FACULTY OF ENGINEERING

Lab 2: Parallel K-Means using Hadoop

Distributed Systems

Presented by

Names	IDs
Aaser Fawzy Zakaria Hassan	**19015403**
Gamal Abdel Hamid Nasef Nowesar	**19015550**
Mohamed Ezzat Saad El-Shazly	**19016441**

9/3/2024

This is our implementation of the K-means algorithm using parallelization with Hadoop's MapReduce API by dividing the work of distance calculations to different clusters (through mappers) and calculating the new means (using reducers).

Table of Contents

1 Problem Definition

It is required to

- implement a parallelized K-means clustering algorithm using Apache's Hadoop MapReduce programming framework
- evaluate the results of the output on the IRIS dataset
- to have the algorithm accept any feature vector size.

2 Algorithms

As stated before, it was required to implement a parallelized K-means clustering algorithm which flows as follows:

```
K-Means algorithm (dataset, max iterations, number of clusters) {
      Centroids = Initialize Random centroids from dataset
      Converged = False
      While (NOT Converged AND iteration < Max iterations) {
            For Each data point
                  Find closest centroid to the point and assign it to its
                  cluster

            For Each cluster
                  Calculate the new mean for the cluster

            If (old centroids – new centroids < threshold)
                  Converged = True
      }
}
```

The Parallel K-means using the MapReduce framework requires us to implement 2 methods the mapping method and the reduce method.

For the mapping method:

```
K-MeansMapper algorithm (centroids, data point){

      Find the closest centroid to the point

      Key = closest centroid index

      Value = data point

      Emit (Key, value)

}
```

For the reduction method

```
K-MeansReducer algorithm (centroid Index, data points[] ){
      Find sum of all data points
      New centroid = (sum / number of points)
      Emit (centroid Index, new centroid)
}
```

For the driver method

```
K-MeansDriver algorithm (dataset, max iterations, number of clusters){
      Centroids = initialize random centroids from dataset
      Converged = False
      While (NOT Converged and iteration < max iterations){
            Setup jobs for MapReduce framework given the initial centroids
            New centroids = Read output file containing centroids
            If(old centroids – new centroids < threshold)
                  Converged= True
      }
}
```

4 Implementation

- Environment:
    - Local machine setup using a personal PC in a virtualized environment using Cloudera Virtual machine (on VBox)
    - Machine specification (virtual machine)
        - Operating System: CentOS
        - CPU: 4 cores @ 2.5 GHz
        - Memory: 8 GB RAM
        - Storage: 50 GB Disk space HDD

- Test Dataset (test cases)
    - IRIS dataset: it was required to evaluate our algorithm on the IRIS dataset, since it is a simple dataset with small sized feature vector to easily visualize.

- Multiple runs on the dataset:
    - Executed the algorithm multiple times to assess consistency, performance under different conditions, and identify potential issues for both k=2 and k=3.

