

Improved Friend Suggestions based

on Location Clustering

**CSE-612 Cloud Computing**

**Spring-2015**

**Kavish Pahade**

**Arun Balaji**

**Instructor**- **Dr. Jian Tang**

**Background and Motivation:**

Social networking has become most popular activity in today’s internet world, with billions of people across the world are using it to find new friends and people who have similar interest. The friend suggestion mechanism used by most of these social networking sites only look for the number of mutual friends one has with the other, who may be far away but they do not consider if they are located in the same area or in close proximity to each other. This report discusses the implementation of a K-Means clustering and Map-Reduce application to search and suggest new friends to a person in a social group based on their geographical locations.

K-means clustering is a data mining/machine learning algorithm used to cluster data sets or observations into groups of related data sets or observations without prior knowledge of those relationships. The algorithm starts by choosing an arbitrary k points as an initial number of cluster centers. Each element in the dataset is assigned to a cluster by calculating the distance between each point and each cluster. Each cluster is recomputed again and again and the average of these elements is taken as new centers.

Map-Reduce is a programming model that is used to analyze big datasets and draw insights from them. This programming model has two phases, the Map phase and the Reduce phase. The Map phase produces a set of key-value pair as intermediate output from the input datasets and feeds it to the Reducer. For each key, a reducer function is called that produces the final output.

1. K-Means clustering and map reduce is a new programming model that is used to efficiently cluster and process Big datasets and create new readings from it.
2. Our project is to suggest new friend from group of people to a person based on the number of common friends between them also based on their location.
3. The performance comparison of the application can be done by running the application on Hadoop which runs on a single node on a local machine and running on Hadoop which runs on multiple nodes on Windows’ Azure.

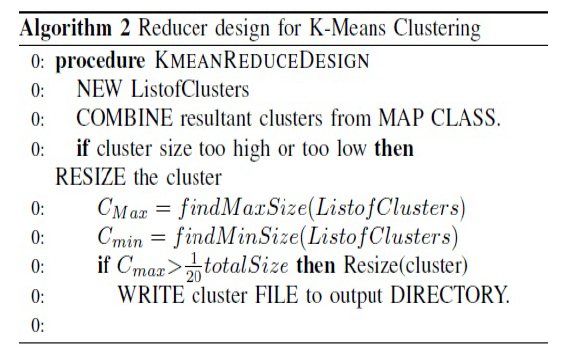
Problem definition:

The problem can be divided into three stages.

1. We want to form clusters based upon the latitude-longitude coordinates from users’ recent check-in data. These clusters can be formed by choosing a k initial latitude-longitude coordinates and comparing each data from the data set with each of these coordinates.
2. Once the clusters are formed, we identify in which cluster the specified user is located and restrict our search to those friends within that cluster.
3. We process the entire friend list, identify the specified user and the list his/her friends and search through the cluster to find those friends who are also in that cluster.
4. We extract people who are in that cluster but are not friends with that user and find the number of mutual between them and the user.
5. Once the mutual friends count is found we suggest top 5 people based on who has the most number of mutual friends.

**Related works and differences:**

The implementation of K-Means clustering and Map-Reduce is done by referring to the following algorithm (1).



**Differences**:

The Mapper part of the above algorithm suggests to use a pre-defined initial set of centroid points to start the clustering process but since our large dataset consists of geographical coordinates we chose to randomize the centroid values so that the initial clusters can be distributed across the entire datasets.

The Reducer part of the above algorithm calculates the new centroids by finding the cluster which has the maximum size and minimum size, which will work efficiently if the elements within those clusters can be plotted on a XY plane (Euclidean distance) but our elements within the clusters are distributed across a sphere (Globe) so chose to find new clusters by finding the geographical distance (great circle formula) between each elements with that of the centroid. If the distance is too big then they are compared with other cluster’s centroids.

