FACULTY OF ENGINEERING

Lab2: Parallel K-Means using Hadoop

Distributed Systems

Presented by

1. **Mohamed Said 57**
2. **Ahmed el bawab 8**
3. **Khalil Ismail 23**

3/11/2020

A very common task in data analysis is grouping a set of unlabeled data such that all elements within a group are more similar among them than they are to the others. This falls under the field of unsupervised learning. Unsupervised learning techniques are widely used in several practical applications, With the development of information technology, data volumes processed by many applications will routinely cross the peta-scale threshold, which would in turn increase the computational requirements. Efficient parallel clustering algorithms and implementation techniques are the key to meeting the scalability and performance requirements entailed in such scientific data analyses.
The Hadoop and the MapReduce programming model represents an easy framework to process large amounts of data where you can just implement the map and reduce functions and the underlying system will automatically parallelize the computations across large-scale clusters, handling machine failures, inter-machine communications, etc.

Table of Contents

1 Problem Definition

With the development of information technology, data volumes processed by many applications will routinely cross the peta-scale threshold, which would in turn increase the computational requirements. Efficient parallel clustering algorithms and implementation techniques are the key to meeting the scalability and performance requirements entailed in such scientific data analyses.

Big data has become popular for processing, storing and managing massive volumes of data. The clustering of datasets has become a challenging issue in the field of big data analytics.

The K-means algorithm is best suited for finding similarities between entities based on distance measures with small datasets. Existing clustering algorithms require scalable solutions to manage large datasets. There are two approaches to the clustering of large datasets using MapReduce.

The first approach, K-Means Hadoop MapReduce (KM-HMR), focuses on the MapReduce implementation of standard K-means.

And the other is the sequential K-Means which is the normal implementation of the Algorithm.

2 Goals

1. Implementing a parallel version of the K-Means algorithm using the MapReduce framework.

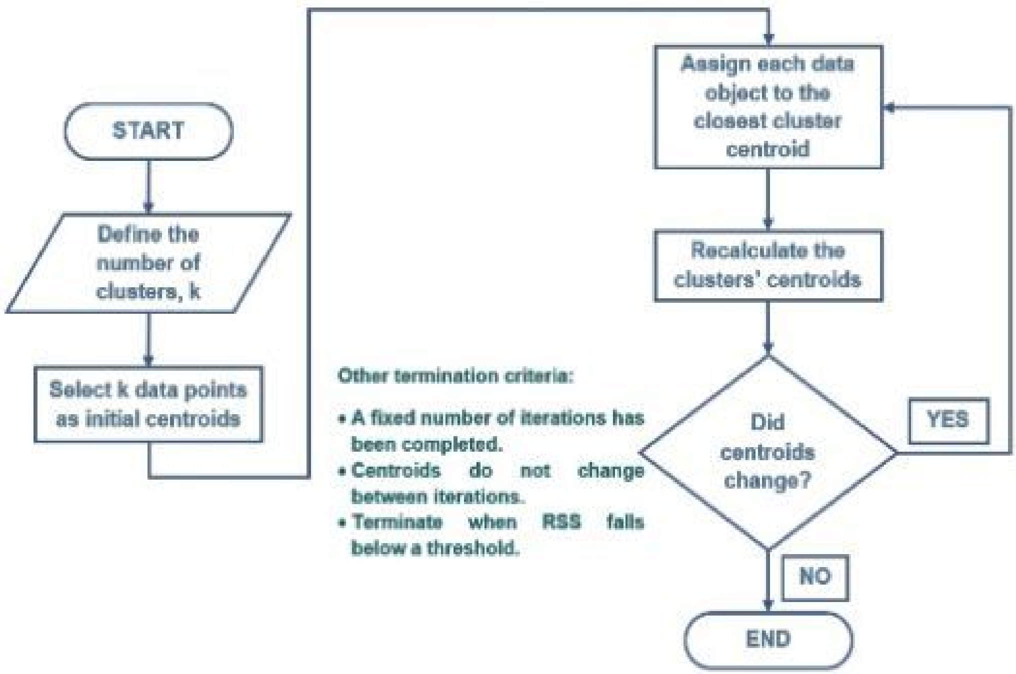
2. Implementing an unparalleled version of the K-Means algorithm using normal implementation.

3. Evaluating the parallel version and comparing it with the unparalleled version of K-Means using the IRIS dataset in terms of run time and clustering accuracy.

