UNIVERSITÀ
DEGLI STUDI
FIRENZE

K-Means Clustering with Hadoop MapReduce

A.A. 2019-2020

Lorenzo Pratesi Mariti

UNIVERSITÀ DEGLI STUDI FIRENZE

Table of contents

Introduction

- As data and work grow, it takes a longer time to produce results.

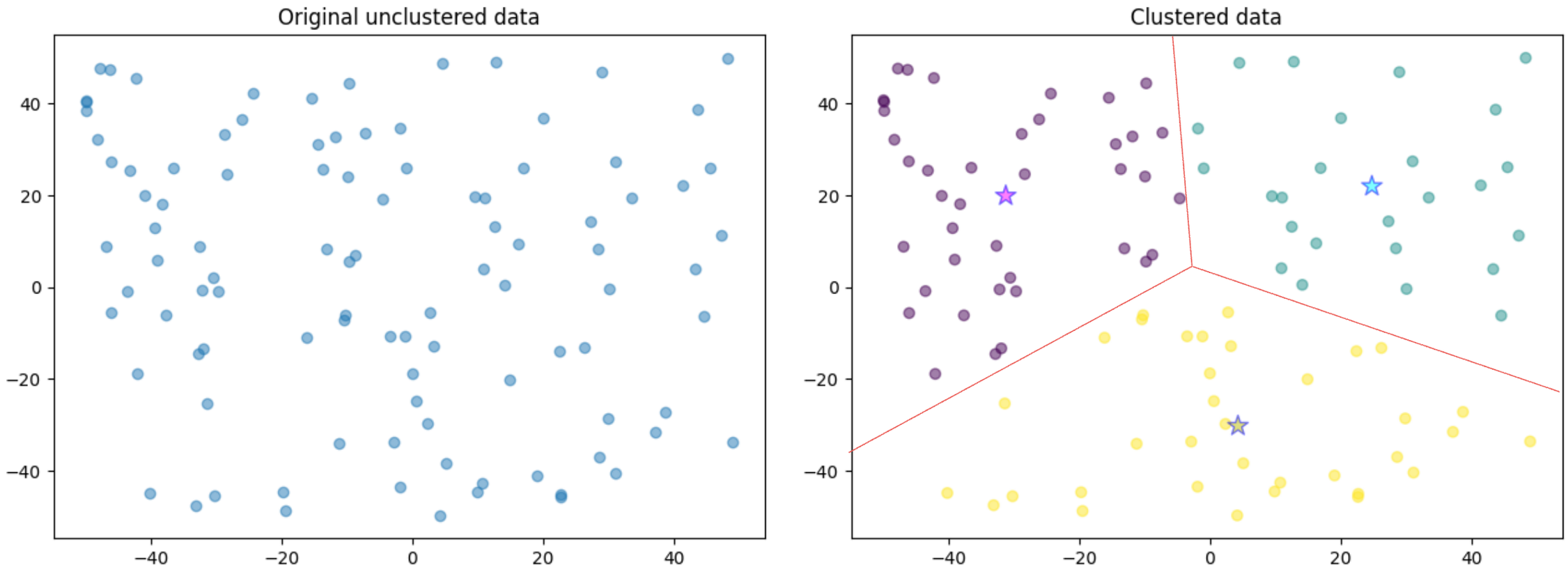
- A certain way is carried out to spread the work across many computers i.e. one needs to scale out.

- MapReduce is a programming model which is designed for processing large volumes of data in parallel.

- In this presentation, we focus on *k-means,* one of the most popular clustering algorithms using MapReduce in Hadoop.