The great-circle distance is the shortest [distance](http://en.wikipedia.org/wiki/Distance) between two [points](http://en.wikipedia.org/wiki/Point_(geometry)) on the surface of a [sphere](http://en.wikipedia.org/wiki/Sphere), measured along the surface of the sphere (as opposed to a straight line through the sphere's interior). The distance between two points in [Euclidean space](http://en.wikipedia.org/wiki/Euclidean_space) is the length of a straight line between them, but on the sphere there are no straight lines. The approximation of the Great Circle Distance formula is used while calculating the distances between two geolocations.

Great Circle Distance Formula:

3963.0 \* arcos [sin(lat1) \*  sin(lat2) + cos(lat1) \* cos(lat2) \* cos(lon2 - lon1)]

Where 3963.0 represents normal miles

Coming to the finding mutual friends in the cluster, we have not referred to any existing friend recommending algorithms. Our algorithm is described in great detail later in the report.

Methodology:

This algorithm takes two dataset files as inputs.

Check-in List:

**User-ID date-time of check-in latitude longitude location-ID**

0 2010-10-17T01:48:53Z 39.747652 -104.99251 88c46bf20db2958

1 2010-03-09T16:34:49Z 37.580304 -122.343679 8fde23d6245c11d

The file contains the list of all the users’ recent check-in details such as latitude-longitude coordinates, date-time of the check-in.

Friend List:

|  |  |
| --- | --- |
| **User-ID** | **Friend-ID** |
| 0 | 1 |
| 0 | 2 |
| 1 | 0,3 |

The file contains a list of all the users and their friends. In the above sample User-ID:0 is friends with User-ID:1, 2 and User-ID:1 is friends with User-ID:0, 3

The check-in list file contains an User-ID, latitude, longitude and check-in time-date and a set of initial randomly chosen centroids are sent to the K-means clustering and Map-Reduce code. These centroids will form the initial clusters. K-Means clustering algorithm has a limitation such that it can form clusters by calculating Euclidian distance (straight line) between two 1-Dimensional points. Since our large dataset consists of geographical locations (latitude-longitude) which are limited -180 to +180 we use another formula called “Great Circle Distance formula” to calculate the geographical distance between two people.

In the Mapper part, the clusters are created by calculating the distance between each latitude-longitude coordinates in the dataset to each centroids and are assigned to the nearest centroids. Now each cluster contains a single centroid value and list of users’ check-in details. This result is taken as a Key-Value pair where Key is the centroid value and Value is the list of data sets.

In the Reducer phase a new centroid for that cluster is recalculated based on the latitude-longitude coordinates present in that cluster. This is important to prevent the cluster from having very small datasets or very large datasets.

Here is the sample result of the Map-Reduce part contains a list of clusters with their assigned datasets.

**Initial random Centroid list:**

|  |  |  |
| --- | --- | --- |
| **Latitude** | | **longitude** |
| 9.3148086605034 | | -70.32531196405387 |
| 52.12287303644333 | | -85.63527822667935 |
| 46.172498333374904 | | -130.75809892351745 |
| -49.42672774850832 | | 58.169838766029486 |
| -41.13987278127408 | | -154.92210519208723 |
| **Cluster 1: (key)** | **(Values)** | | |
| 46.172498333374904  -130.75809892351745 | 2008-11-30T22:30:12Z 63.484654518 -75.43634777118933 ::30 | | |
| 2008-11-16T21:39:34Z 39.73333 -104.9911021148711 ::13 | | |
| 2008-06-14T22:34:13Z 39.731044 -104.9861591248963 ::29 | | |

|  |  |
| --- | --- |
| **Cluster 2: (key)** | **(Values)** |
| 61.61215838922236 -75.73399166319334 | 2009-05-14T17:12:39Z 39.765345 -104.99482::0 |
| 2009-06-23T17:36:01Z 39.765345 -104.99482::12 |

These centroids and clusters are re-sent for the next iteration for Map-Reduce phase. The final sets of clusters are formed once the value of previous set of centroids does not deviate much with the newly formed centroids. For this purpose we calculate the distance between the recently generated centroids and the previous ones which are stored in the output file generated by the previous iteration. Again these clusters are the geolocations with latitudes and longitudes, so we need to use the great circle distance formula described earlier. Now all the centroids are sorted and the convergence distance between is calculated. If this convergence distance is below a certain limit, the iterative map reduce passes are stopped. The final clusters will contain all those users who have checked-in closer to the others in the same cluster.

