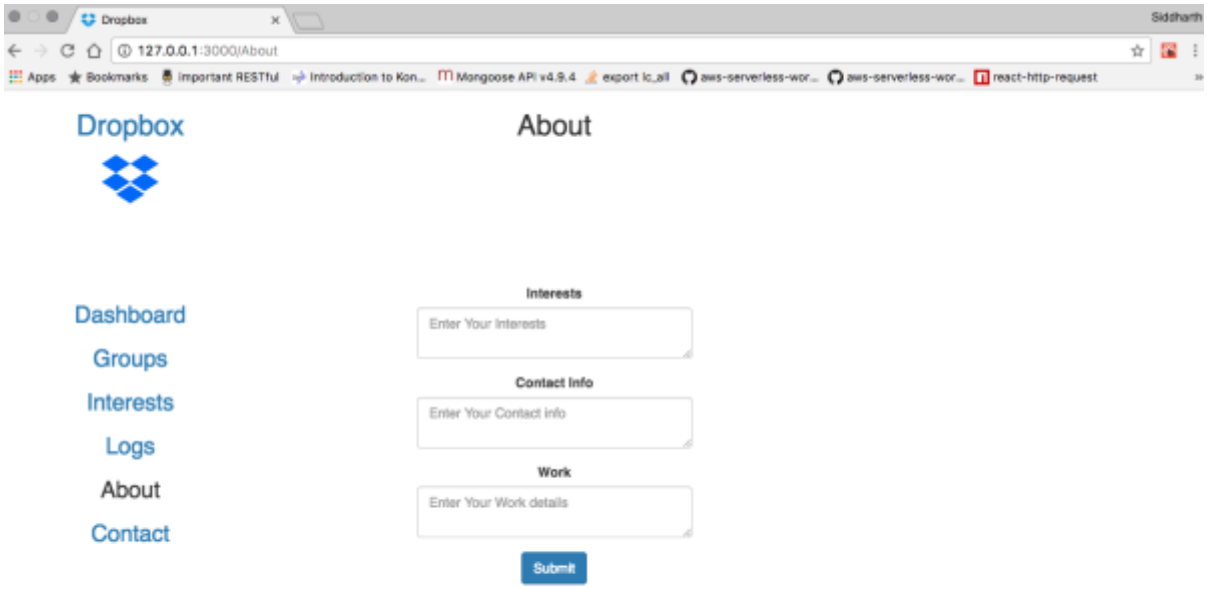
Group with owner as a default member –



Adding a new member and checking in that member's groups page-


Dropbox
Groups
Groups
Dashboard
Create a group
Logs
About
Contact
seattle
Members
dishant
test
Delete
Add Member
Add File
logout

About Page –

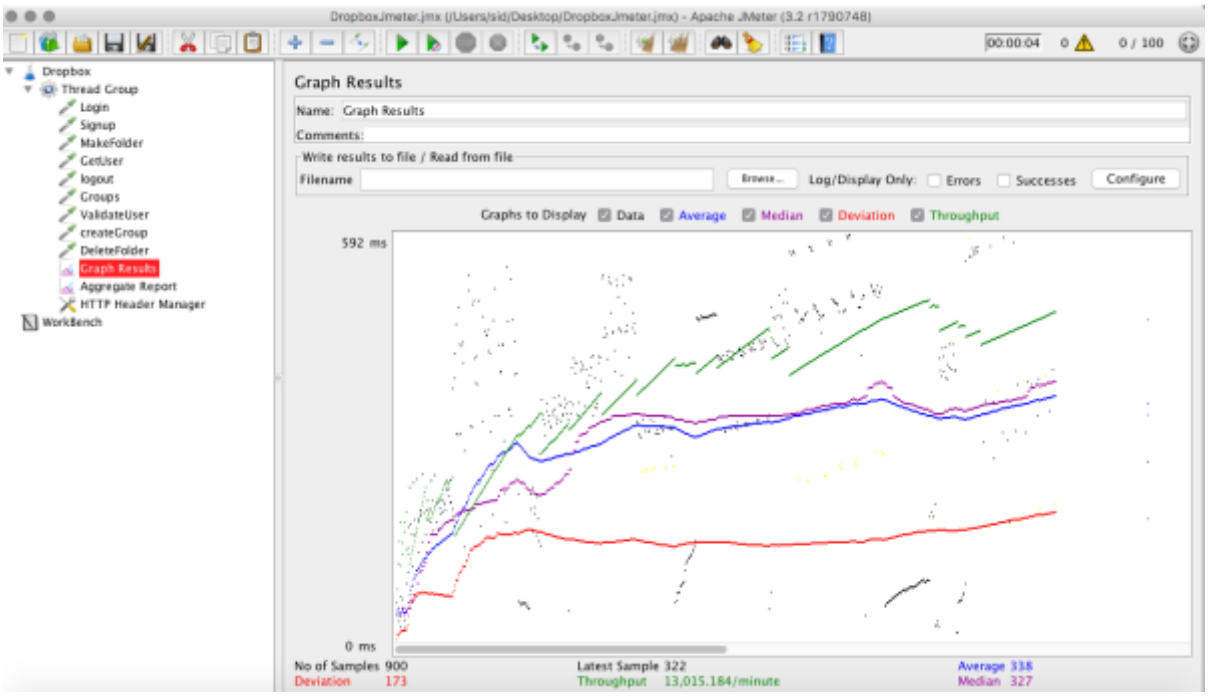

Dropbox
About
Dashboard
Groups
Interests
Logs
About
Contact
Interests
Enter Your Interests
Contact Info
Enter Your Contact info
Work
Enter Your Work details
Submit

Performance :