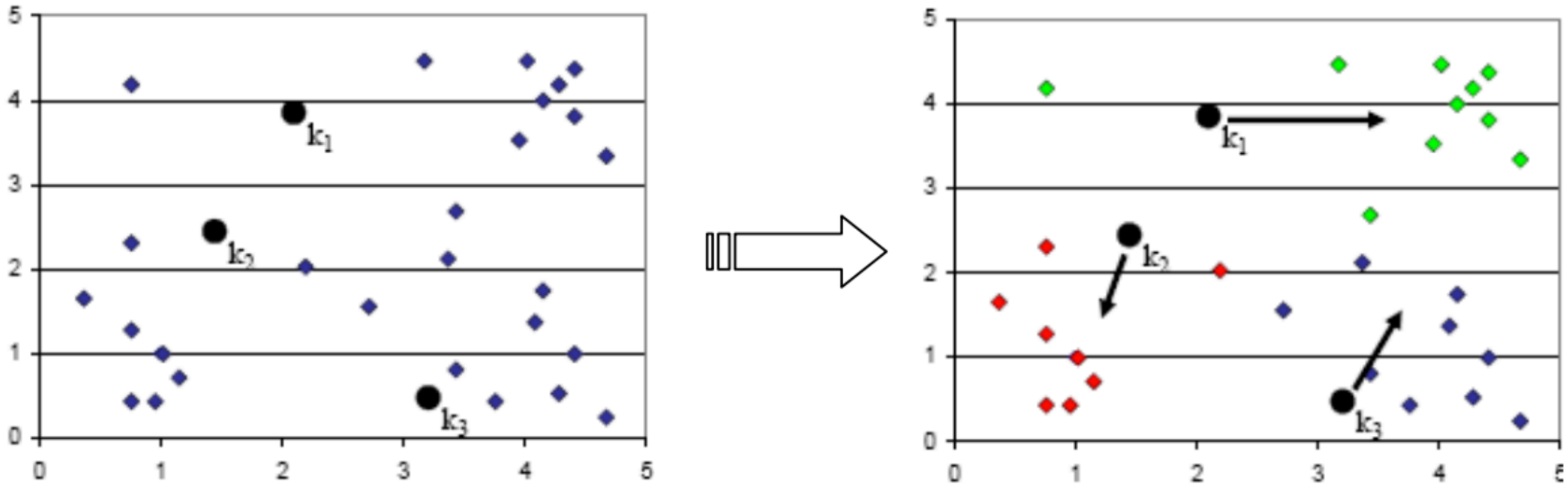
What is K-Means clustering?

k-means clustering aims to partition *n* observation into *k* clusters ($k < n$) in which each observation belongs to the cluster with the nearest mean.



Works for *n*-dimensional space as well.

What is K-Means clustering?



K-Means Algorithm

- Let $X = \{x_1, \ldots, x_n\}$ be a set of n data points, each with dimension d.
- The k-means problem seeks to find a set of k means $M = \{\mu_1, \ldots, \mu_k\}$ which minimizes the function

$$f(M) = \sum_{x \in X} \min_{\mu \in M} \|x - \mu\|_2^2$$

- We wish to choose k means so as to minimize the sum of the squared distances between each point and the mean closest to that point.

Finding an exact solution to this problem is ***NP-hard***.

K-Means Algorithm

- Each cluster is associated with a centroid (center point)
- Each point is assigned to the cluster with the closest centroid
- Number of clusters, K, must be specified
- Each iteration can be divided into two phases

1: Select K points as the initial centroids.
2: **repeat**
3: Form K clusters by assigning all points to the closest centroid.
4: Recompute the centroid of each cluster.
5: **until** The centroids don't change

K-Means Algorithm

Recall that each iteration of *k*-means can be divided into two phases:

1. Computes the sets of points closest to mean μ_i ,

Classification Step as Map

2. Computes new means as the centroids of these sets.

Recenter Step as Reduce

K-Means using MapReduce

- K-Means needs an iterative version of MapReduce
- Each iteration is implemented as a MapReduce job
- **Map**: classification step; data parallel over data point
- **Reduce**: recompute means; data parallel over centers

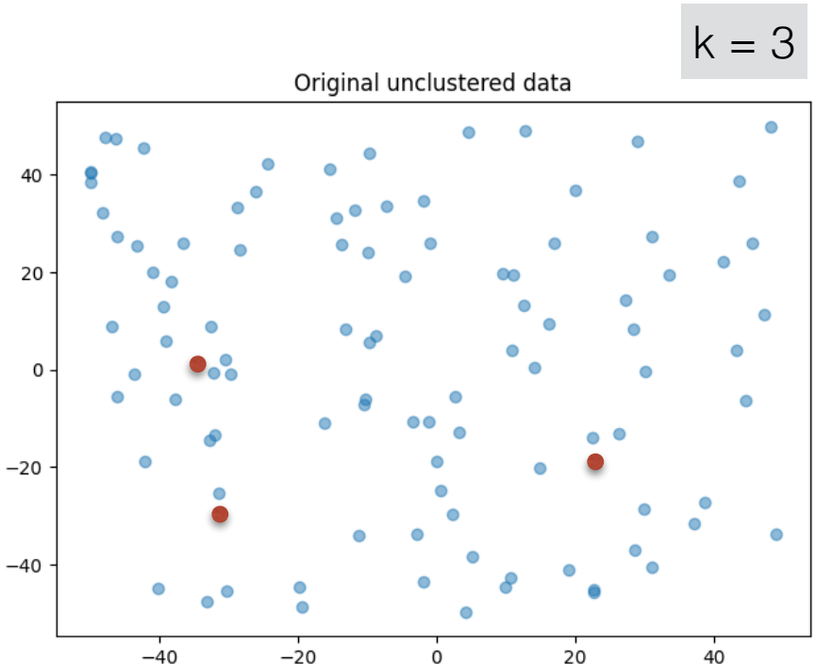
Algorithm 1: K-Means with MapReduce

```
Result: K centroids
Randomly choose K centroids;
repeat
    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;
```