3 Algorithms & Code description

Here is a comparison between Unparalleled K-Means and Paralleled K-Means using Map-Reduce technique:

<u>Unparalleled K-Means:</u>



<u>Paralleled K-Means:</u>

We will make each iteration of the K-Means as a map-reduce phase.

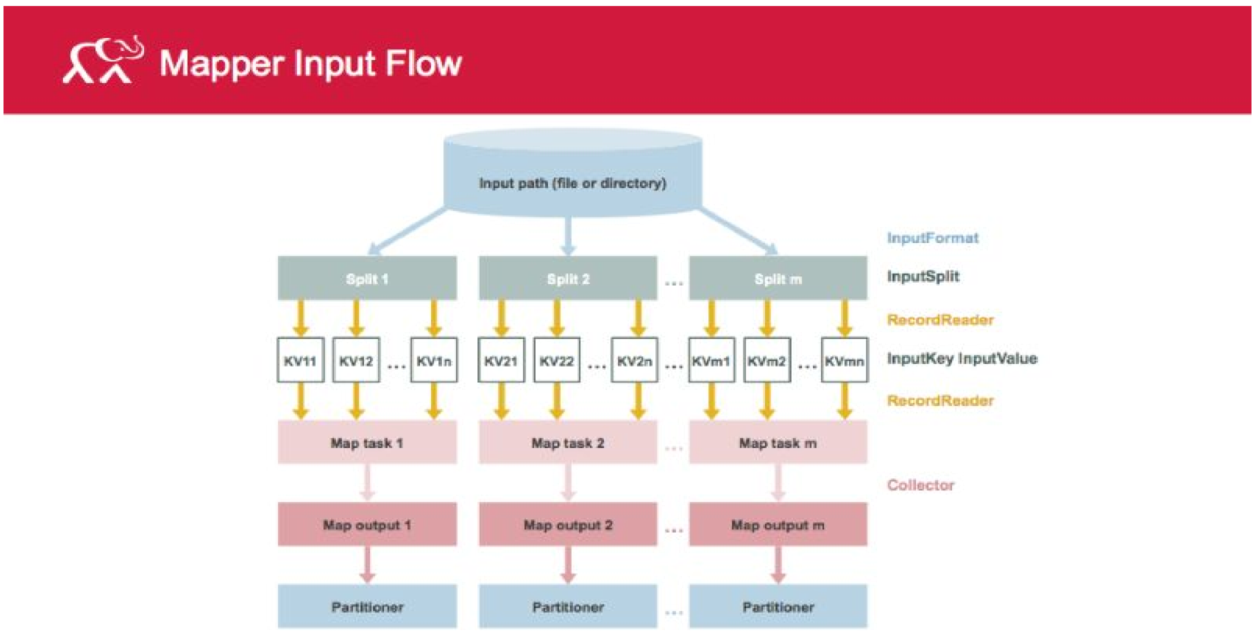
We will follow the following map and reduce procedure.

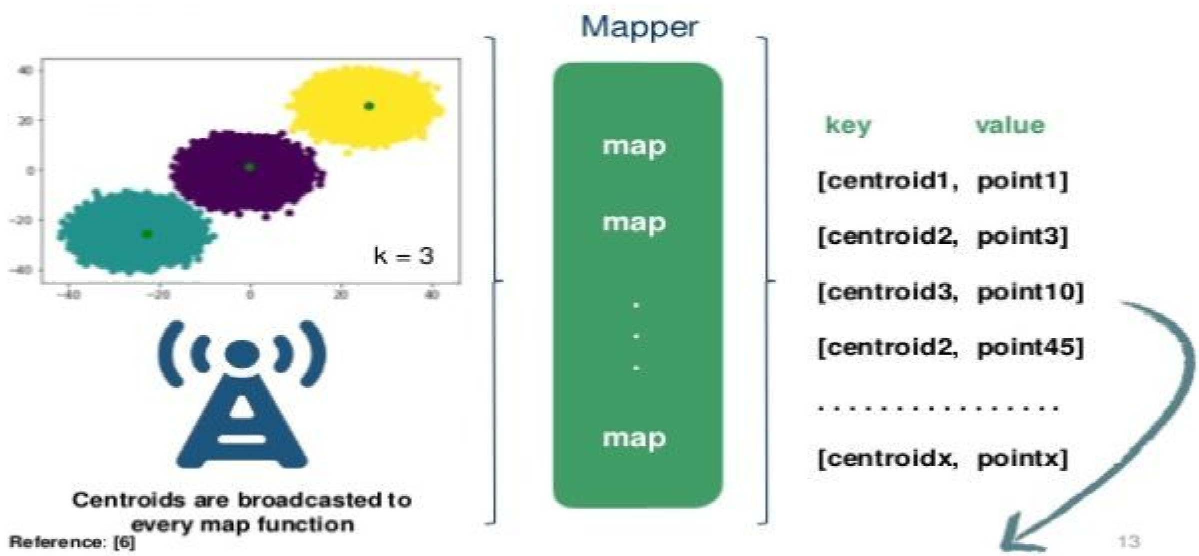
```
Do
    Map
        Input is a data point and k centers are broadcasted
        Finds the closest center among k centers for the input point

    Reduce
        Input is one of k centers and all data points having this center as their closest center.
        Calculates the new center using data points
Until all of new centers are not changed
```