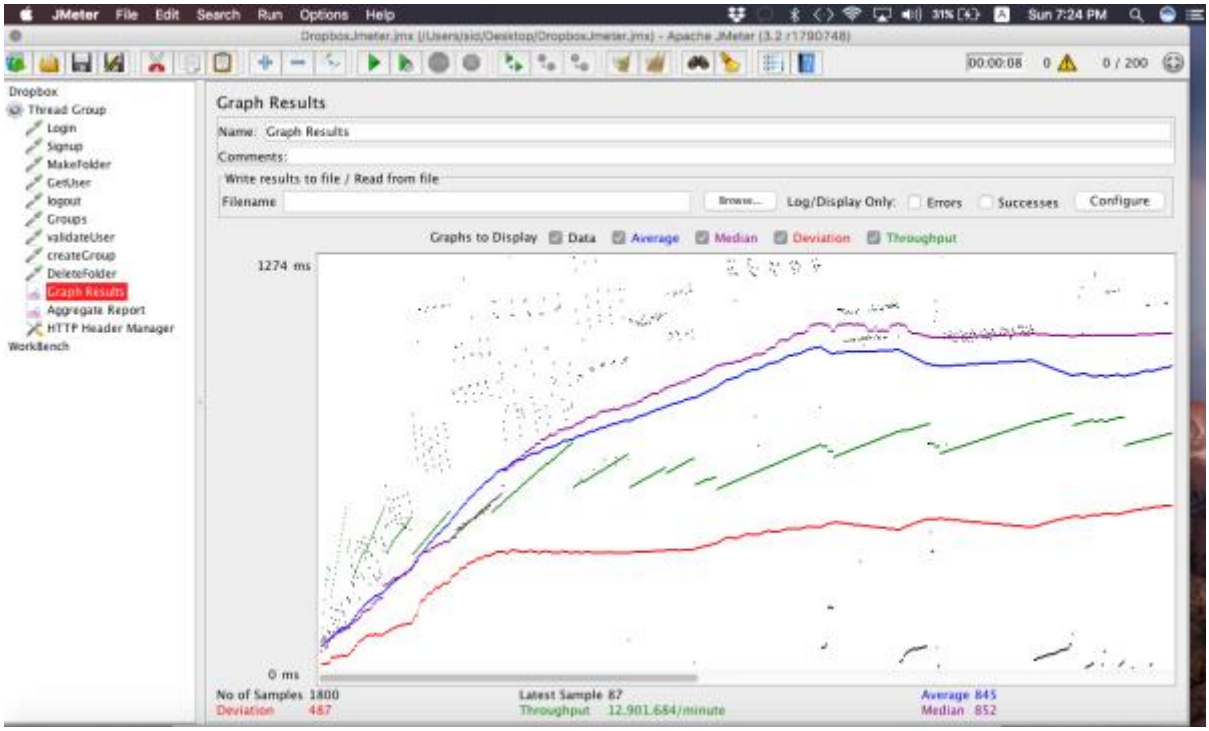
The performance of the system has been measured using Apache Jmeter.

1) For 100 concurrent users –

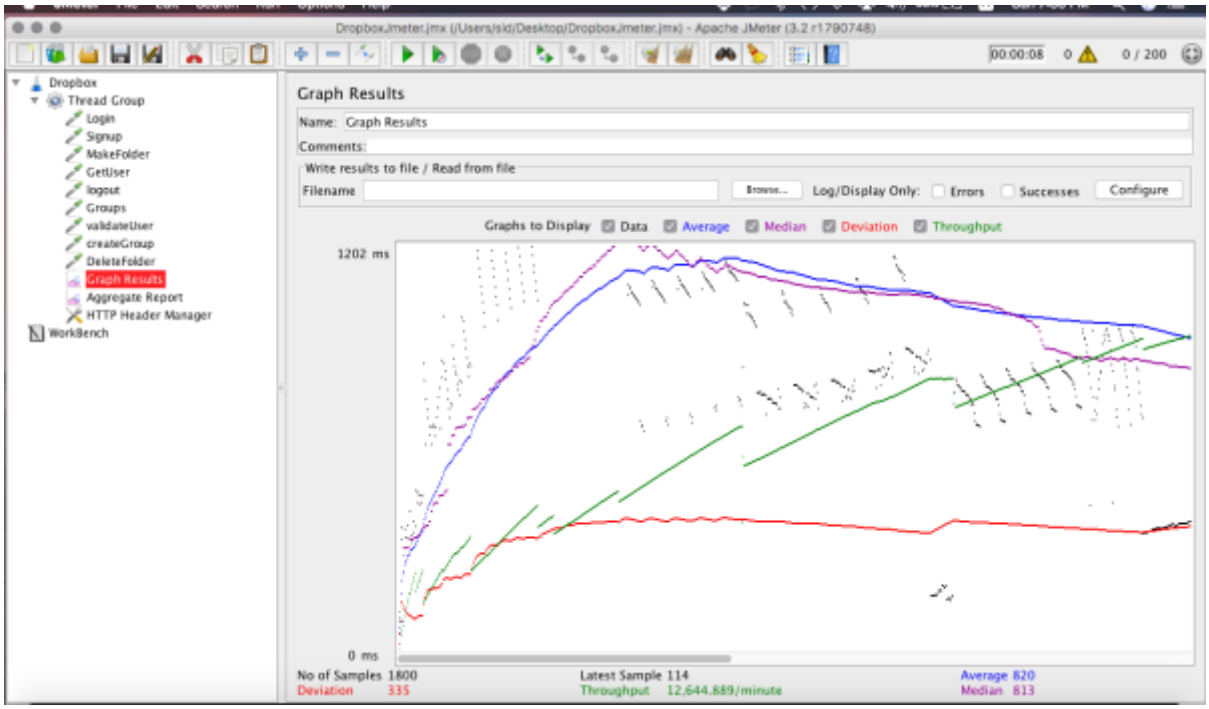
- Without connection Pooling –



- With Mongodb enabled Pooling :

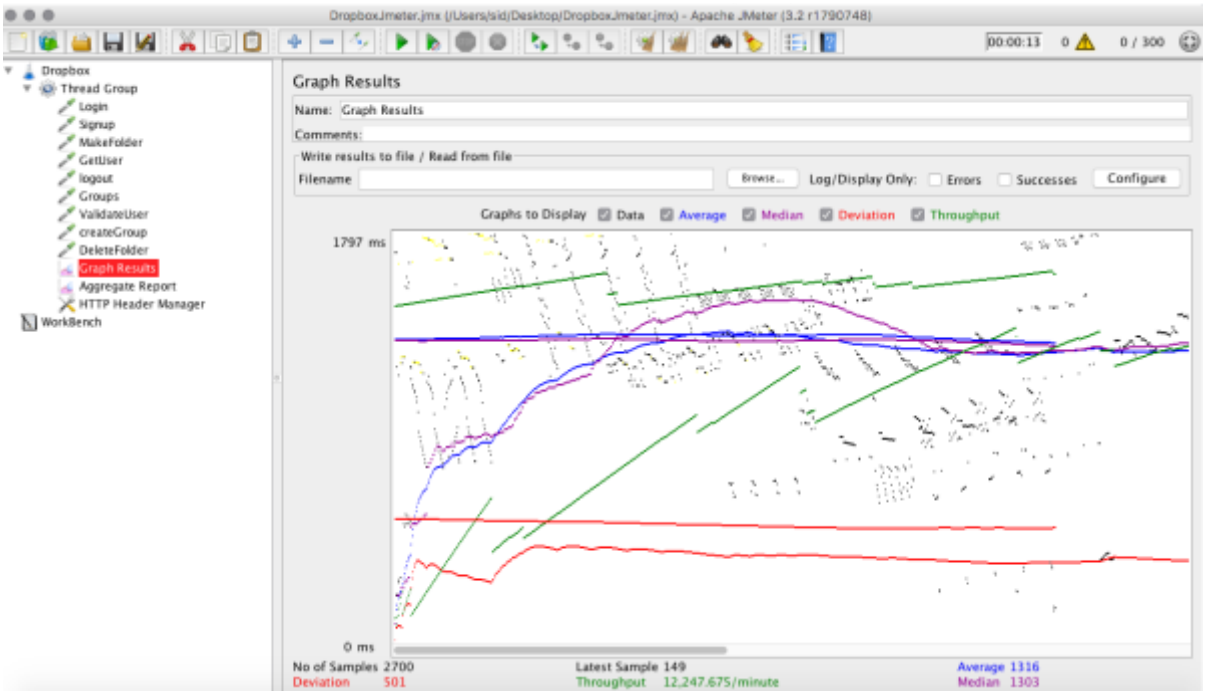


- With User defined Connection Pooling :

