Distributed Systems
Lab 2

Parallel K-Means using Spark

Members

- Amr Mohamed Nasreldin Elsayed (47)
- Michael Raafat Mikhail (57)
- Amira Nabil Haiba (18)

Unparallel K-means Pseudo-Code

```
Function K-means ( K : number of clusters, D : dataset of samples) :
    1. Initialize k cluster centroid randomly : M(1), M(2), …. to M(k).
    2. Repeat Until Convergence:
        a. For every sample i in D :
            i. C(i) = argmin_j(ecludian_distance(D(i) - M(j))
        b. For j from 1 to K:
            i. M(j) = Mean(any sample i where C(i) == j)
    3. Return the cluster centroids.
```

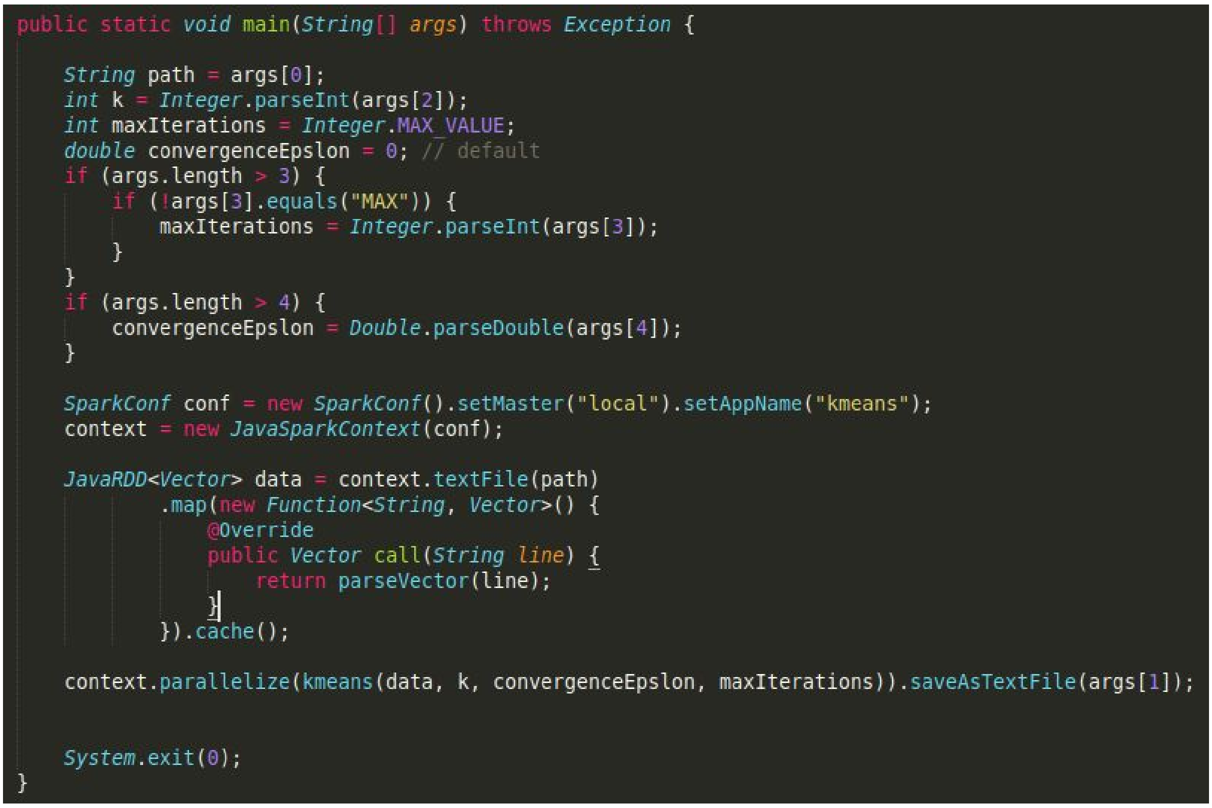
Parallel K-means Pseudo-Code

```
Function K-means ( K : number of clusters, D : dataset of samples) :
    1. Initialize k cluster centroid randomly : M(1), M(2), …. to M(k).
    2. Repeat Until Convergence:
        a. Map each sample i in D to a centroid in parallel:
            i. C(i) = argmin_j(ecludian_distance(D(i) - M(j))
            ii. Return Pairs of (C(i), D(i))
        b. For j from 1 to K, calculate each centroid new mean in parallel by reducing
           pairs with same key C(i):
            i. M(j) = Mean(any sample i where C(i) == j)
    3. Return the cluster centroids.
```