3 Algorithms & Code description

Map Phase:

The Mapper has input of a single point then it calculates the closet cluster centroid to be assigned to it.

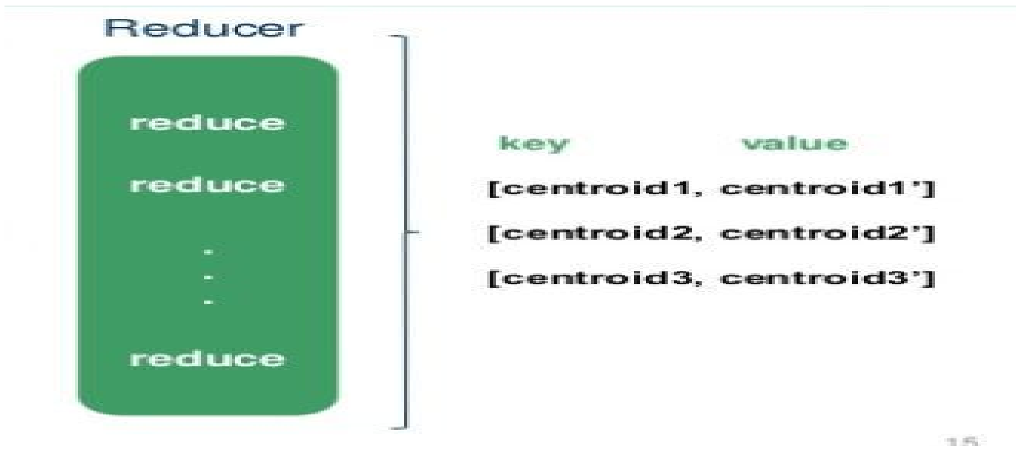




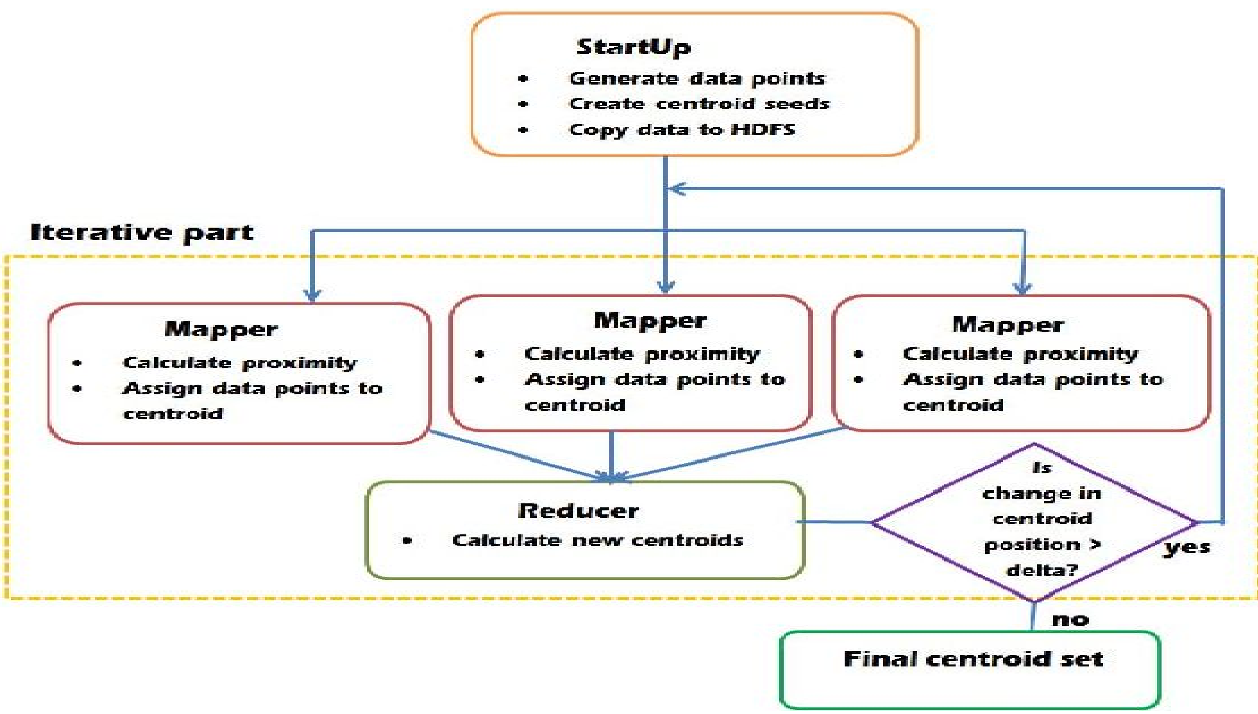
3 Algorithms & Code description

Reduce Phase:

The Reducer will take the cluster index as a key, and the list of points which are assigned to that cluster. Then it will update the cluster centroid by finding the mean of the assigned points.



Over All MapReduce K-means Using Hadoop:



4 Implementation

First, Centroids and Context (Configuration) are loaded into the Distributed Cache. This is done by overriding setup function in the Mapper and Reducer class.

Afterwards, the input data file is split and each data point is processed by one of the map functions (in Map process).

The function writes key-value pairs <Centroid, Point>, where the Centroid is the closest one to the Point.

The Reducer performs the same procedure as the Combiner, but it also checks whether centroids converged (no change) then the global Counter remains unchanged, otherwise, it is incremented.

After the one iteration is done, new centroids are saved and the program checks a condition, if the total distance (no change in centroids) value is unchanged. If then the program is finished, otherwise, the whole MapReduce process is run again with the updated centroids.

Here are some references I used to help in implementing the K-Means:

https://www.slideshare.net/LampriniKoutsokera/implementation-of-k-means-algorithm-on-hadoop