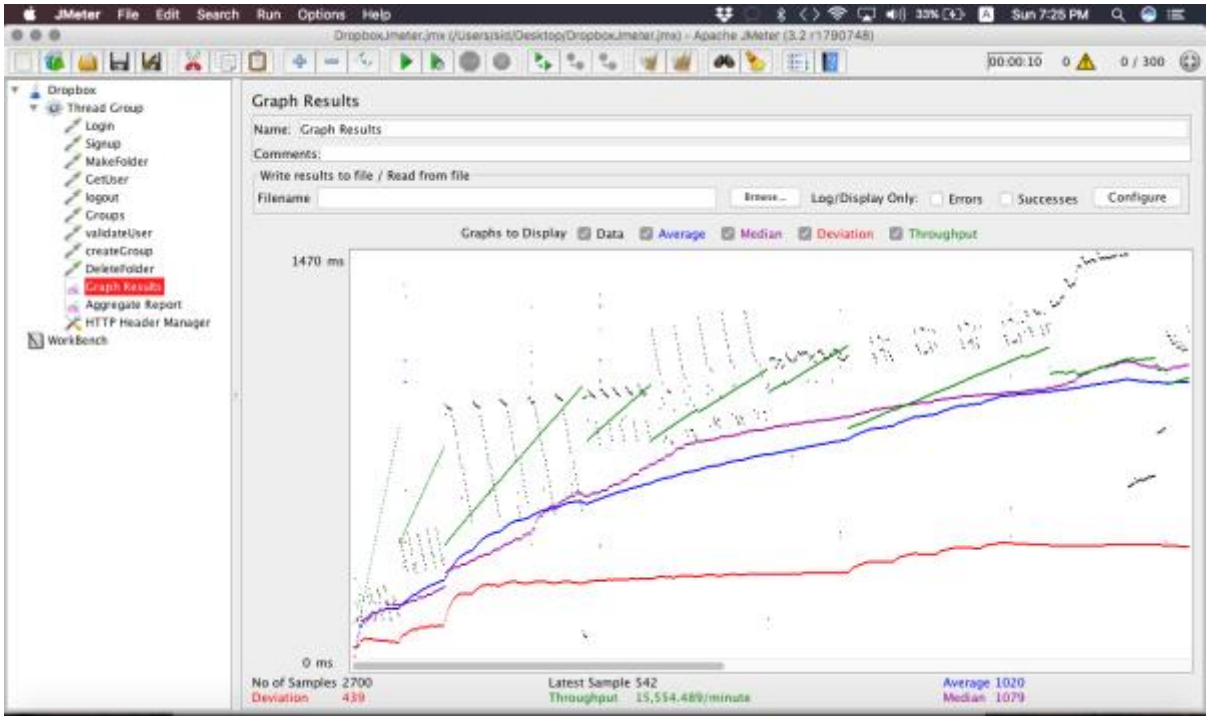


2) For 200 Concurrent Users :

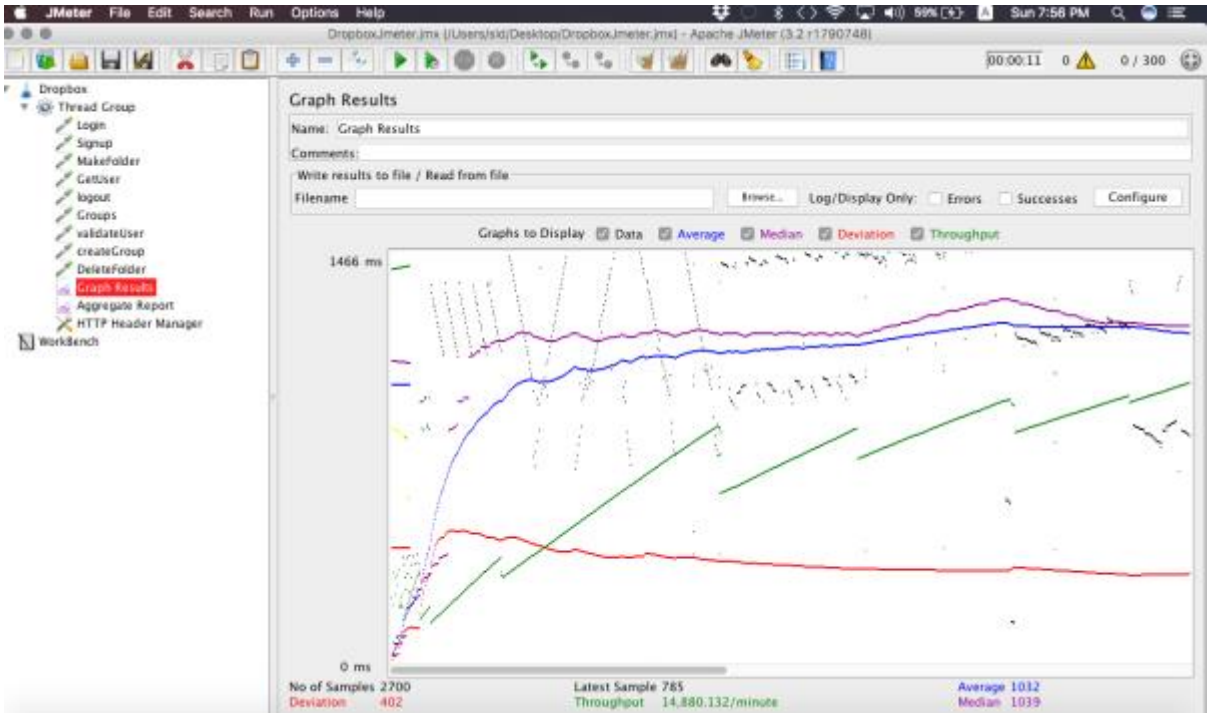
- Without Connection Pooling



- With mongodb provided connection pooling :