UNIVERSITÀ DEGLI STUDI FIRENZE

Main Schema

INPUT	MAP	Map output



- Given k as input
- Generate k random centroids
- Broadcast the centroids to every map function

map

map

map

. . . . . .

map

key value

[centroid_1, point_1]

[centroid_2, point_3]

[centroid_3, point_10]

[centroid_2, point_45]

.....

.....

[centroid_x, point_x]

Main Schema

Map output	COMBINE	Combine output

key **values**

```
[centroid_1, point_1, point_4]

[centroid_2, point_3, point_7, point_9]

[centroid_3, point_10]

[centroid_2, point_45, point_73]

.....

.....

[centroid_x, point_x, …]
```

combine

combine

combine

.
.
.
.
.
.
.

combine

key **values**

```
[centroid_1, partialsum_1]

[centroid_2, partialsum_1]

[centroid_3, partialsum_1]

[centroid_2, partialsum_2]

.....

.....

[centroid_x, partialsum_x]
```

Main Schema

Combine output	REDUCE	OUTPUT

key **values**

```
[centroid_1, partialsum_1, …]

[centroid_2, partialsum_1, …]

[centroid_3, partialsum_1, …]
```

reduce

reduce

.
.
.
.
.
.
.

reduce

key **value**

```
[centroid_1, centroid_1’]

[centroid_2, centroid_1’]

[centroid_3, centroid_1’]
```

Implementation - Map

```
public void map(Object ikey, Text ivalue, Context context) throws IOException, InterruptedException {
    int first_comma_index = ivalue.toString().indexOf(",");
    double[] sample = getSample(ivalue.toString().substring(first_comma_index + 1));
    double min = Double.MAX_VALUE, dist;
    long index = -1;

    for (int i = 0; i < centers.size(); i++) {
        dist = EuclideanDistance.calculateDistance(sample, centers.get(i));
        if (dist < min) {
            min = dist;
            index = i;
        }
    }

    if (index != -1) {
        context.write(new LongWritable(index), new Text(ivalue.toString()));
    } else {
        logger.info("\n\nNo nearest cluster found? min = " + min + " sample = " + sample + "\n\n");
    }
}
```

Read the cluster centroids

Iterate over each cluster centroid for each input key-value pair

Compute the Euclidean distances and save the nearest center with the lowest distance to the input point

Write the key-value pair to be consumed by reducers, where the key is the nearest cluster center to the input point

Implementation - Combine

- After each map task, a combiner is applied to mix the map task's intermediate data.
- Partially sum the values of the points assigned to the same cluster.

```
public void reduce(LongWritable _key, Iterable<Text> values, Context context) throws IOException,
InterruptedException {
    double[] sum;
    double[] sample;
    int num = 0;
    sum = IntStream.range(0, nb_dimensions).mapToDouble(i -> 0.0).toArray();

    for (Text val : values) {
        int first_comma_index = val.toString().indexOf(",");
        sample = getSample(val.toString().substring(first_comma_index + 1));
        for (int i = 0; i < nb_dimensions; i++) {
            sum[i] += sample[i];
        }
        num++;
    }

    String sb = IntStream.range(0, nb_dimensions).mapToObj(i -> sum[i] + “,”)
                         .collect(Collectors.joining("", "", String.valueOf(num)));

    context.write(_key, new Text(sb));
}
```

the sum is needed to compute the mean values

The combiner function improves the efficiency of the k-means because it avoids network traffic by transferring less data between slave (i.e., worker) nodes