shorturl.at/lyJY6 shorturl.at/apPVW shorturl.at/eisQ2

Note that there are some other implementations which use combiner between the map and reduce but we don't use it here.

Algorithm 1 Algorithm for Startup

Require:

- A set of d-dimensional objects X= $\{x_1, x_2, \ldots, x_n\}$
- k-number of clusters where $k < n$
- initial set of centroids $C = \{c_1, c_2, \ldots, c_k\}$
- δ convergence delta

Output: a new set of centroids, number of iterations, final clusters, time taken to converg

$load\ (X, C)$
$current\_centroids \Leftarrow C$
$initialise$ numIter, timetoConverge, finalClusters
$startTime \Leftarrow currentTime()$
$C' \Leftarrow perform\ MapReduce$
$new\_centroids \Leftarrow C'$
$numIter \Leftarrow 1$
while $change(new\_centroids, current\_centroids) > \delta$ **do**
    $current\_centroids \Leftarrow new\_centroids$
    $C' \Leftarrow$ perform MapReduce
    $new\_centroids \Leftarrow C'$
    $numIter \Leftarrow numIter + 1$
end while
$endTime \Leftarrow currentTime()$
$timetoConverge \Leftarrow (endTime - startTime)$
$perform\ outlierRemoval$
$finalClusters \Leftarrow perform\ finalClustering$
$writeStatistics$
return $current\_centroids, numIter, finalClusters, timetoConverge$

Algorithm 2 Algorithm for Mapper

Require:

- A subset of d-dimensional objects of $\{x_1, x_2, \ldots, x_n\}$ in each mapper
- initial set of centroids $C = \{c_1, c_2, \ldots, c_k\}$

Output: A list of centroids and objects assigned to each centroid separated by tab. This list is written locally one line per data point and read by the Reducer program.