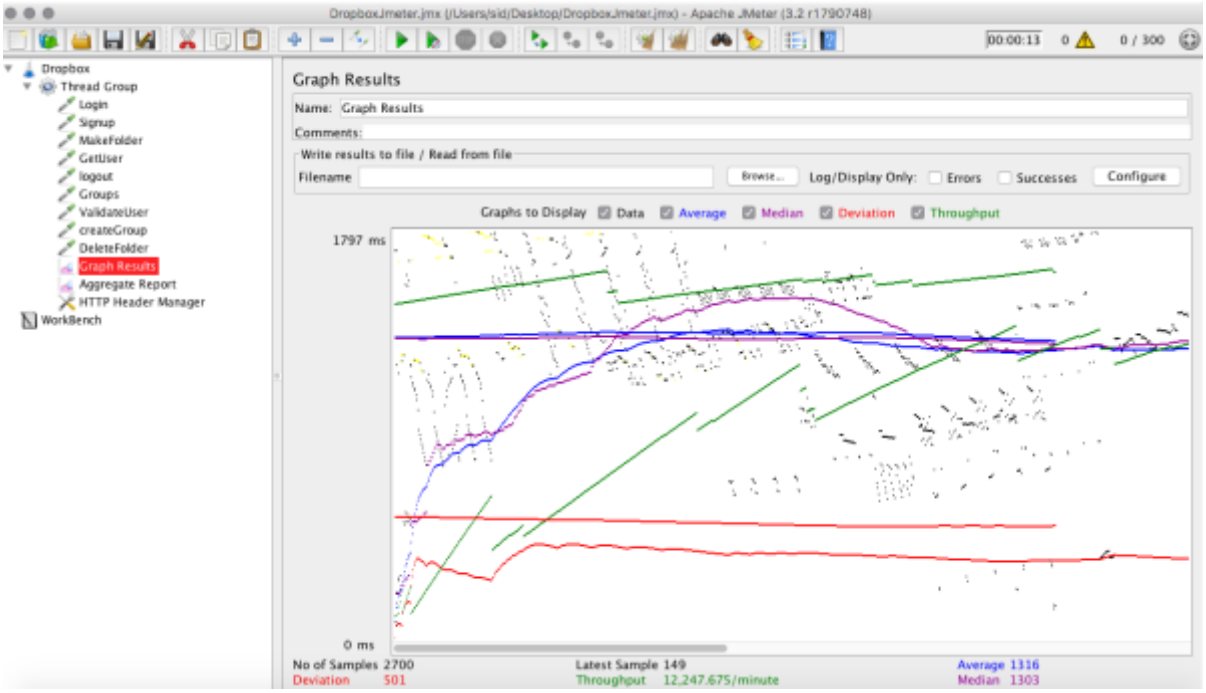


- With User Defined connection Pooling :

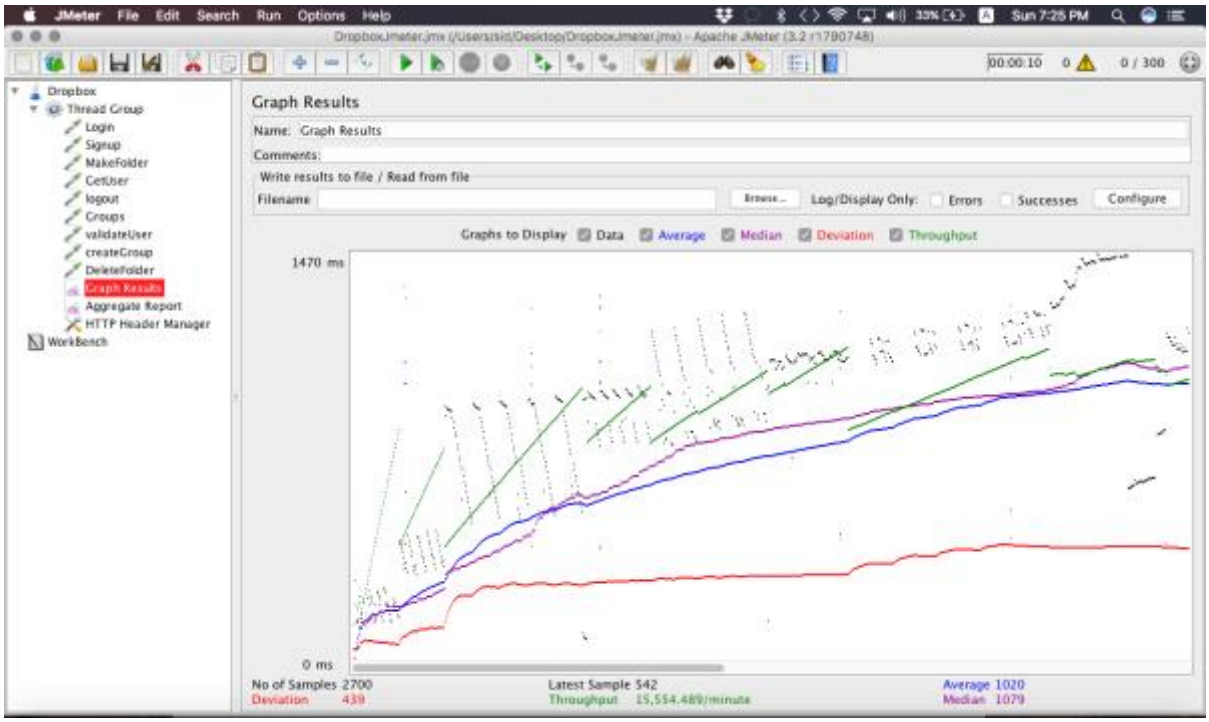


3) For 300 concurrent users :