UNIVERSITÀ DEGLI STUDI FIRENZE

Implementation - Reduce

- The reducer recreates the centroids for all clusters by recalculating the means for all clusters
- Each reducer iterates over each value vector and calculates the mean vector.

```
public void reduce(LongWritable _key, Iterable<Text> values, Context context) throws IOException,
InterruptedException {
    double[] center;
    double[] sample;
    int num = 0, last_comma_index;
    center = IntStream.range(0, nb_dimensions).mapToDouble(i -> 0.0).toArray();

    for (Text val : values) {
        last_comma_index = val.toString().lastIndexOf(",");
        if (last_comma_index < 1) {
            logger.info("No comma found in text: " + val.toString());
            continue;
        }
        sample = getSample(val.toString().substring(0, last_comma_index));
        for (int i = 0; i < nb_dimensions; i++) {
            center[i] += sample[i];
        }
        num += Integer.parseInt(val.toString().substring(last_comma_index + 1));
    }

    StringBuilder center_sb = new StringBuilder();
    for (int i = 0; i < nb_dimensions; i++) {
        center[i] /= num;
        center_sb.append(i < nb_dimensions - 1 ? center[i] + " " : center[i]);
    }

    centers.set((int) _key.get(), center);
    context.write(_key, new Text(center_sb.toString()));
}
```

all dimensions in `center` are initialized to 0.0

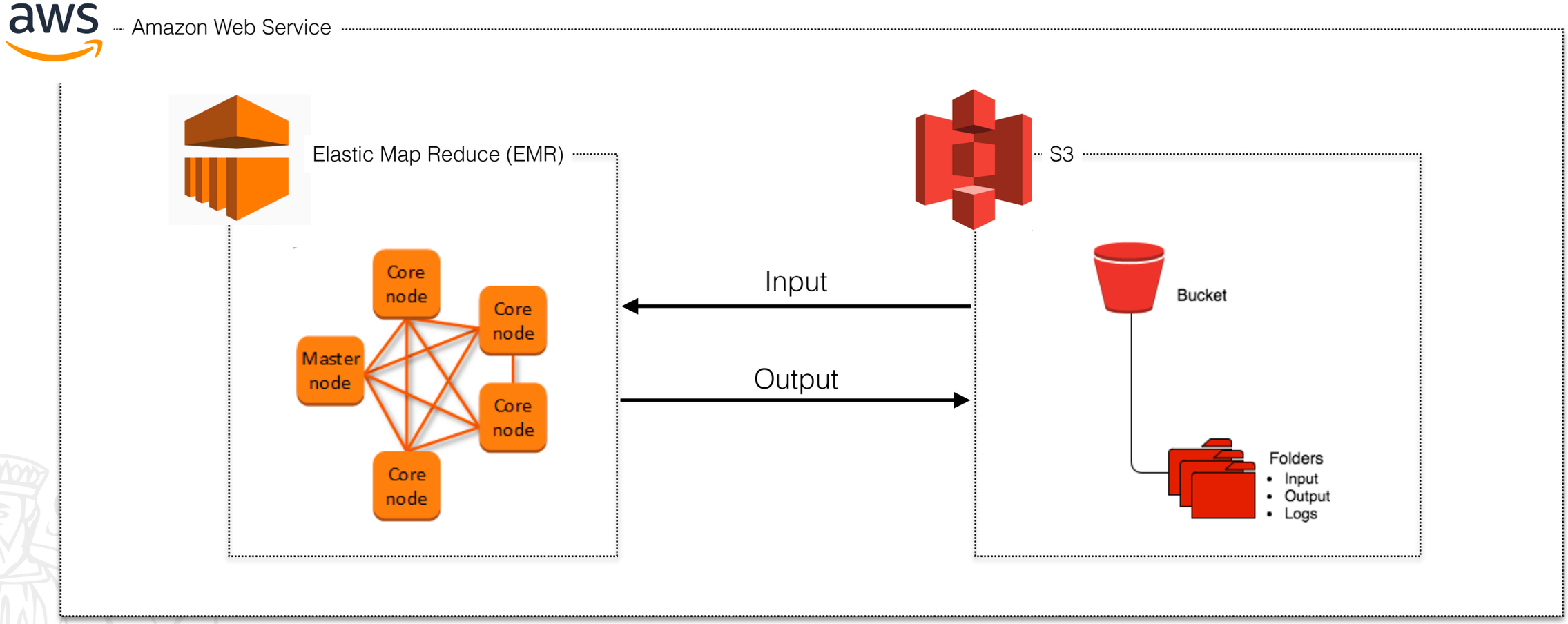
set new mean for each dimension

Once you have found the mean, this is the new center; your final step is to save it into a persistent store.