Map-Reduce algorithm

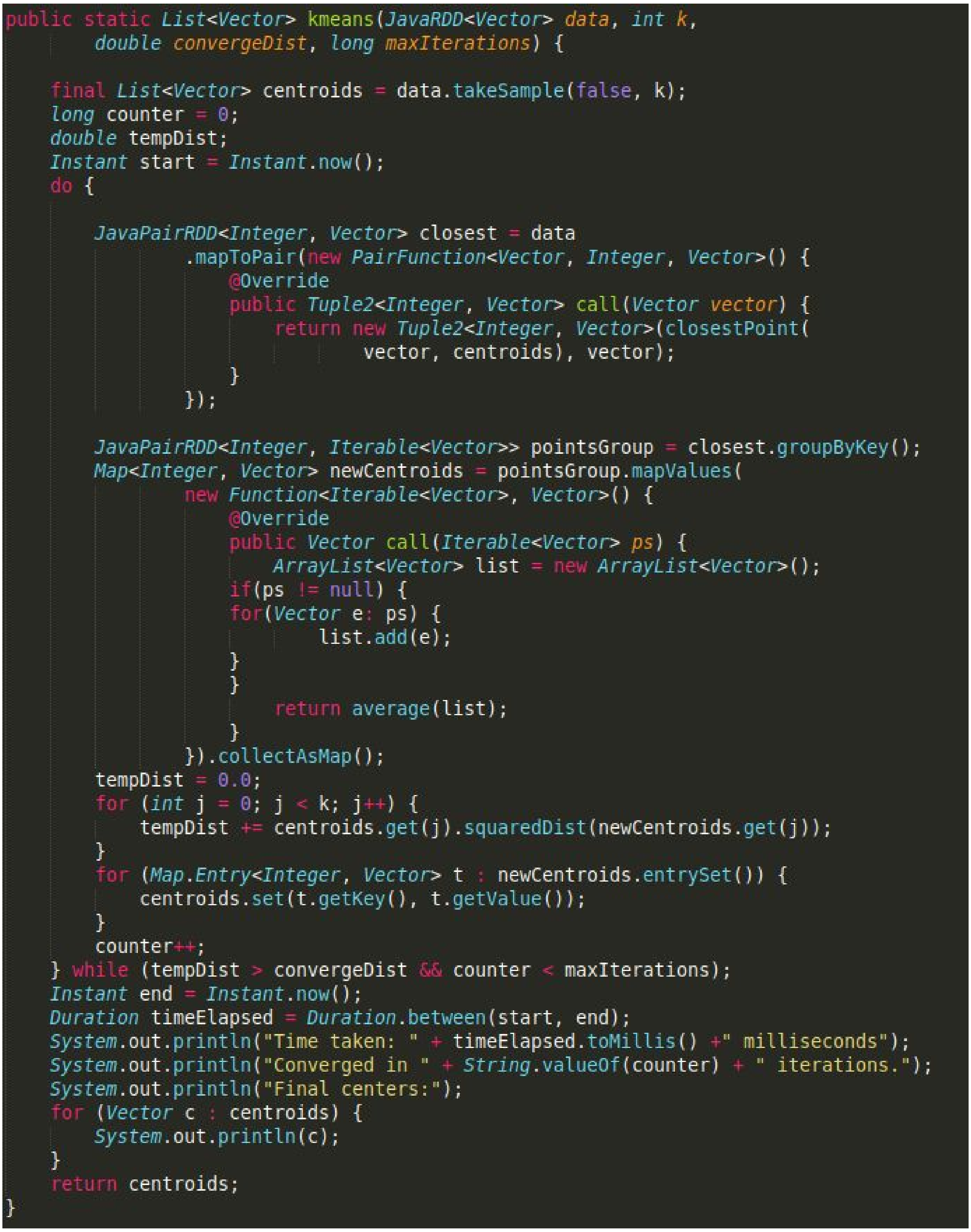
- The program takes the path of the input file, number of means and an optional maximum number of iterations, and change threshold.
- Create spark context that takes the input file and sets the map function and returns a vector of data.
- The K-means algorithms allocates each vector to the closest centroid then group by cluster id and average the vectors within each cluster to compute new centroids until the max number of iterations is reached or change in centroids is less than or equal the threshold.