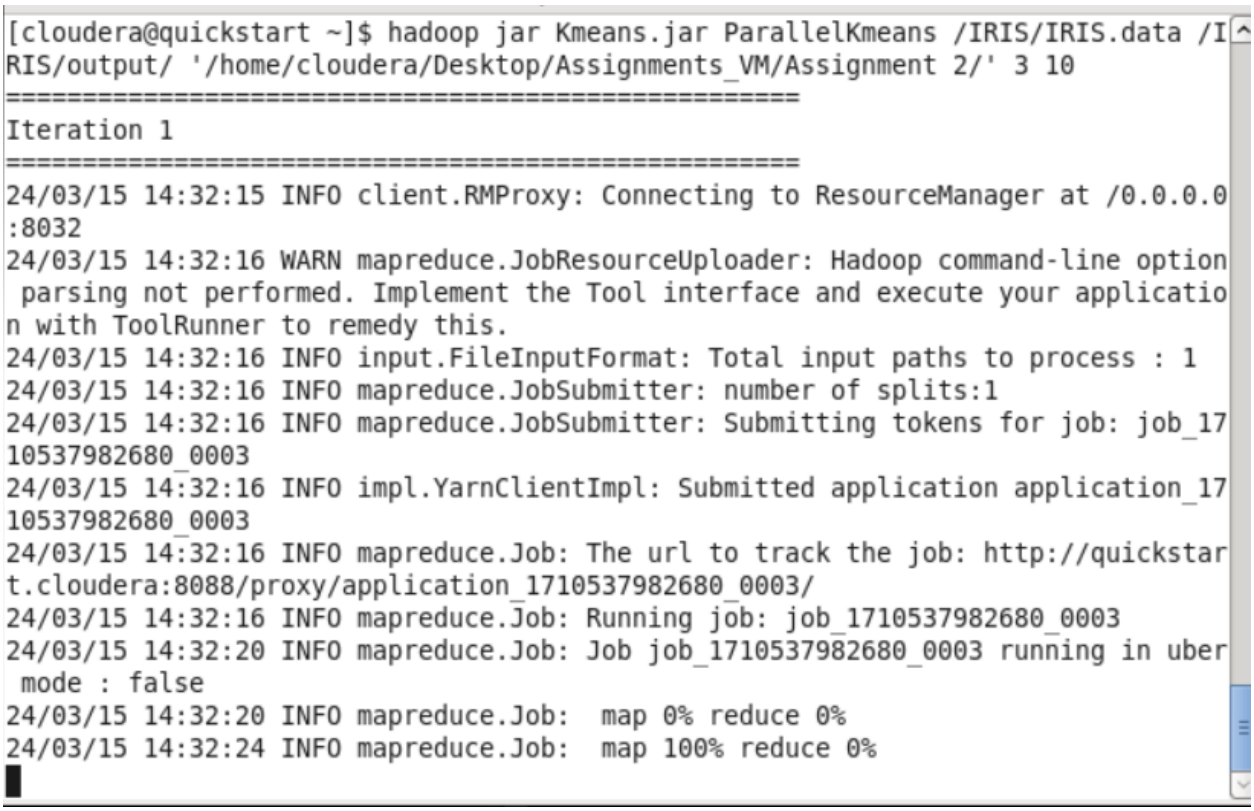
5 Results

A Note about the runtime

for the runtime a normal k-means (done using python) was much faster (0.0097 seconds unparallel & 28 seconds for the parallel) as there was overhead due to creating jobs and synchronizing mappers and reducers together which makes most of the time spent was either preparation or communications, unlike the normal k-means that does not require all of this overhead.

Parallelization would fetch far more efficiency given a larger dataset and a large number of clusters (nodes).