UNIVERSITÀ DEGLI STUDI FIRENZE

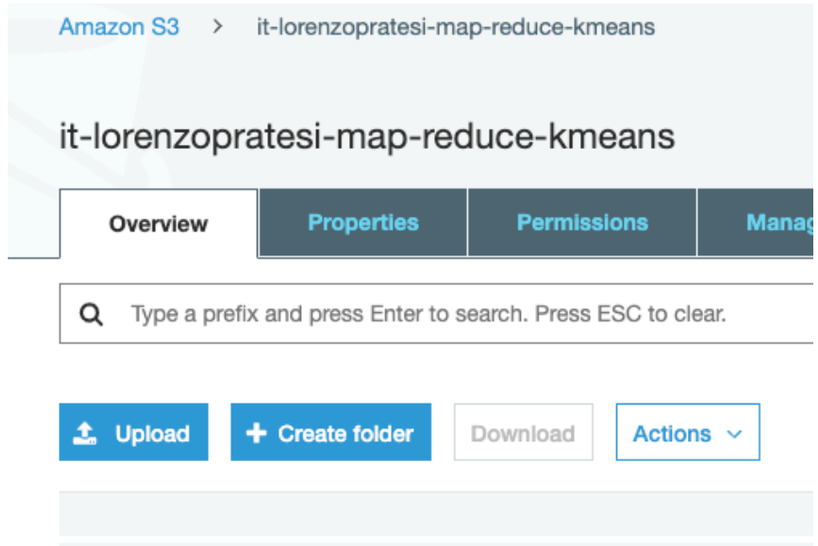
Experimental Setup

- Hadoop 2.8.5
- Cluster of machines, node number form 1 to 3
  - Each with 4 vCpu 16GB memory
- Java 1.8.0



Experimental Result

- 3 input dataset with: 100K, 300K and 500K points
- 7D points in range $(-50; 50)$
- Number of cluster $k = 5$
- Tolerance $\epsilon = 1 \times 10^{-8}$