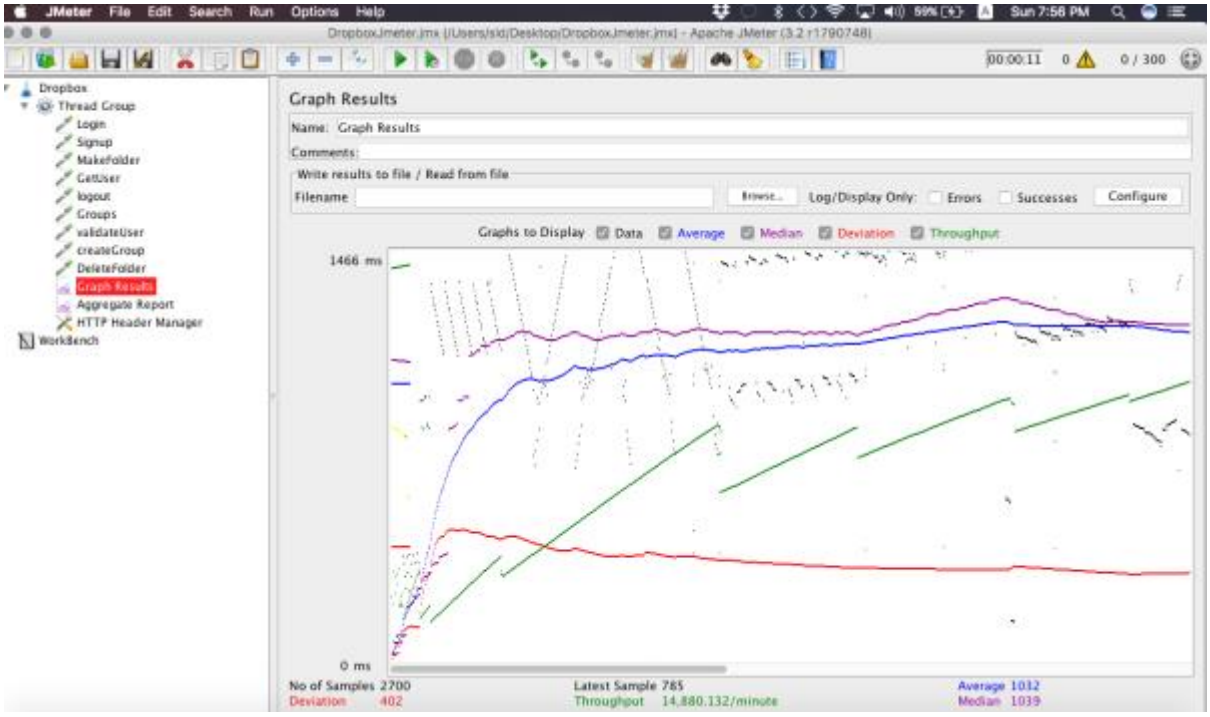
- Without connection pooling –



- With Mongodb provided connection pooling :

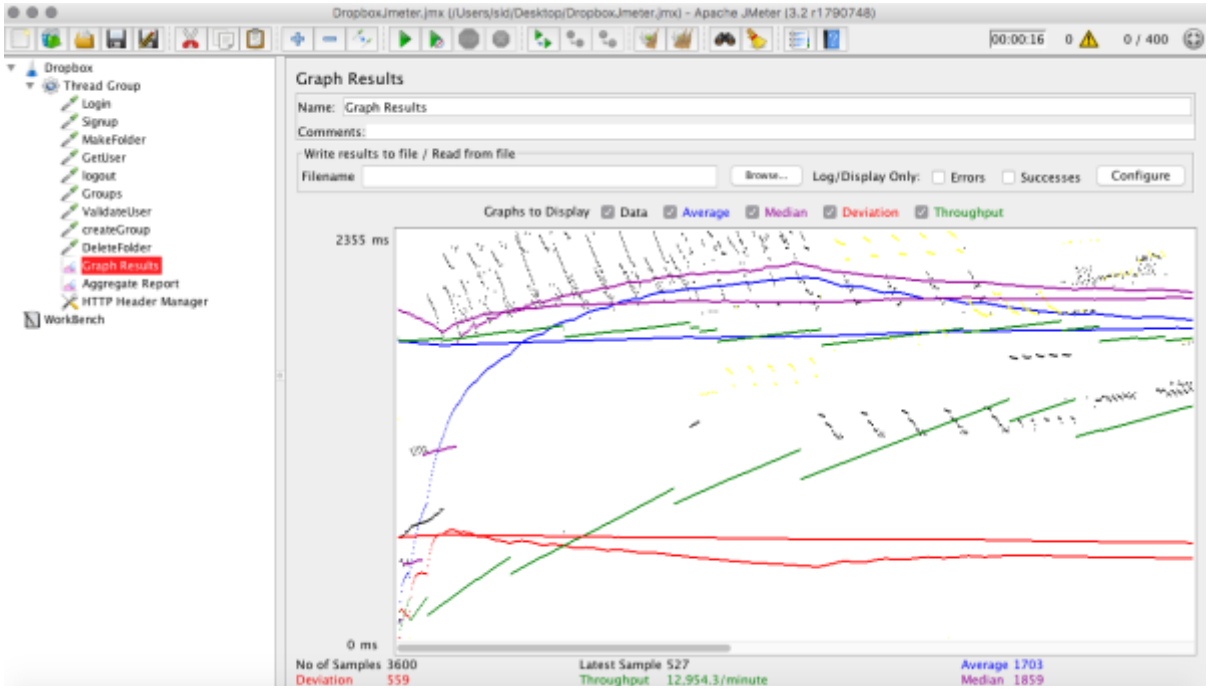


- With User defined connection pooling :

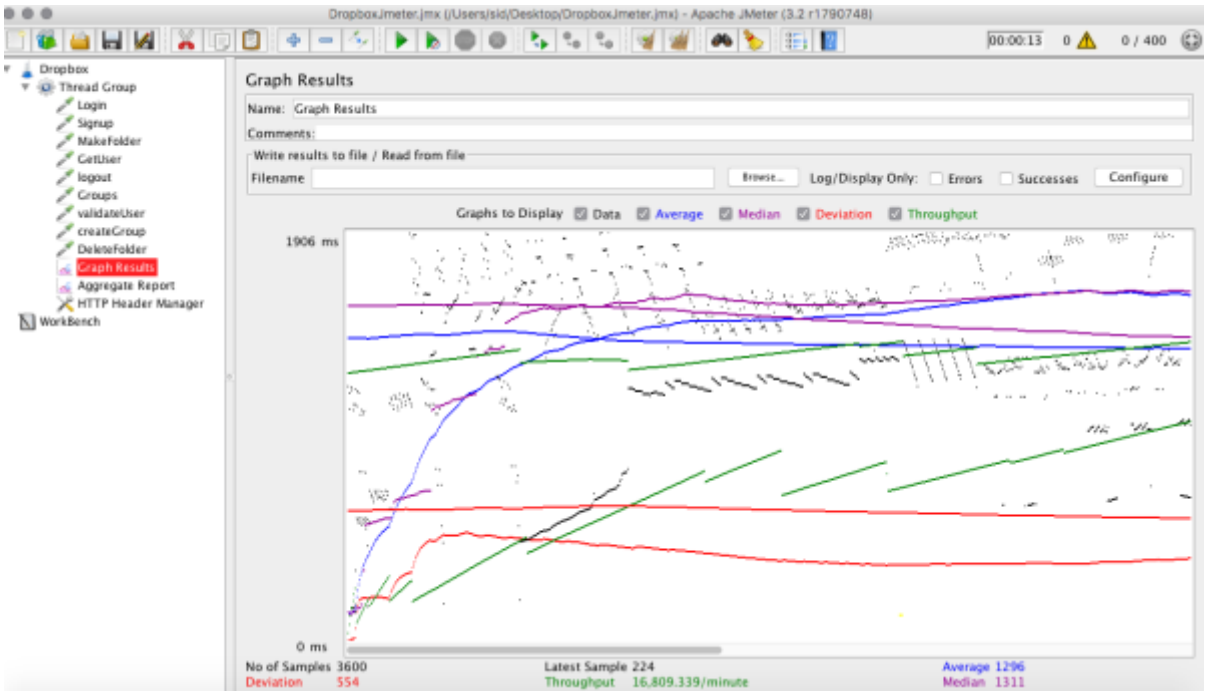


4) For 400 Concurrent Users :