Main method



```java
public static void main(String[] args) throws Exception {

    String path = args[0];
    int k = Integer.parseInt(args[2]);
    int maxIterations = Integer.MAX_VALUE;
    double convergenceEpslon = 0; // default
    if (args.length > 3) {
        if (!args[3].equals("MAX")) {
            maxIterations = Integer.parseInt(args[3]);
        }
    }
    if (args.length > 4) {
        convergenceEpslon = Double.parseDouble(args[4]);
    }

    SparkConf conf = new SparkConf().setMaster("local").setAppName("kmeans");
    context = new JavaSparkContext(conf);

    JavaRDD<Vector> data = context.textFile(path)
            .map(new Function<String, Vector>() {
                @Override
                public Vector call(String line) {
                    return parseVector(line);
                }
            }).cache();

    context.parallelize(kmeans(data, k, convergenceEpslon, maxIterations)).saveAsTextFile(args[1]);


    System.exit(0);
}
```

K-means method

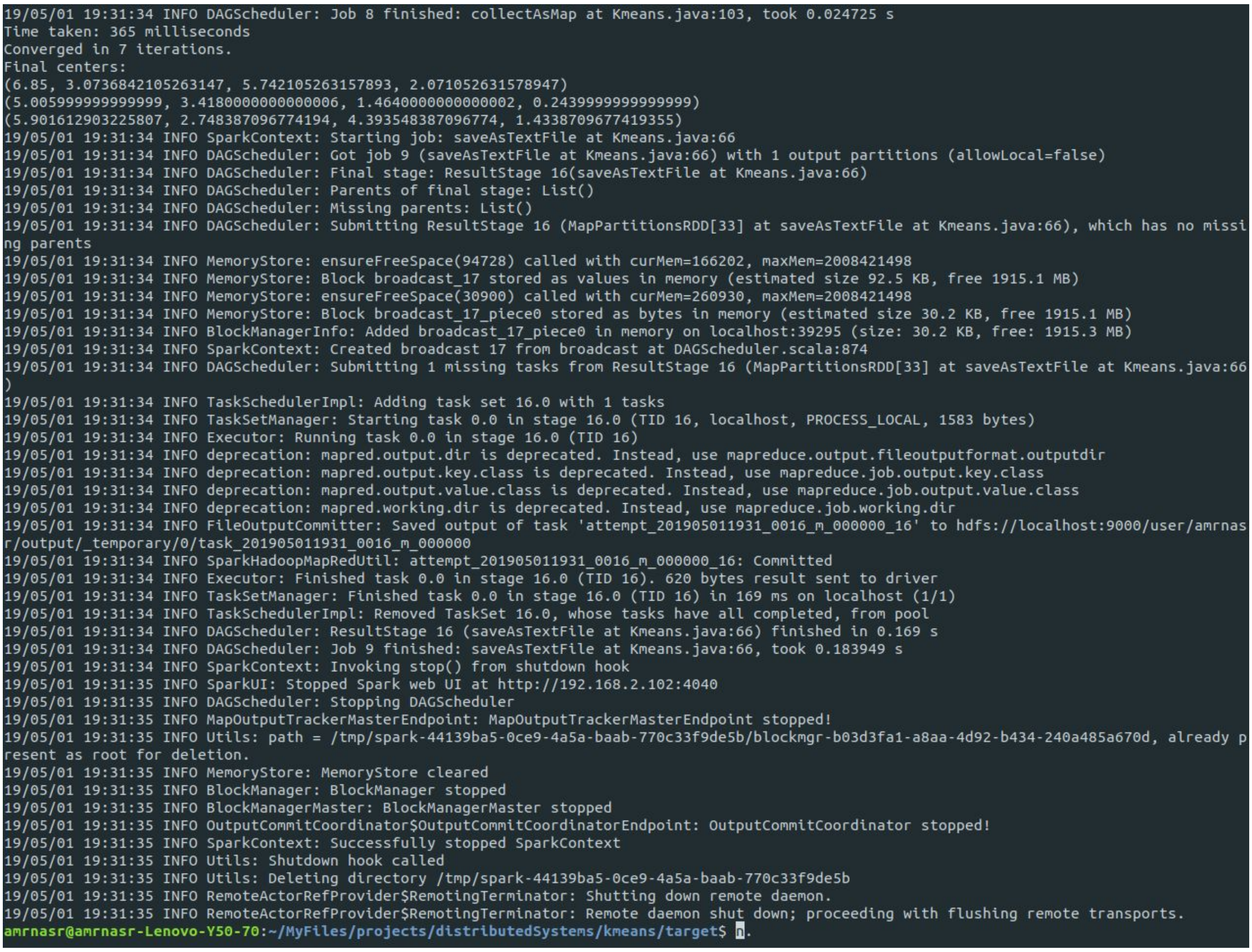


```java
public static List<Vector> kmeans(JavaRDD<Vector> data, int k,
        double convergeDist, long maxIterations) {

    final List<Vector> centroids = data.takeSample(false, k);
    long counter = 0;
    double tempDist;
    Instant start = Instant.now();
    do {

        JavaPairRDD<Integer, Vector> closest = data
                .mapToPair(new PairFunction<Vector, Integer, Vector>() {
                    @Override
                    public Tuple2<Integer, Vector> call(Vector vector) {