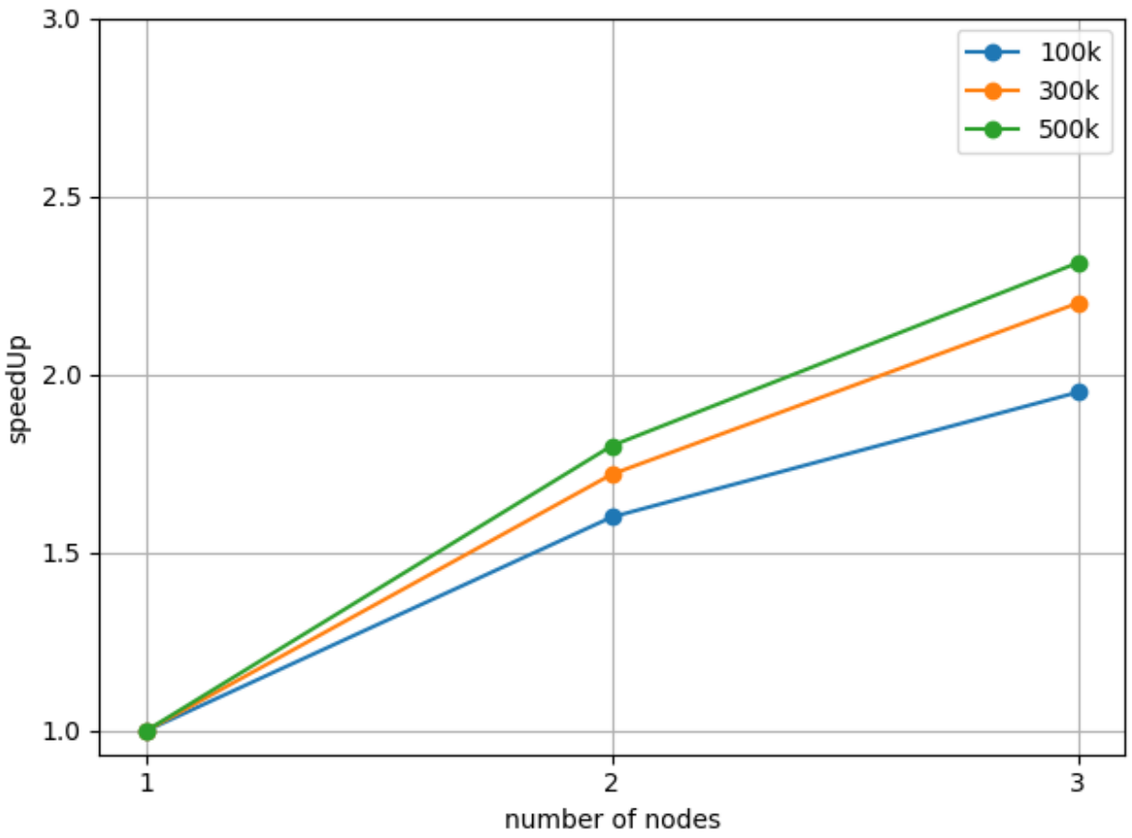


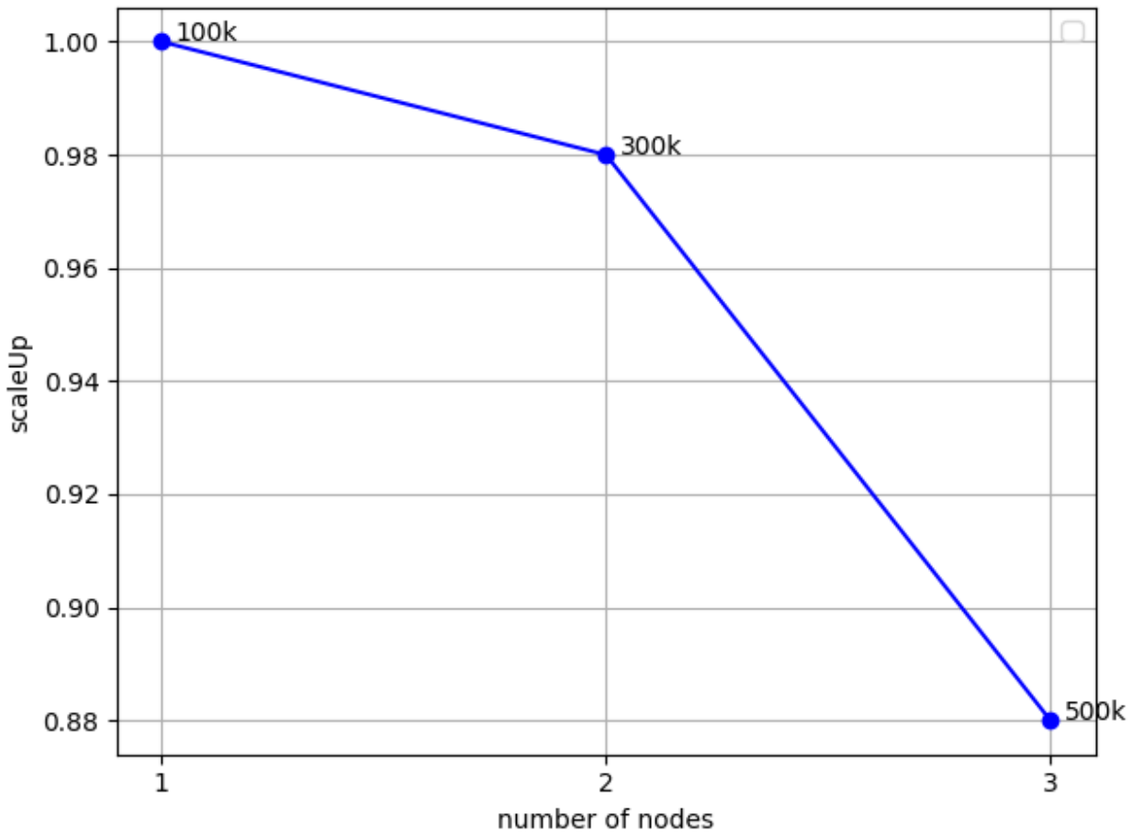
Example dataset

1	10.627546002789636	-11.100869326071404	21.82565321119715	-26.698793856240734	44.09001125216298	5.418192423753467	-5.574857659453137
2	35.814588818518274	-38.31207179847247	-41.50508883601722	-23.16486051627884	-20.10410547480703	-36.51340394999241	-48.18361883938687
3	-47.682342311352954	-35.4862913083684	8.967315527598338	32.39525297986975	21.00099270778483	48.66450635162461	-48.32346875216794
4	-16.145258260057737	-6.0061085509536625	-15.88376881551008	20.44169396659329	43.10017602618976	6.095421983025652	27.52405914114574
5	-40.77536218018473	26.893349725053184	22.755500544452516	-32.60081445080675	40.03011943545761	48.059833999261755	-49.84477108359516
6	20.8096525659423	-30.969178658071062	7.410206422392704	31.157002453028113	-8.851092088169352	-7.907400411124087	-6.566263369262437
7	36.39946239406031	46.348530442173924	-12.225923054471878	-19.625822597618004	24.429371604171934	37.965732248982036	34.80964782314004
8	-20.858049815211633	-45.751510610916924	48.08080969768196	-40.781048393118645	28.508215645231743	-43.87054182431012	-31.22453373941727
9	41.91678663386186	20.08339333167328	-49.059435592909416	7.6516726174282965	21.828457103938334	40.10911989838465	-16.702067876318964