Before implementing the second Map-Reduce code the distance is calculated between the specified user’s latitude-longitude coordinate and each cluster and the cluster which gives the minimum distance is chosen. **Once that cluster is identified then all further operations are limited to that cluster alone.**

During the first iteration, in the Mapper phase each line from the Friend list file (edges) containing the User-ID and Friend-ID is compared with the user-IDs in the clusters. If both the User-ID and the Friend-ID are present inside the cluster then that edge is taken. Once all these edges are obtained they are passed into the Reducer Phase. The Reducer identifies the friends list for each User-ID. The input and output during the 1st iteration of Map-Reduce is shown below.

Input for the Mapper phase.

|  |  |
| --- | --- |
| **User-ID** | **Friend-ID** |
| 0 | 2 |
| 1 | 3 |
| 1 | 4 |
| 4 | 10 |

Output from the Reducer phase.

|  |  |
| --- | --- |
| **User-ID** | **Friend-ID** |
| 1 | 5,3,4 |
| 10 | 4, |
| 3 | 1,9,4,8 |
| 4 | 10,3,5,9,1,8 |
| 5 | 18,4 |

During the 2nd iteration we emit the Cartesian product by taking the <User-ID, Friend-IDi> and < Friend-IDi , Friend-IDj > where i, j is one of the friends among that user’s friend list. This elements in the pair are also looked upon to find if there are element pairs in the reverse order such as <Friend-Idi, User-ID> and if so then the second pair is ignored.

1. In the Mapper phase considering the above table, 1🡪5, 3, 4: pairs (1, 5), (1, 3), (1, 4) are given value of -1 since user 1 is already a friend with 5, 3 and 4.
2. Pairs (5, 3), (5, 4) and (3, 4) are given the value of 1 which means that user 5 and user 3 have one mutual friend who is user 1.
3. Similarly when considering the second row (10, 4) will have value of -1 which indicates that user 10 and user 1 are friends and neither user 10 nor user 4 have any mutual friends with user 1.
4. Similarly the other rows are populated. Here the key will be the pair and value will be either -1 or 1.

|  |  |
| --- | --- |
| **Mapper** | **Reducer** |
| (5, 3) = 1, (5, 4) = 1, (3, 4) = 1 | (5, 3) =2, (9, 1) = 2, (9, 8)=2, (1, 8) = 2 |
| (1, 9) = 1, (1, 8) = 1, (9, 4) = 1, (4, 8) = 1 |
| (1, 9)=1, (1, 8)=1, (9, 4) =1, (9,8)=1, (1, 8) = 1 |
| (10, 3)=1,(10,5)=1, (10, 9)=1, (10,1)=1,(10,8)=1,(3,9)=1, (3,8)=1,(5,9)=1,(5,8)=1, (9,1)=1, (9,8)=1, (1, 8) =1 |
| (18, 4) = 1 |

We consider the output from the reducer part which contains user 1 as one of the pairs. From the above output user 8 and user 9 will be recommend as friends for user 1.

Another approach is to generate a sparse matrix which is a linear way to find mutual friends which will drastically increase the time to find mutual friends considering the large datasets on friend list.

Sample sparse matrix to for the above table:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **USER-ID** | **1** | **3** | **4** | **5** | **8** | **9** | **10** | **18** |
| **1** | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| **3** | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| **4** | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| **5** | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| **8** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **9** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **10** | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| **18** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Thus implementing Map-Reduce approach on such a large dataset to find mutual friend can save a lot time and resource.