                        return new Tuple2<Integer, Vector>(closestPoint(
                                vector, centroids), vector);
                    }
                });

        JavaPairRDD<Integer, Iterable<Vector>> pointsGroup = closest.groupByKey();
        Map<Integer, Vector> newCentroids = pointsGroup.mapValues(
                new Function<Iterable<Vector>, Vector>() {
                    @Override
                    public Vector call(Iterable<Vector> ps) {
                        ArrayList<Vector> list = new ArrayList<Vector>();
                    if(ps != null) {
                    for(Vector e: ps) {
                            list.add(e);
                    }
                    }
                        return average(list);
                    }
                }).collectAsMap();
        tempDist = 0.0;
        for (int j = 0; j < k; j++) {
            tempDist += centroids.get(j).squaredDist(newCentroids.get(j));
        }
        for (Map.Entry<Integer, Vector> t : newCentroids.entrySet()) {
            centroids.set(t.getKey(), t.getValue());
        }
        counter++;
    } while (tempDist > convergeDist && counter < maxIterations);
    Instant end = Instant.now();
    Duration timeElapsed = Duration.between(start, end);
    System.out.println("Time taken: " + timeElapsed.toMillis() +" milliseconds");
    System.out.println("Converged in " + String.valueOf(counter) + " iterations.");
    System.out.println("Final centers:");
    for (Vector c : centroids) {
        System.out.println(c);
    }
    return centroids;
}
```

