Alexandria University

Faculty Of Engineering

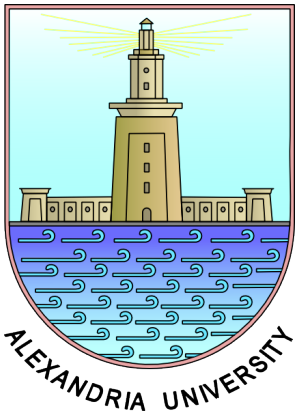
Computer and Systems Engineering Department

Parallel K-Means using Hadoop

Authors:
Aya Aboud [1]
Ramy Wagdy [20]
Mina Makram [70]

Supervisors:
Prof. Dr. Mohamed Khalefa
Eng. Samia
Eng. Reham

April 27, 2016


ALEXANDRIA UNIVERSITY

Abstract

Clustering has been used in many applications to classify data with respect to their closeness to each other. As nowadays the data becomes larger, the need for parallel clustering algorithms becomes greater. In this report, we implement and discuss how K-means clustering can be used with mapreduce framework to get a parallel classification of data.

1 Problem Definition

In this lab, it is required to implement K-means clustering algorithm with two versions: Sequential and parallel. Below are the tasks required.

1. Implementing Sequential version of K-means algorithm.
2. Implementing parallel k-means algorithm.
3. Running the parallel K-means on multiple-node cluster.
4. Running the parallel K-means on Microsoft Azure Cluster.
5. Comparing the efficiency of the parallel and nonparallel k-means clustering

2 Algorithms

2.1 Sequential K-means

Non-paralleled K-means implemented by Java language. First, read data from file and store it, then choose random K point to be center. Begin to assign points to clusters calculating the euclidean distance A point belongs to a cluster to which it has minimum Euclidean distance. Recalculate center by take average of point in every clusters. then repeat this cluster and calculate center operation until tolerance satisfy or max iteration is achieved.

2.1.1 Pseudo code

Algorithm 1 K-Mean Clusters

```
1: Get initial random k point to be centers of clusters
2: while max number of iteration not match do
3:     while next point not equal null do
4:         min-distance ← ∞
5:         while next center not equal null do
6:             if Euclidian distance is smaller than min-distance then
7:                 min-distance ← Euclidian distance
8:             end if
9:         end while
10:        insert point to cluster with min Euclidian distance
11:    end while
12:    calculate new center by take Average of point
13:    if no change in center then
14:        go to line 16
15:    end if
16: end while
```

2.1.2 Time Complexity

The time complexity for nonparallel k-means is ***O(IKND)*** where:

- **I** is the number of iterations.
- **K** is the number of clusters.
- **N** is the number of points.
- **D** is the dimension of the point.

2.1.3 Space Complexity

The Space complexity for nonparallel k-means is ***O((N+K)D)***

2.2 Parallel K-means

2.2.1 Pseudo code

The pseudo codes of the mapper and reducer of parallel K-means are shown in Algorithms 2 and 3 respectively.

Algorithm 2 Parallel K-means Mapper

```
1: procedure SETUP(context)
2:     Set initial random k point to be centers of clusters
3:     Define EPSILON ← 0.1.
4: end procedure
5:
6: procedure MAP(key, value)
7:     dataSize++;
8:     tokenizer = Tokenize(value).
9:     define IntWritable[] point;
10:    point = tokenizer.toInt();
11:    define euclideanDistance;
12:    minimum ← ∞;
13:    for  i = 0; i ¡ centers.length; i++ do
14:        euclideanDistance ← 0
15:        for  col ← 0 to dimension do
16:            difference = point[col].get() - centers[i][col].get();
17:            euclideanDistance += Math.pow(difference, 2);
18:        end for
19:        if minimum > SQRT(euclideanDistance) then
20:            minimum ← Math.sqrt(euclideanDistance)
21:            cluster = i
22:        end if
23:    end for
        Emit(cluster, point);
24: end procedure
25:
```