$M_1 \Leftarrow \{x_1, x_2, \ldots, x_m\}$
$current\_centroids \Leftarrow C$
$distance\ (p, q) = \sqrt{\sum_{i=1}^{d} (p_i - q_i)^2}$ where p_i (or q_i) is the coordinate of p (or q) in dimension i
for all $x_i \in M_1$ such that $1 \le i \le m$ **do**
    $bestCentroid \Leftarrow null$
    $minDist \Leftarrow \infty$
    for all $c \in current\_centroids$ **do**
        $dist \Leftarrow distance(x_i, c)$
        if $bestCentroid = null \mid\mid dist < minDist$ **then**
            $minDist \Leftarrow dist$
            $bestCentroid \Leftarrow c$
        end if
    end for
    $outputlist << (bestCentroid, x_i)$
    i +=1
end for
return $outputlist$

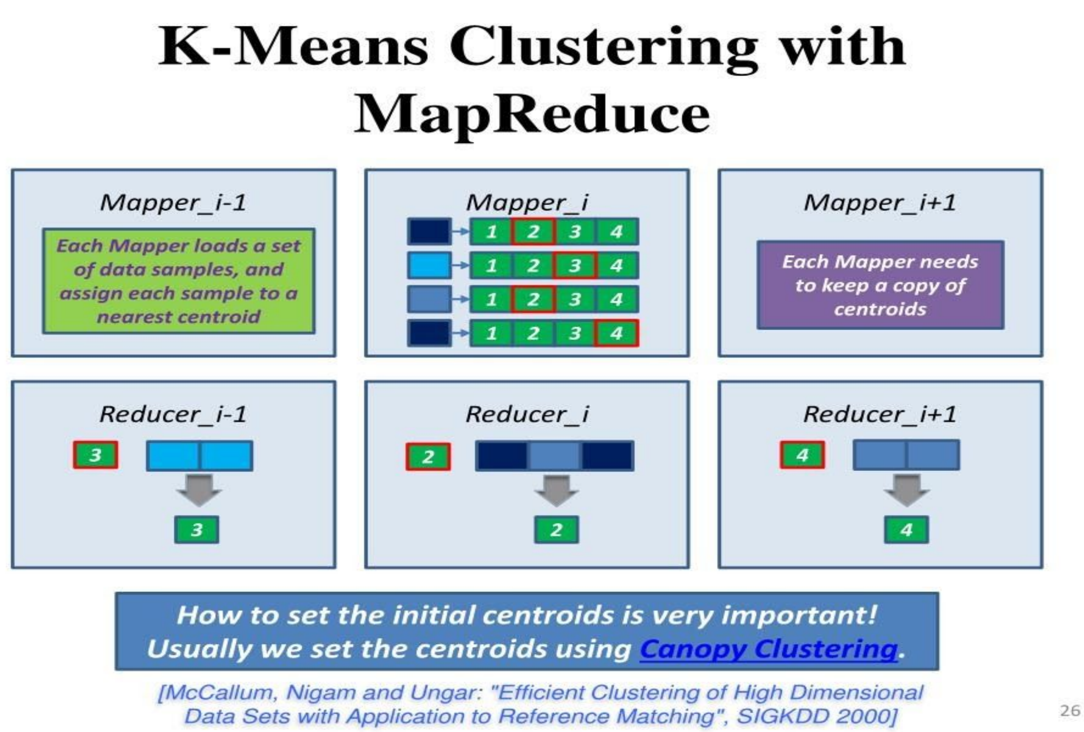
Algorithm 3 Algorithm for Reducer

Require:

Input: (key, value) where key = bestCentroid and value = objects assigned to the centroids by the mapper.