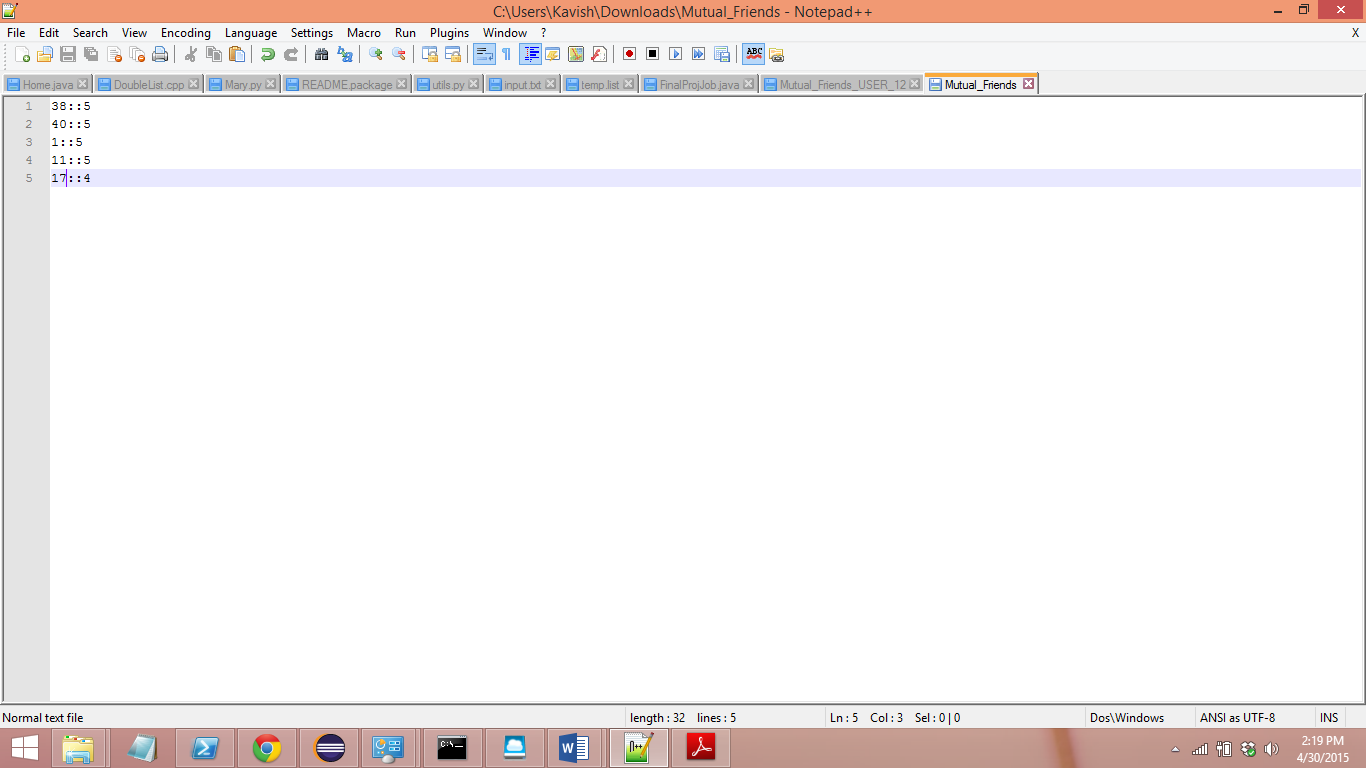
**Simulation and Analysis:**

We are using the dataset provided by Stanford Network Analysis Platform (SNAP). [Brightkite](http://www.brightkite.com/) was once a location-based social networking service provider where users shared their locations by checking-in. The friendship network was collected using their public API, and consists of 58,228 nodes and 214,078 edges. It has a total of 4,491,143 check-ins of these users over the period of Apr. 2008 - Oct. 2010.