10	17.5485459595142	-23.60751356629256	30.314641226462413	-6.624395640397637	1.171317899938927	-19.913837390234125	1.8597123370510786
11	-16.935377264150752	3.82363576826814	-28.589070066229226	-29.963639127182752	1.6116118854282178	-27.00350045811858	45.70724200252937
12	43.490204055121694	-23.8709923424139	8.672373239263173	-44.649632731855974	19.49389086852436	-37.35696364744988	43.36667682296364
13	46.76779676249026	11.641372487168347	-11.357829365784887	43.51430866109085	25.77820993670474	25.254083569379233	-21.048751533836597
14	17.08202971959419	33.15682257112033	40.254313747910786	25.911153723025038	45.562206638224154	-44.797059760272816	-9.643647238866194
15	-37.11748819885465	-27.899417873291675	-33.11621277780392	-0.13168820996381214	32.04836651535996	48.83290138155793	28.552112846654992
16	-15.858035877082742	-45.14318349917139	39.94498395404898	32.910843876528745	36.07920846172111	-13.825546146320391	16.886966287507548
17	3.5972006492749458	-10.207469481615682	-8.334300114662419	-3.51488625769165	-45.81461010416312	14.43192521805831	-17.956758915619176
18	23.90333971302138	-24.08608721734643	-20.015150869259436	13.660956035774483	35.67865722731254	6.909579630290651	-20.606726594851963
19	46.13402334529411	-41.6158936449452	9.457304726941906	-27.682312781339814	41.61300729682607	-28.51684988992369	-47.81189634652401