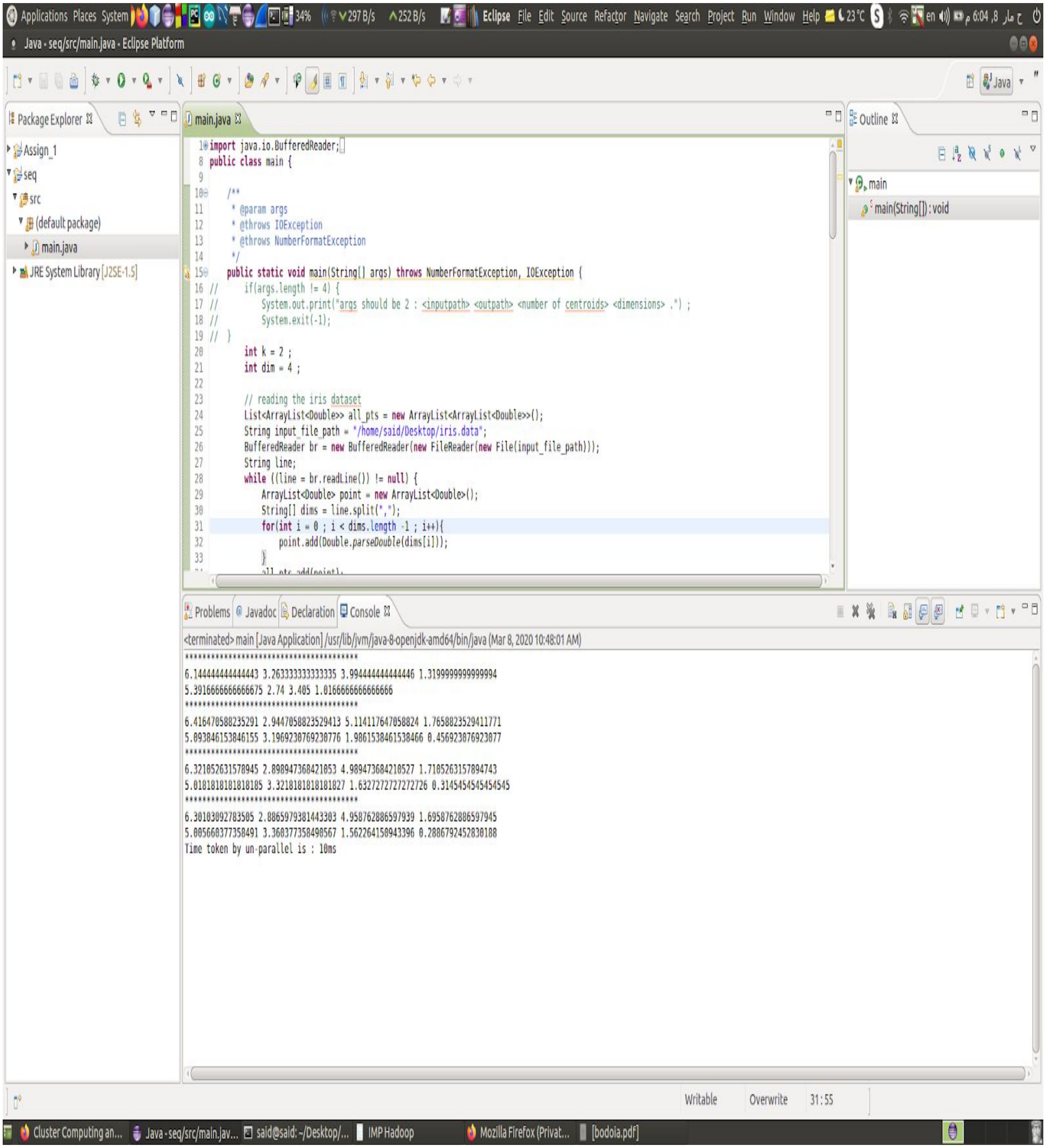
Output: (key, value) where key = oldcentroid and value = newBestCentroid which is the new centroid value calculated for that bestCentroid.

$outputlist \Leftarrow outputlists$ from mappers
$v \Leftarrow \{\}$
$newCentroidList \Leftarrow null$
for all $y \in outputlist$ **do**
    $centroid \Leftarrow y.key$
    $object \Leftarrow y.value$
    $v[centroid] \Leftarrow object$
end for
for all $centroid \in v$ **do**
    $newCentroid, sumofObjects, numofObjects \Leftarrow null$
    for all $object \in v[centroid]$ **do**
        $sumofObjects += object$
        $numofObjects += 1$
    end for
    $newCentroid \Leftarrow (sumofObjects \div numofObjects)$
    $newCentroidList << (newCentroid)$
end for
return $newCentroidList$