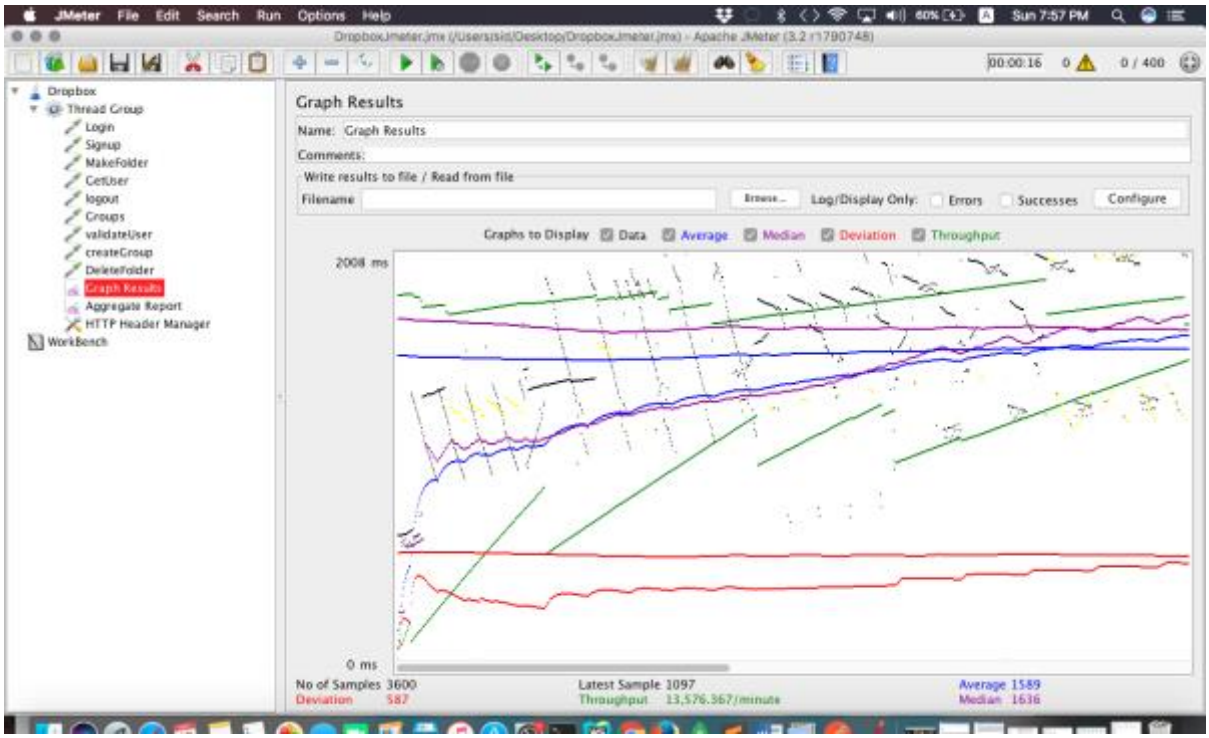
- Without connection Pooling –



- With mongodb provided connection pooling :