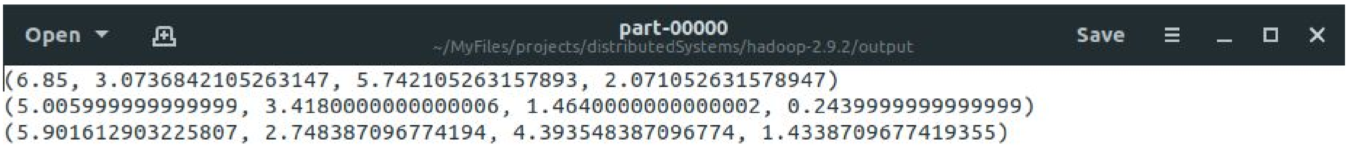
Challenges Faced

- **Passing Feature Row per Sample in Mapper:**
  We decided to parse each line that represents values of features per sample and separating them by the delimiter ‘,’, then converting these values to a vector of double values.
- **How to get Initial Centroid:**
  We decided to set initial centroid randomly picking k-samples from the initial file.
- **How to pass results of each round:**
  Pass centroids as arguments to the mapping function, and update their values with each round.
- **Number of clusters:**
  Take it as a command line argument from user.
- **Termination condition:**
  Either terminate with a maximum number of iterations, or when change to centroides is less than or equal a certain threshold. Both maximum number of iterations and the threshold are command line arguments by the user with default values of maximum possible integer for number of iterations and a threshold of 0 for the change.