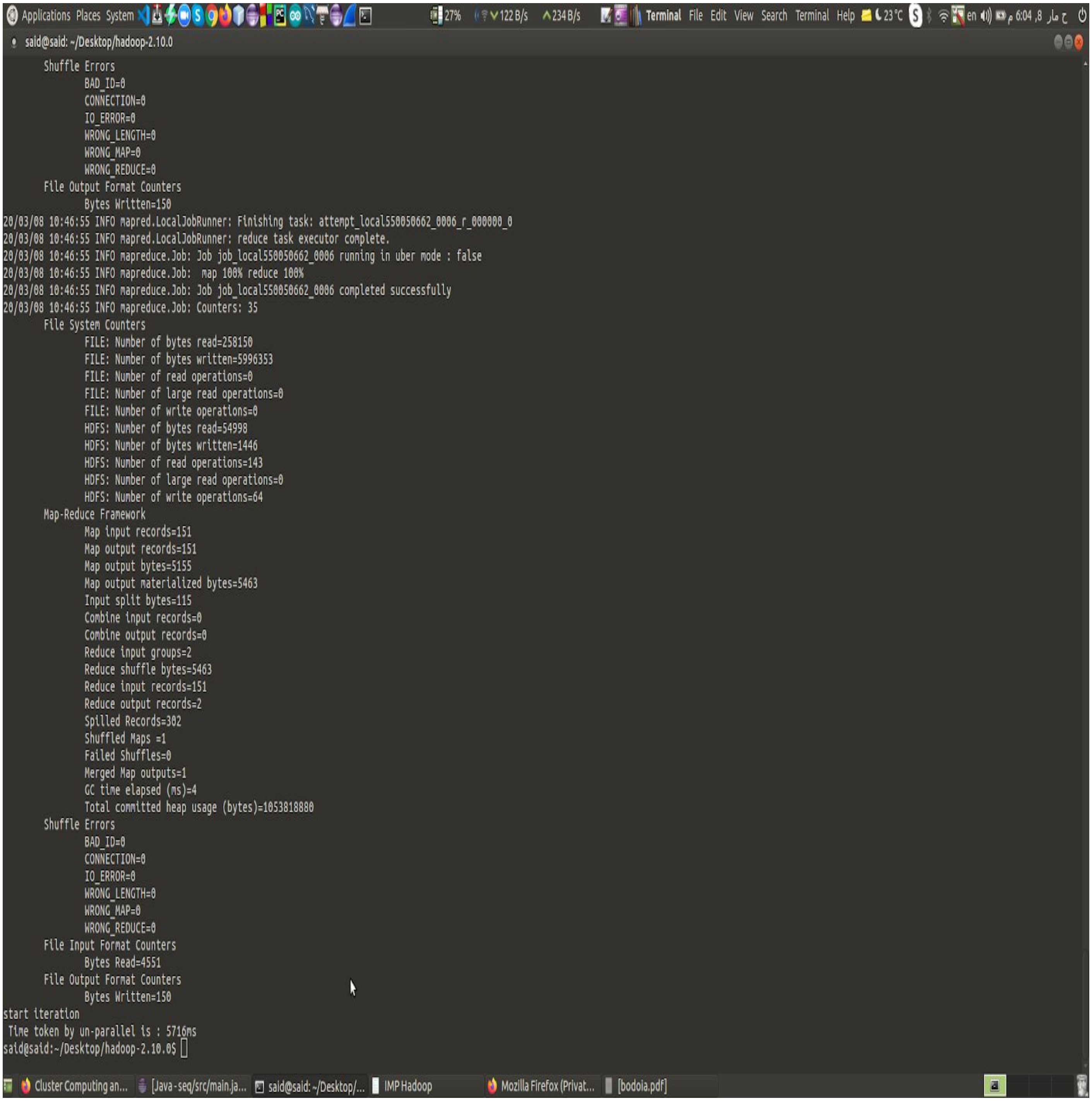
5 Sample Runs

Un-Parallel K-means (Sequential):



5 Sample Runs

<u>Parallel K-means (MapReduce) Hadoop:</u>



K-Means will finish after 14 iterations, so we will find 14 output directory.

6 Results & Conclusion

Both the parallel K-Means and unparallel K-Means versions will have the same final cluster centroids as following.

When using only 2 centroids for iris dataset:

[6.30103093 2.88659794 4.95876289 1.69587629]

[5.00566038 3.36037736 1.56226415 0.28867925]

The Sequential Un-Parallel K-Means takes 10 ms.

The Parallel MapReduce Hadoop K-Means takes 5716 ms

Notice that the same results obtained using the built-in K-Means in Sklearn.clusters package in python.