Algorithm 3 Parallel K-means Reducer

```
1: procedure REDUCE(key, Iterable value)
2:     counter ← 0
3:     define newCentroid[dimension];
4:     sum[dimension];
5:     for val : values do
6:         counter++;
7:         for d ← 0 to dimension do
8:             sum[d] += val.get() [d];
9:             d++;
10:        end for
11:        d = 0;
12:        while ( dod ¡ dimension)
13:            newDimensionCentroid = sum[d]/counter;
14:            newCentroid[d] = newDimensionCentroid;
15:            d++;
16:        end while
17:    end for
18:    isCentroidsStable = true;
19:    for (i ← 0 to dimension) do
20:        if  Abs(centers[key.get()][i].get() - newCentroid[i].get()) > EPS
   then
21:            isCentroidsStable = false;
22:            centers[key.get()][i] = newCentroid[i]
23:        end if
24:        if (isCentroidsStable = true) then
25:            isJobDone ← true;
26:        else
27:            isJobDone = false;
28:        end if
29:    end for
30:    Emit (key, (counter / dataSize) * 100 )
31:    Emit(cluster, point);
32: end procedure
```

2.2.2 Time Complexity

The time complexity for Parallel k-means is ***O(IKND / M)*** where:

- **M** is the number of the mappers.

2.2.3 Space Complexity

The Space complexity for Parallel k-means is ***O((N+K)D)***

3 Implementation

3.1 Environment

- Java Programming was used.
- Hadoop Package is used.
- Linux Platform is used.

3.2 Machines Specifications

- Architecture: x86_64
- CPU op-mode(s): 32-bit, 64-bit
- Byte Order: Little Endian
- CPU(s): 4
- On-line CPU(s) list: 0-3
- Thread(s) per core:2
- Core(s) per socket:2
- Socket(s): 1
- NUMA node(s): 1
- Vendor ID: GenuineIntel
- CPU family: 6

- Model: 42
- Stepping: 7
- CPU MHz: 1696.093

3.3 Network Type

Wireless Local Area Network was used to test multiple machines.

3.4 Data sets

We used the ***Census Data (1990)*** Data Set. It has the following specifications:

- **Number of Instances**: 2458285
- **Number of Attributes**: 68

4 Results

4.1 Non-paralleled K-means

Table 1: Unparalleled k-Cluster = 5

	Percentage
Cluster 1	64.10 %
Cluster 2	25.03 %
Cluster 3	3.64 %
Cluster 4	0.00 %
Cluster 5	7.23 %
Time	44.4 seconds
Iterations	10

Table 2: Unparalleled k-Cluster = 10

	Percentage
Cluster 1	6.40 %
Cluster 2	6.05 %
Cluster 3	10.04 %
Cluster 4	10.09 %
Cluster 5	7.425 %
Cluster 6	0.84 %
Cluster 7	24.65 %
Cluster 8	25.66 %
Cluster 9	1.64 %
Cluster 10	7.20 %
Time	67.988 seconds
Iterations	13

Table 3: Unparalleled k-Cluster = 15

	Percentage
Cluster 1	9.54 %
Cluster 2	3.62 %
Cluster 3	5.82 %
Cluster 4	1.996 %
Cluster 5	7.10 %
Cluster 6	8.985 %
Cluster 7	13.012 %
Cluster 8	7.345 %
Cluster 9	6.3 %
Cluster 10	7.15 %
Cluster 11	6.05 %
Cluster 12	9.14 %
Cluster 13	5.43 %
Cluster 14	4.90 %
Cluster 15	3.51 %
Time	278.564 seconds
Iterations	60

Table 4: Unparalleled k-Cluster = 20

	Percentage
Cluster 1	4.69 %
Cluster 2	3.22 %
Cluster 3	1.89 %
Cluster 4	15.23 %
Cluster 5	1.89 %
Cluster 6	5.81 %
Cluster 7	2.31 %
Cluster 8	2.45 %
Cluster 9	3.78 %
Cluster 10	22.32 %
Cluster 11	3.36 %
Cluster 12	5.73 %
Cluster 13	4.74 %
Cluster 14	0.81 %
Cluster 15	2.27 %
Cluster 16	7.58 %
Cluster 17	2.86 %
Cluster 18	2.93 %
Cluster 19	2.44 %
Cluster 20	3.67 %
Time	694 seconds
Iterations	25