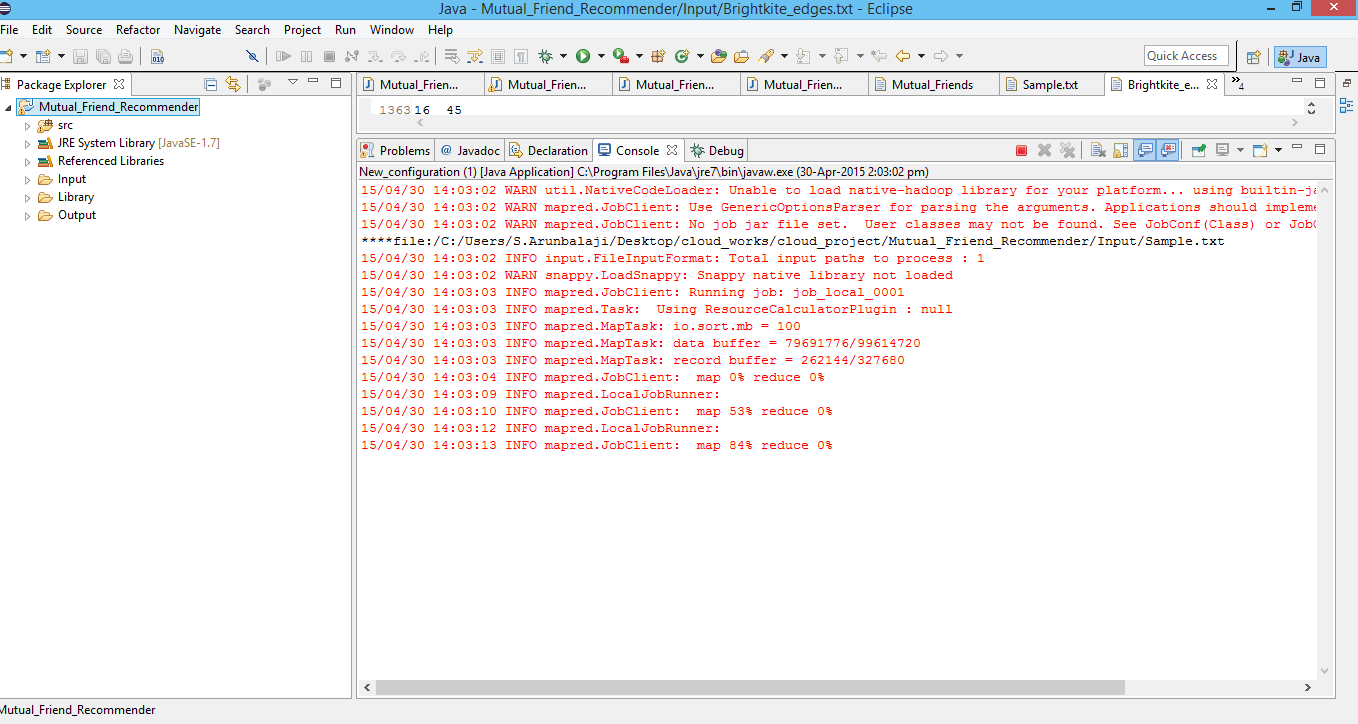
Below are the results for the map reduce application when run on different amounts of records,

1) Running for suggesting friends for userid 16 with total check-ins of 5K and edges of 1K