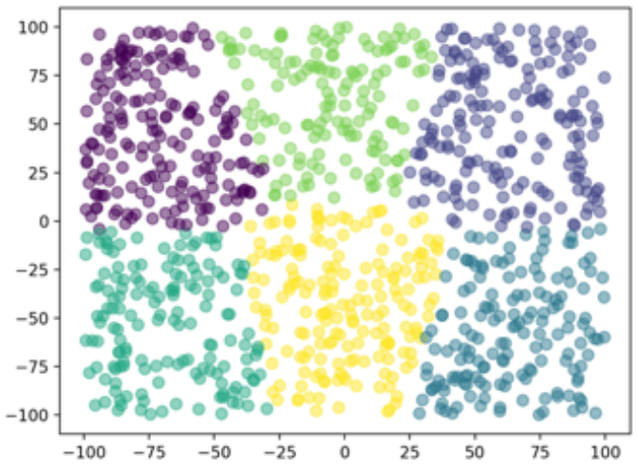
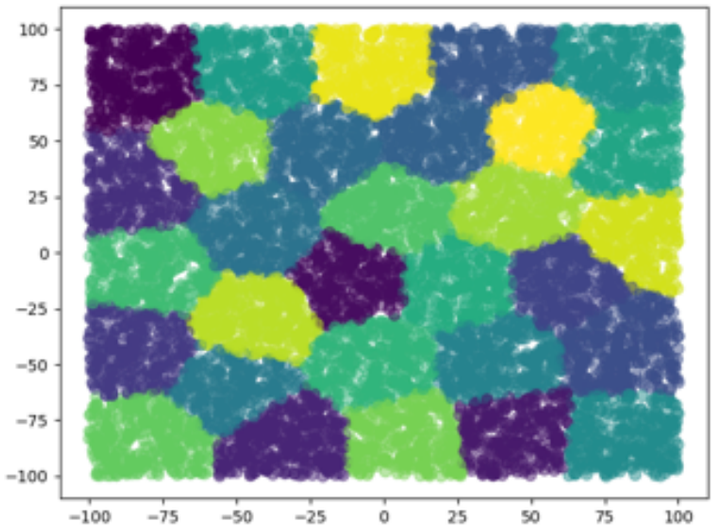
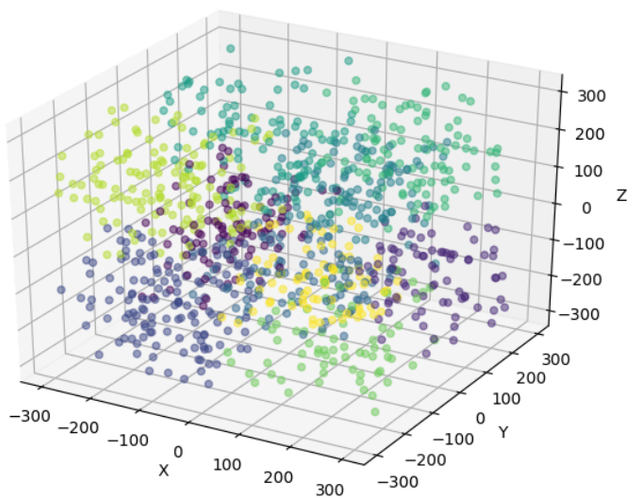
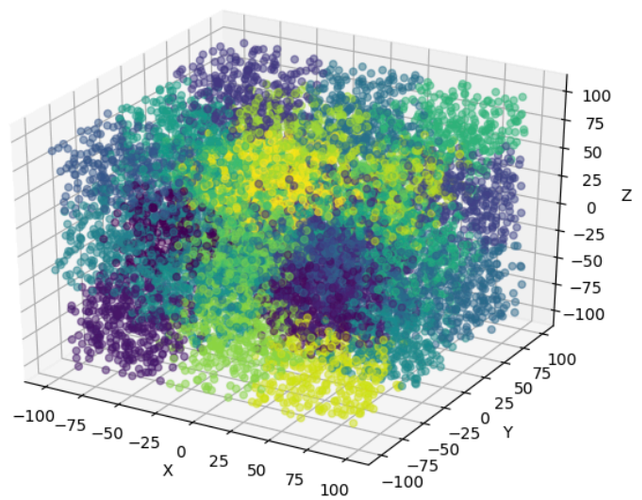
Experimental Result

- In order to correctly interpret the results, it must be taken into account that the Hadoop Framework has been conceived for:
  - A huge number of data (i.e. > 1TB)
  - Massively parallel, i.e. with systems distributed with hundreds of CPUs;
  - Fault-tolerance





Some Plots

Clustering 1000 2D point with K = 6	Clustering 10000 2D point with K = 30
	
Clustering 1000 3D point with K = 10	Clustering 10000 3D point with K = 30
	

Conclusion

- It has been shown an implementation of K-Means clustering with Apache Hadoop MapReduce
- The result shown the algorithm can process large datasets effectively
- The proposed algorithm is more efficient while clustering larger datasets than smaller.

Future work

- The proposed K-Means can be modified such that it can automatically settle on the number of cluster and effectively select initial centroids depending on the datasets.

- Proposed k-means can be combined with techniques like hierarchical clustering algorithms, in order to obtain better quality clustering.

- Hadoop provides choices to optimize on disk, memory, network and CPU. Hadoop cluster can be optimized for each particular job for more efficiency.

References

- Mahmoud Parsian - Data Algorithms: Recipes for Scaling Up with Hadoop
- Weizhong Zhao, Huifang Ma, and Qing He - Parallel K-Means Clustering Based on MapReduce
- Carlos Guestrin - Parallel Programming Map-Reduce
- Kenneth Heafield - Hadoop Design and k-Means Clustering