Sample run for K=3



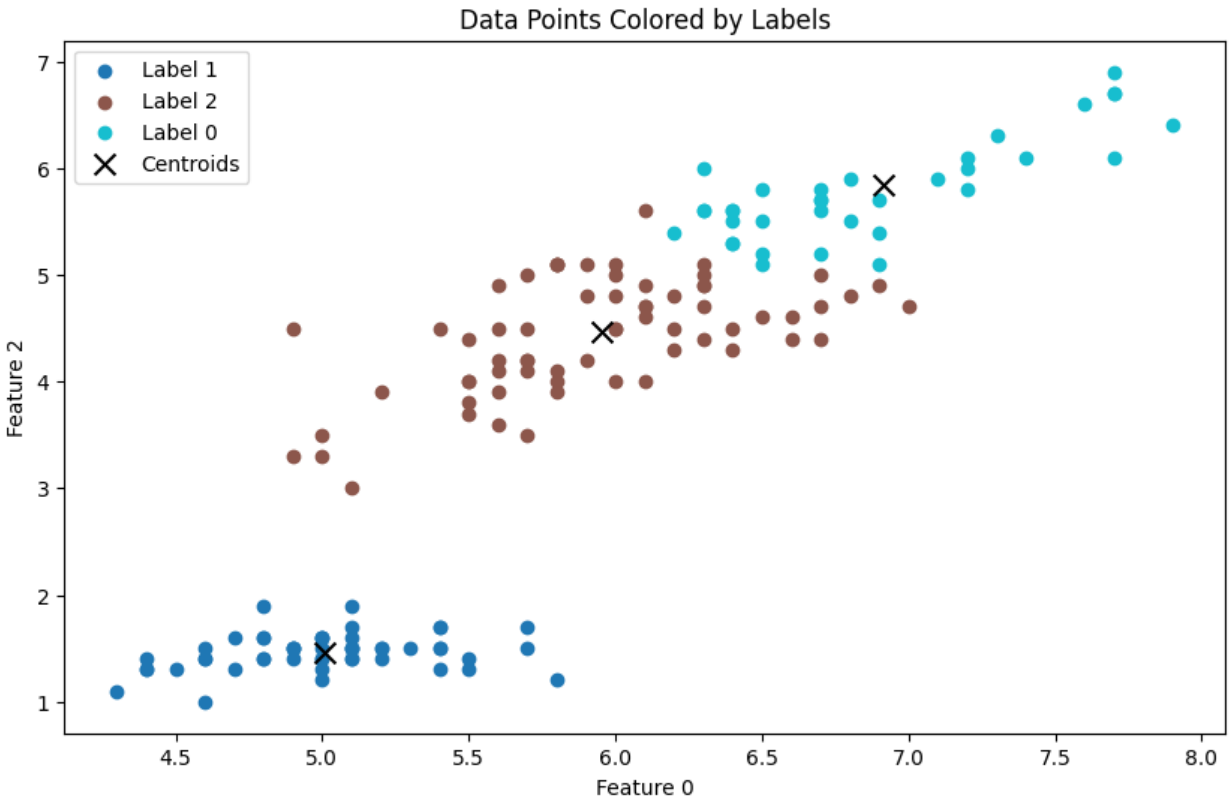
```
1       5.006000000000001,3.4180000000000006,1.4640000000000004,0.244
2       6.027848101265823,2.8000000000000003,4.603797468354429,1.550632911392405

==========================================================
Iteration 2
==========================================================
24/03/15 14:33:09 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0
:8032
24/03/15 14:33:09 WARN mapreduce.JobResourceUploader: Hadoop command-line option
 parsing not performed. Implement the Tool interface and execute your applicatio
n with ToolRunner to remedy this.
24/03/15 14:33:09 INFO input.FileInputFormat: Total input paths to process : 1
24/03/15 14:33:09 INFO mapreduce.JobSubmitter: number of splits:1
24/03/15 14:33:09 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_17
10537982680_0005
24/03/15 14:33:10 INFO impl.YarnClientImpl: Submitted application application_17
10537982680_0005
24/03/15 14:33:10 INFO mapreduce.Job: The url to track the job: http://quickstar
t.cloudera:8088/proxy/application_1710537982680_0005/
24/03/15 14:33:10 INFO mapreduce.Job: Running job: job_1710537982680_0005
S24/03/15 14:33:15 INFO mapreduce.Job: Job job_1710537982680_0005 running in ube
r mode : false
24/03/15 14:33:15 INFO mapreduce.Job:  map 0% reduce 0%
```