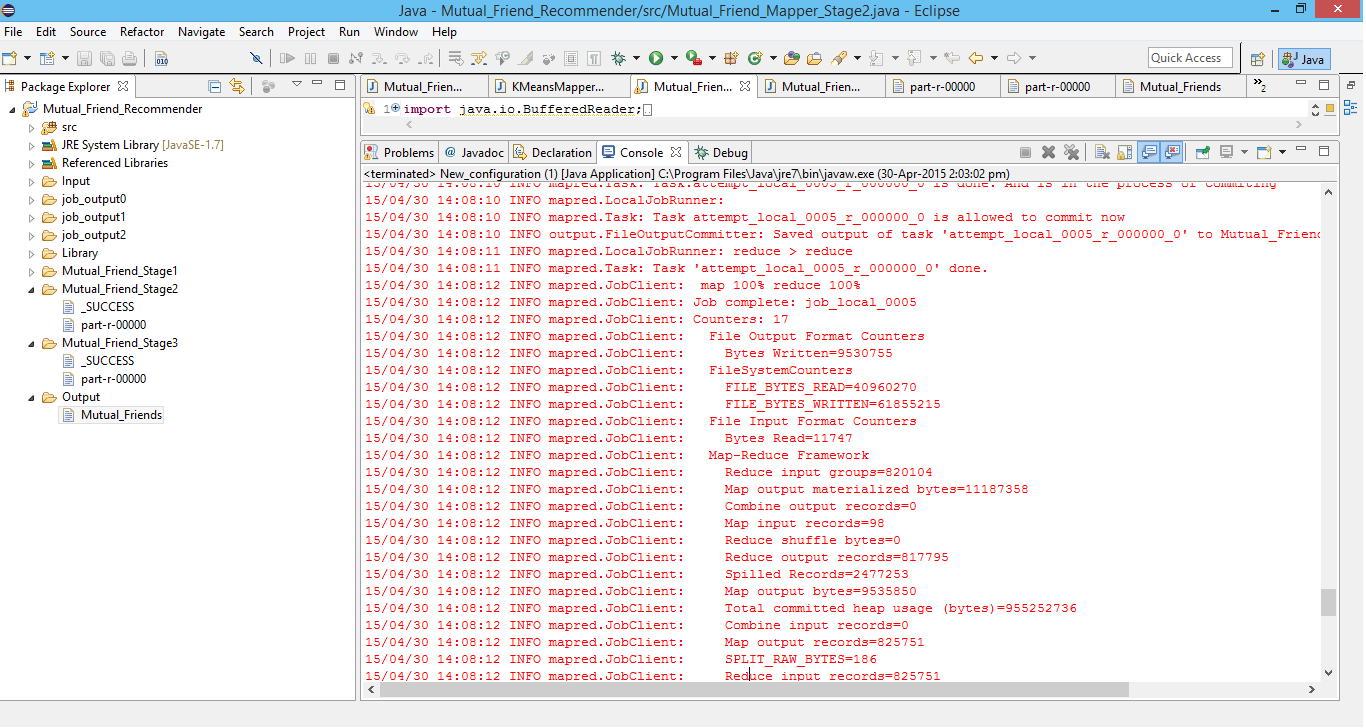
Output: Shows that user with Id 38 has the maximum 5 mutual friends with user 16