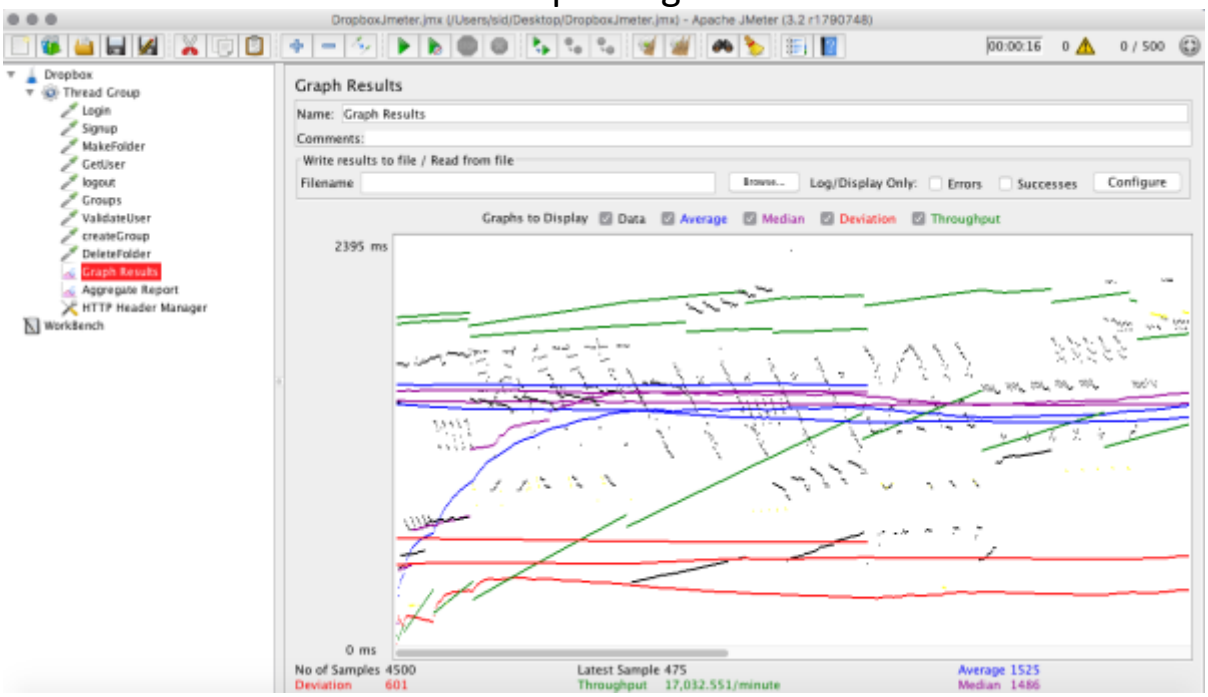


- With User defined connection pooling –

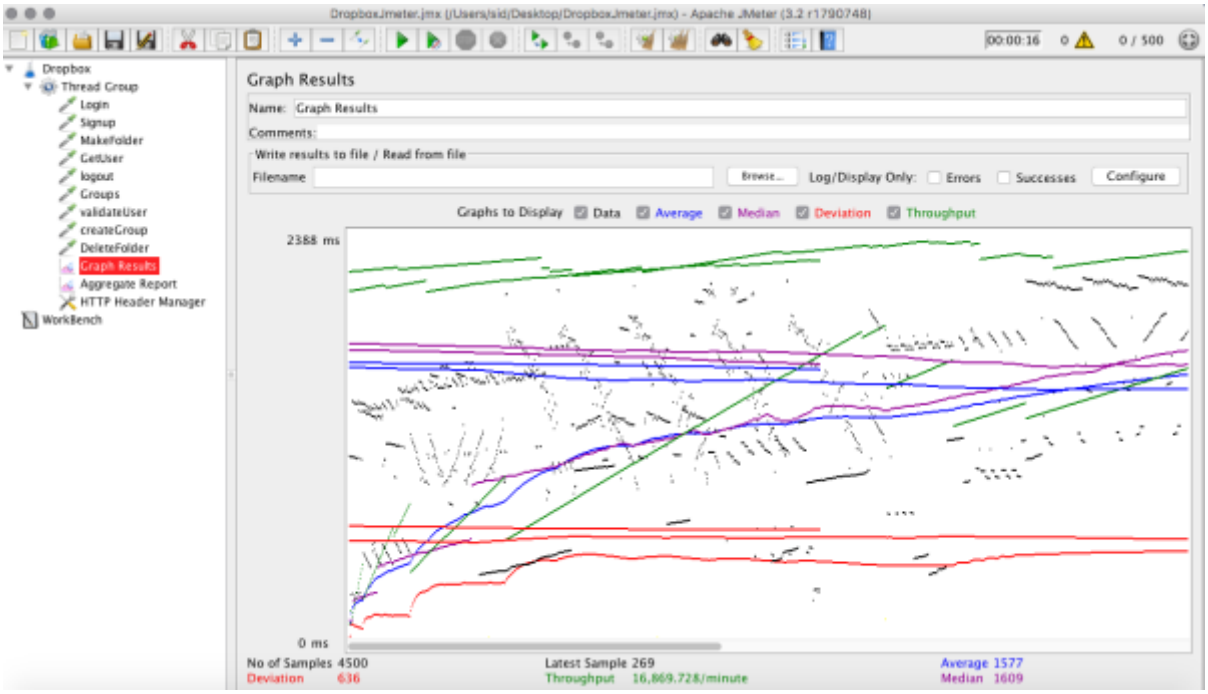


5) For 500 concurrent Users :

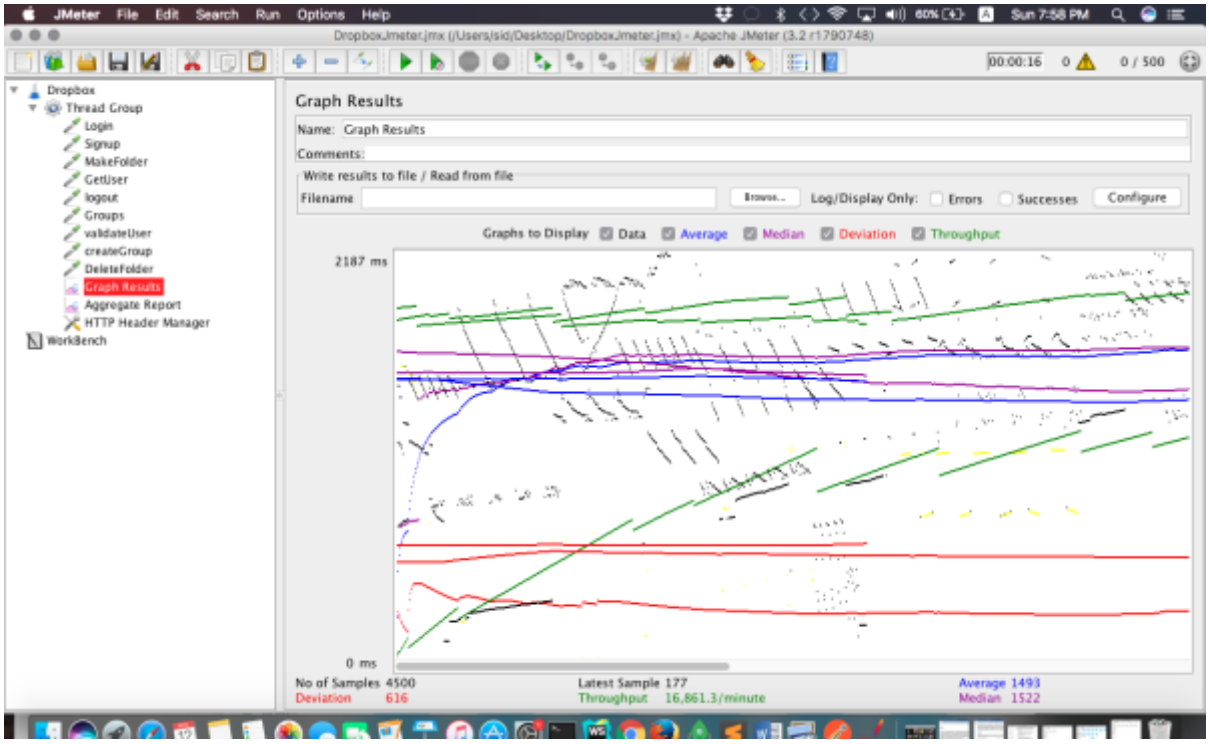
- Without connection pooling –



- With mongodb provided connection pooling –



- With User defined connection pooling :



From the above tests we come to a conclusion that the connection pooling significantly increases the performance of the system and increases the through put of the application , it takes less time to process the request using the connection pooling because a majority of time is spent in establishing the connection but IN MONGODB this is inbuilt so , the average time on all the three cases for 100 concurrent users is not much different but as the number of concurrent users increases the connection pooling with higher number of open connections increases the performance.

Mocha Test Case:

Questions :

1. Compare passport authentication process with the authentication process used in Lab1?

Answer : In lab 1 we have used the user defined authentication system where we manually check from the database whether the username and password matches or not, this authentication process takes more time compared to the more efficient passport authentication system.