Results



```
19/05/01 19:31:34 INFO DAGScheduler: Job 8 finished: collectAsMap at Kmeans.java:103, took 0.024725 s
Time taken: 365 milliseconds
Converged in 7 iterations.
Final centers:
(6.85, 3.0736842105263147, 5.742105263157893, 2.071052631578947)
(5.005999999999999, 3.4180000000000006, 1.4640000000000002, 0.2439999999999999)
(5.901612903225807, 2.748387096774194, 4.393548387096774, 1.4338709677419355)
19/05/01 19:31:34 INFO SparkContext: Starting job: saveAsTextFile at Kmeans.java:66
19/05/01 19:31:34 INFO DAGScheduler: Got job 9 (saveAsTextFile at Kmeans.java:66) with 1 output partitions (allowLocal=false)
19/05/01 19:31:34 INFO DAGScheduler: Final stage: ResultStage 16(saveAsTextFile at Kmeans.java:66)
19/05/01 19:31:34 INFO DAGScheduler: Parents of final stage: List()
19/05/01 19:31:34 INFO DAGScheduler: Missing parents: List()
19/05/01 19:31:34 INFO DAGScheduler: Submitting ResultStage 16 (MapPartitionsRDD[33] at saveAsTextFile at Kmeans.java:66), which has no missi
ng parents
19/05/01 19:31:34 INFO MemoryStore: ensureFreeSpace(94728) called with curMem=166202, maxMem=2008421498
19/05/01 19:31:34 INFO MemoryStore: Block broadcast_17 stored as values in memory (estimated size 92.5 KB, free 1915.1 MB)
19/05/01 19:31:34 INFO MemoryStore: ensureFreeSpace(30900) called with curMem=260930, maxMem=2008421498
19/05/01 19:31:34 INFO MemoryStore: Block broadcast_17_piece0 stored as bytes in memory (estimated size 30.2 KB, free 1915.1 MB)
19/05/01 19:31:34 INFO BlockManagerInfo: Added broadcast_17_piece0 in memory on localhost:39295 (size: 30.2 KB, free: 1915.3 MB)
19/05/01 19:31:34 INFO SparkContext: Created broadcast 17 from broadcast at DAGScheduler.scala:874
19/05/01 19:31:34 INFO DAGScheduler: Submitting 1 missing tasks from ResultStage 16 (MapPartitionsRDD[33] at saveAsTextFile at Kmeans.java:66
)
19/05/01 19:31:34 INFO TaskSchedulerImpl: Adding task set 16.0 with 1 tasks
19/05/01 19:31:34 INFO TaskSetManager: Starting task 0.0 in stage 16.0 (TID 16, localhost, PROCESS_LOCAL, 1583 bytes)
19/05/01 19:31:34 INFO Executor: Running task 0.0 in stage 16.0 (TID 16)
19/05/01 19:31:34 INFO deprecation: mapred.output.dir is deprecated. Instead, use mapreduce.output.fileoutputformat.outputdir
19/05/01 19:31:34 INFO deprecation: mapred.output.key.class is deprecated. Instead, use mapreduce.job.output.key.class
19/05/01 19:31:34 INFO deprecation: mapred.output.value.class is deprecated. Instead, use mapreduce.job.output.value.class
19/05/01 19:31:34 INFO deprecation: mapred.working.dir is deprecated. Instead, use mapreduce.job.working.dir
19/05/01 19:31:34 INFO FileOutputCommitter: Saved output of task 'attempt_201905011931_0016_m_000000_16' to hdfs://localhost:9000/user/amrnas
r/output/_temporary/0/task_201905011931_0016_m_000000
19/05/01 19:31:34 INFO SparkHadoopMapRedUtil: attempt_201905011931_0016_m_000000_16: Committed
19/05/01 19:31:34 INFO Executor: Finished task 0.0 in stage 16.0 (TID 16). 620 bytes result sent to driver
19/05/01 19:31:34 INFO TaskSetManager: Finished task 0.0 in stage 16.0 (TID 16) in 169 ms on localhost (1/1)
19/05/01 19:31:34 INFO TaskSchedulerImpl: Removed TaskSet 16.0, whose tasks have all completed, from pool
19/05/01 19:31:34 INFO DAGScheduler: ResultStage 16 (saveAsTextFile at Kmeans.java:66) finished in 0.169 s
19/05/01 19:31:34 INFO DAGScheduler: Job 9 finished: saveAsTextFile at Kmeans.java:66, took 0.183949 s
19/05/01 19:31:34 INFO SparkContext: Invoking stop() from shutdown hook
19/05/01 19:31:35 INFO SparkUI: Stopped Spark web UI at http://192.168.2.102:4040
19/05/01 19:31:35 INFO DAGScheduler: Stopping DAGScheduler
19/05/01 19:31:35 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
19/05/01 19:31:35 INFO Utils: path = /tmp/spark-44139ba5-0ce9-4a5a-baab-770c33f9de5b/blockmgr-b03d3fa1-a8aa-4d92-b434-240a485a670d, already p
resent as root for deletion.
19/05/01 19:31:35 INFO MemoryStore: MemoryStore cleared
19/05/01 19:31:35 INFO BlockManager: BlockManager stopped
19/05/01 19:31:35 INFO BlockManagerMaster: BlockManagerMaster stopped
19/05/01 19:31:35 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
19/05/01 19:31:35 INFO SparkContext: Successfully stopped SparkContext
19/05/01 19:31:35 INFO Utils: Shutdown hook called
19/05/01 19:31:35 INFO Utils: Deleting directory /tmp/spark-44139ba5-0ce9-4a5a-baab-770c33f9de5b
19/05/01 19:31:35 INFO RemoteActorRefProvider$RemotingTerminator: Shutting down remote daemon.
19/05/01 19:31:35 INFO RemoteActorRefProvider$RemotingTerminator: Remote daemon shut down; proceeding with flushing remote transports.
amrnasr@amrnasr-Lenovo-Y50-70:~/MyFiles/projects/distributedSystems/kmeans/target$ n.
```



```
(6.85, 3.0736842105263147, 5.742105263157893, 2.071052631578947)
(5.005999999999999, 3.4180000000000006, 1.4640000000000002, 0.2439999999999999)
(5.901612903225807, 2.748387096774194, 4.393548387096774, 1.4338709677419355)
```

Comparison Results

Measure	Sklearn	Hadoop	Spark
F-Measure	0.6818495514147688	0.6818495514147688	0.6818495514147688
Conditional Entropy	0.27302119105777406	0.27302119105777406	0.27302119105777406
Run time (sec)	0.11899614334106445	9.114	0.365