Shows the starting time of the program,



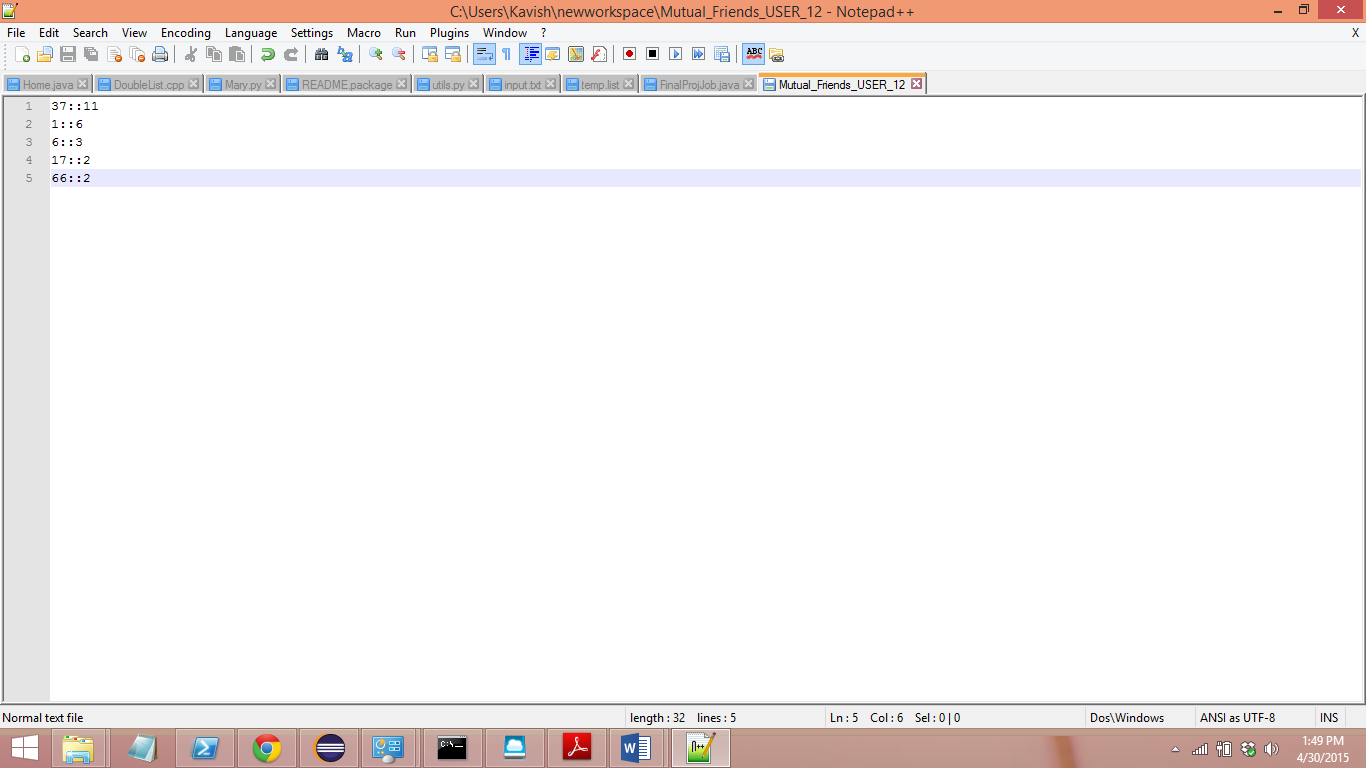
Shows the ending time for the computation for the 5K checkins,