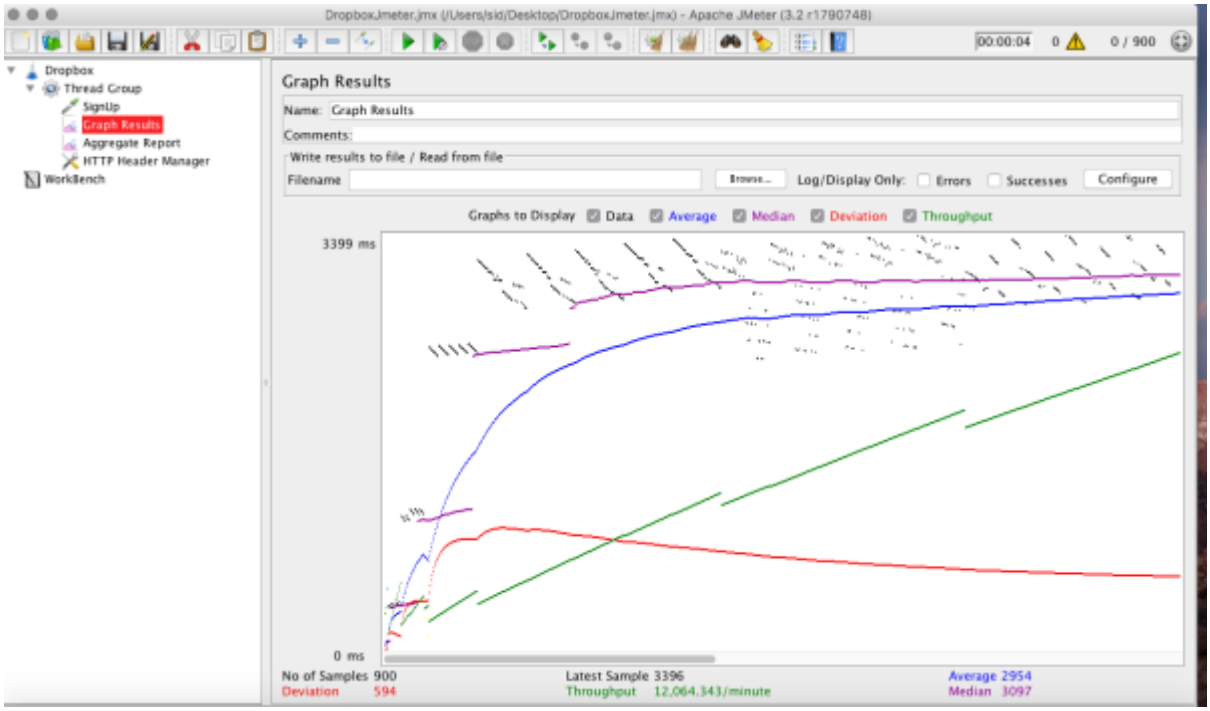
In Lab 2 we have used the passport authentication method where we use the local strategy method to validate the user from the database but the major advantage of using the passport authentication is if we want to enable the user to login using the google or facebook's credentials they can, using the passport authentication , all that needs

to be done is to implement the authentication provided by google and or facebook , and their API , so just an API call would solve the problem and user would be able to login using their google or facebook's credentials.

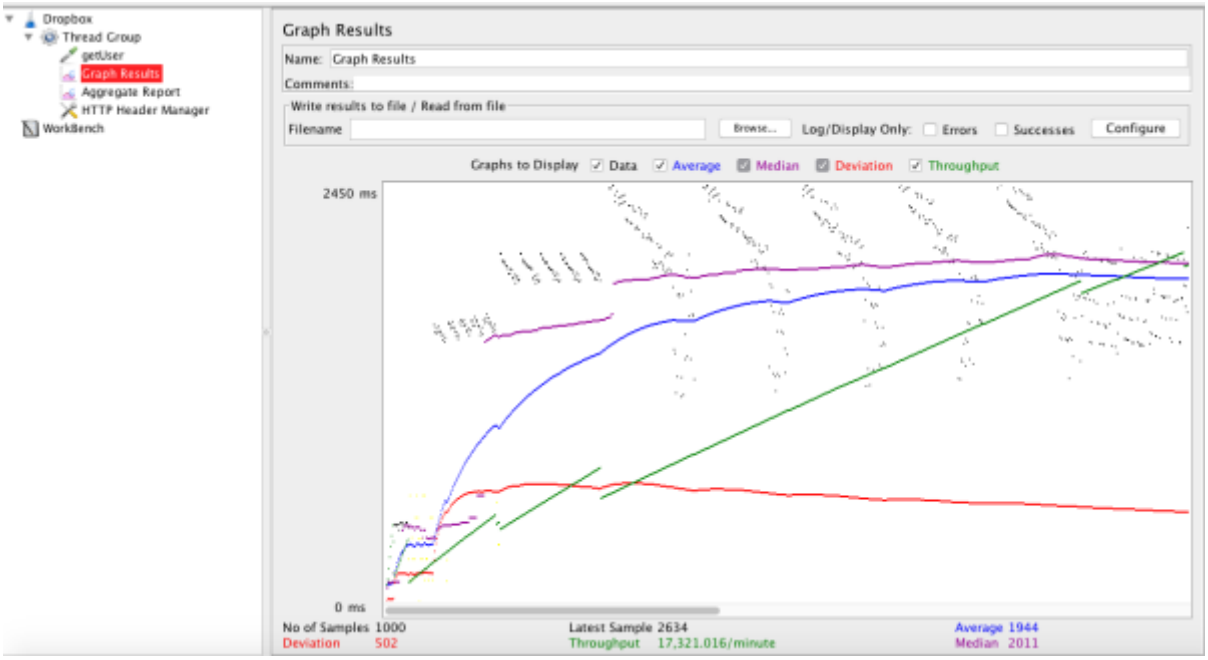
2. Compare performance with and without Kafka. Explain in detail the reason for difference in performance.

Answer :

- Performance without Kafka –



- Performance with Kafka –



As it is seen from the above images and seeing the average time for processing the API call it is obvious that the performance with kafka is significantly higher , as the number concurrent requests increases the difference becomes more and more prominent and the reason for that is that whenever an API call is done the majority of time is spent in serving the request is wasted while fetching the data from the database , so using kafka what we do is we reduce that time , the node server's (kafka frontend) job is to open up the request parse it and send only the revelent data to the backend server so the backend server does not waste any time unpacking the request and can directly work on getting the response from the database . and also another major advantage of using kafka is that the application can become distributed , we can replicate the backend code into more number of servers and can make them into consumers and increase the performance of the system.

3. If given an option to implement MySQL and MongoDB both in your application, specify which data of the applications will you store in MongoDB and MySQL respectively

Answer : MySQL stores data in tables and uses structured query language (SQL) for database access, while MongoDB stores data in JSON-like documents that can vary in structure.
If given an option, we will store user data and user group details in MySQL since MySQL offers high performance in data access and user schema will not require frequent changes.

MongoDb will be used to store file data due to its flexible data model and its NoSQL format which allows us to change database schema with change in business requirements.
MongoDB can also be scaled within and across multiple distributed data centers, providing new levels of availability and scalability.

Also the files can be large so storing large size files in relational database can be more expensive performance wise.
Also the User activity data and the logs should be stored inside the mongodb because of its large size and the unstructured format.
And the need to access the user activity logs is also less frequent.