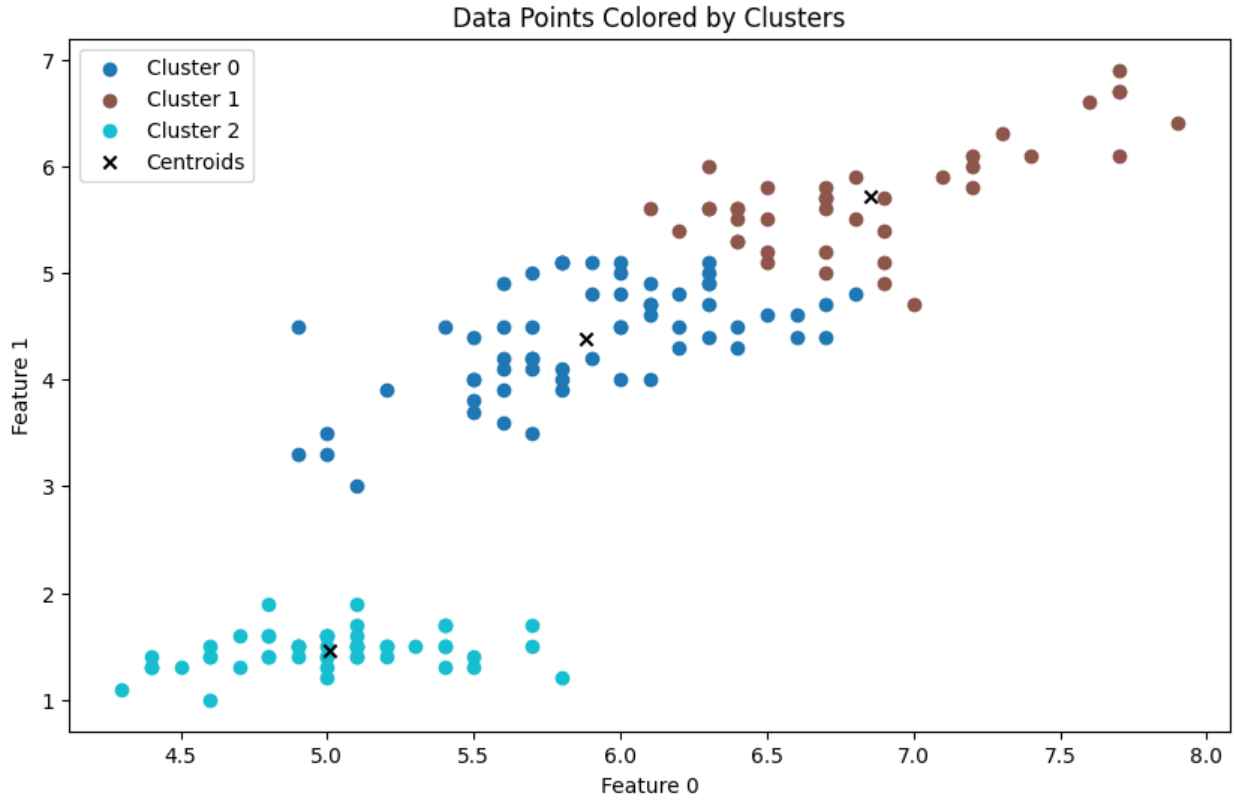
```
File   Edit   View   Search   Terminal   Help
                Physical memory (bytes) snapshot=514326528
                Virtual memory (bytes) snapshot=3147395072
                Total committed heap usage (bytes)=535298048
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=4700
        File Output Format Counters
                Bytes Written=228
0       6.912500000000001,3.1000000000000005,5.846875000000001,2.131249999999999
1       5.005999999999999,3.4180000000000006,1.4639999999999995,0.24399999999999
988
2       5.955882352941177,2.7647058823529407,4.463235294117646,1.461764705882353

---------------------------------------------------
Converged
---------------------------------------------------
Execution time: 28 seconds
[cloudera@quickstart ~]$ SS
```

Output plotted for visualizations Parallel K-means on previous run:



Output plotted for visualizations Unparallel K-means on previous run:



Evaluation Metrics ARI and NMI for both parallel and unparallel K-means on the previous run

```
Evaluation for unparallel K-means
Adjusted Rand Index (ARI): 0.7163421126838476
Normalized Mutual Information (NMI): 0.7419116631817836
================================================
Evaluation for parallel K-means
Adjusted Rand Index (ARI): 0.9239067985955953
Normalized Mutual Information (NMI): 0.9134361171137956
```

6 Conclusion

- In this lab we were able to learn more about the map reduce framework for Hadoop.
- We were able to interact directly more with the HDFS through Apache's java API.
- We were able to implement parallelized code using the MapReduce framework.

7 References

- sklearn.metrics.normalized_mutual_info_score — scikit-learn 1.4.1 documentation
- sklearn.metrics.adjusted_rand_score — scikit-learn 1.4.1 documentation