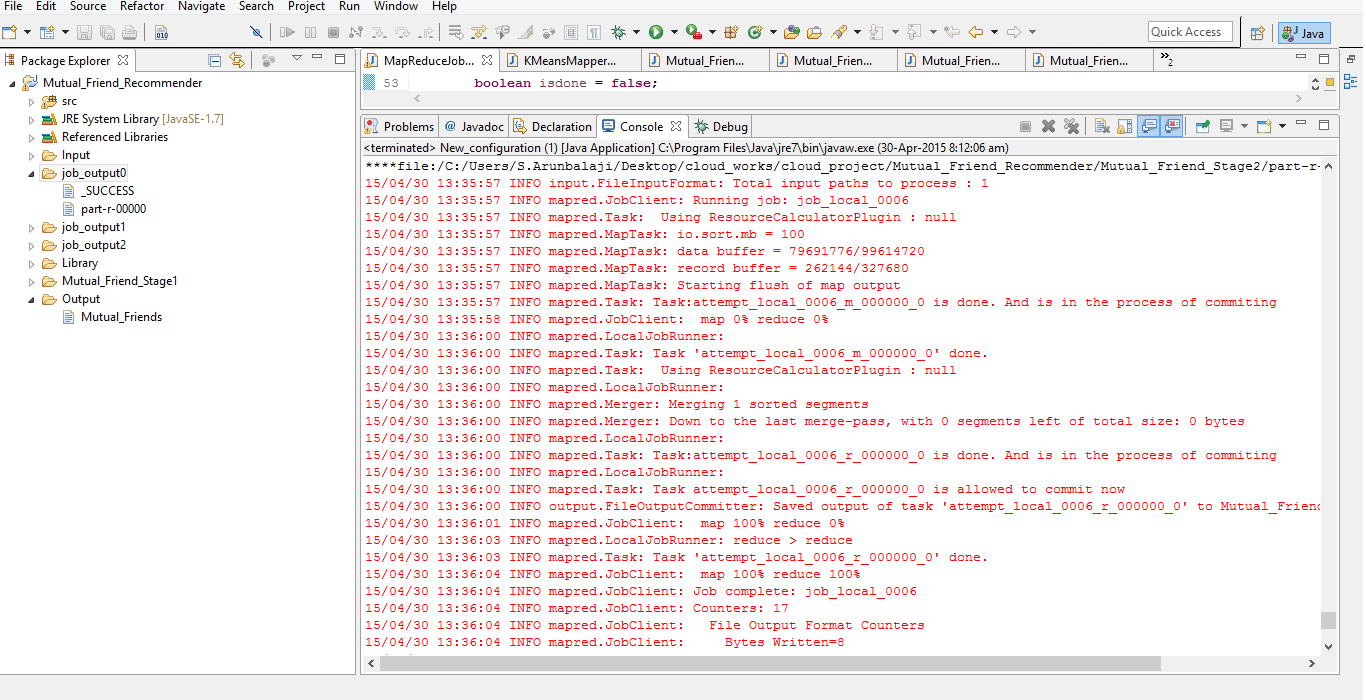
So the computation was finished in 5 min for 5000records in the check-ins file.

2) Running for suggesting friends for userid 12 with total check-ins of 50K and edges of 50K

Output: Shows that user with Id 37 has the maximum 11 mutual friends with user 12



This screenshot shows the time required for the computation. It started at 10.30am and finished at 13.36pm



This chart shows the timings required for different sizes of Input,

We don’t have results of similar live application running times which processes the similar operation using linear processing so we cannot compare it with a live application. But we have optimized the processing using iterative map reduce for forming the clusters and restricting the later computations to that cluster. Then again using the chained map reduce for suggesting the friend to the person instead of following the linear sparse matrix approach. One more disadvantage in creating sparse matrix is that it will huge memory to store the edges of each and every user.

You can see the difference in following graph of running times if we run using linear method of sparse matrix rather than using the map reduce algorithm,

**Learning from Development Process:**

1. Correct use of data structures can reduce the size and time of the application by a lot when big data is considered.(Avoiding use of sparse matrix)
2. Map reduce model saves a lot of time when compared to the linear model.
3. Running times and memory can get exponential if proper algorithms are not used.(Using the Clustering machine learning algorithm and reuse of output data in iterative map reduce stages)
4. To follow programming practices so that the code does not fail when running in distributed processing systems.

**Team contribution:**

**Kavish:**

Wrote the algorithm for the K-Means clustering using map reduce which processes the big datasets containing the list of multiple check-in details of all the users and generated multiple sets of distributed clusters.

Wrote the code for third map reduce which collects,sorts and suggests the friend with the maximum number of mutual friends.

Drafted the final report, explaining the K-Means clustering and Map-Reduce application and how it was modified to fit the problem statement, analyzed the performance of the application by running it against different sized datasets and environments.

**Arunbalaji Sivakumar:**

Collected the datasets of the users in the social networking site called Brighkite which provides the large dataset containing the check-in details and friends list of all its users.

Wrote the code for the Second Map-Reduce which uses the datasets from the selected cluster to identify the mutual friends fro the specified user.

Installed and setup Hadoop and Windows Azure, executed the application on the cluster with our datasets for performance analysis comparison.

**References**:

**[1]**<http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6579448&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6579448>

**[2]** http://www.movable-type.co.uk/scripts/latlong-vincenty.html

**[3]** <https://snap.stanford.edu/data/loc-brightkite.html>