Figure 1: Relation between k mean and time running in non-parallel K-mean

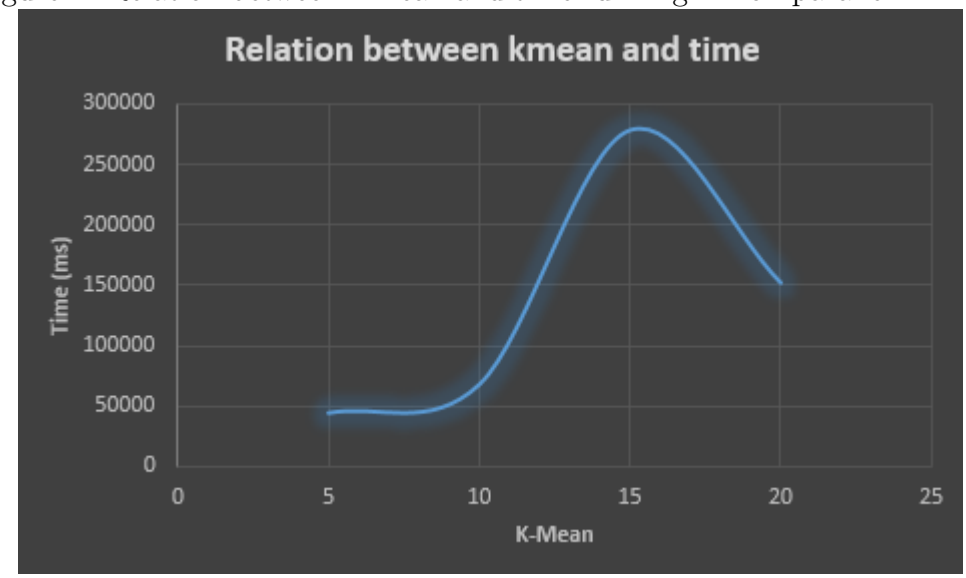
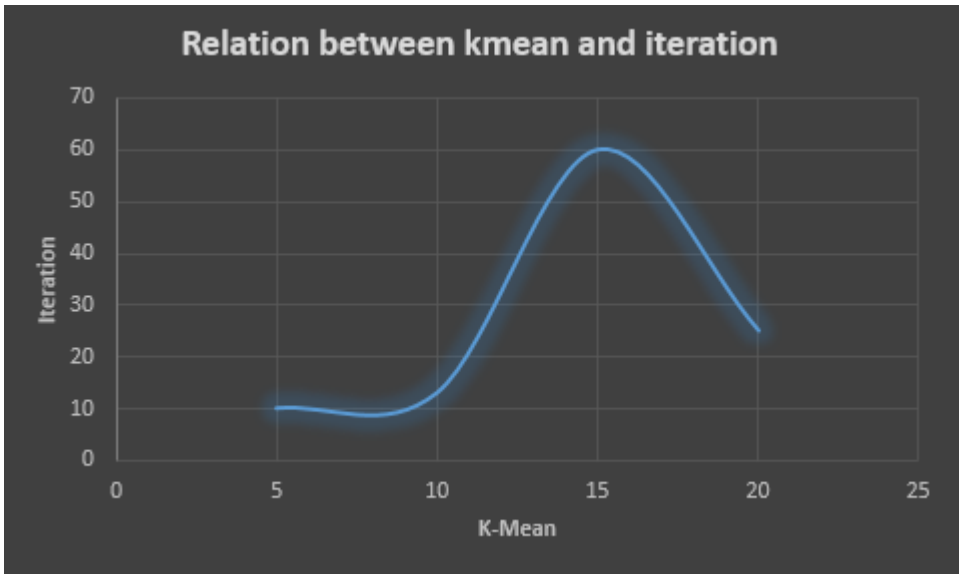


Figure 2: Relation between number of clusters and number of iteration non-parallel K-mean



4.2 Parallel K-means

4.2.1 Single Node

Table 5: Single Node k-Cluster = 5

	Percentage
Cluster 1	32.015 %
Cluster 2	6.05 %
Cluster 3	42.95 %
Cluster 4	11.76 %
Cluster 5	7.22 %
Time	261 seconds
Iterations	2

Table 6: Single Node k-Cluster = 10

	Percentage
Cluster 1	15.29 %
Cluster 2	6.75 %
Cluster 3	23.15 %
Cluster 4	18.13 %
Cluster 5	6.05 %
Cluster 6	0.85 %
Cluster 7	14.8 %
Cluster 8	4.02 %
Cluster 9	10.13 %
Cluster 10	0.84 %
Time	465 seconds
Iterations	4

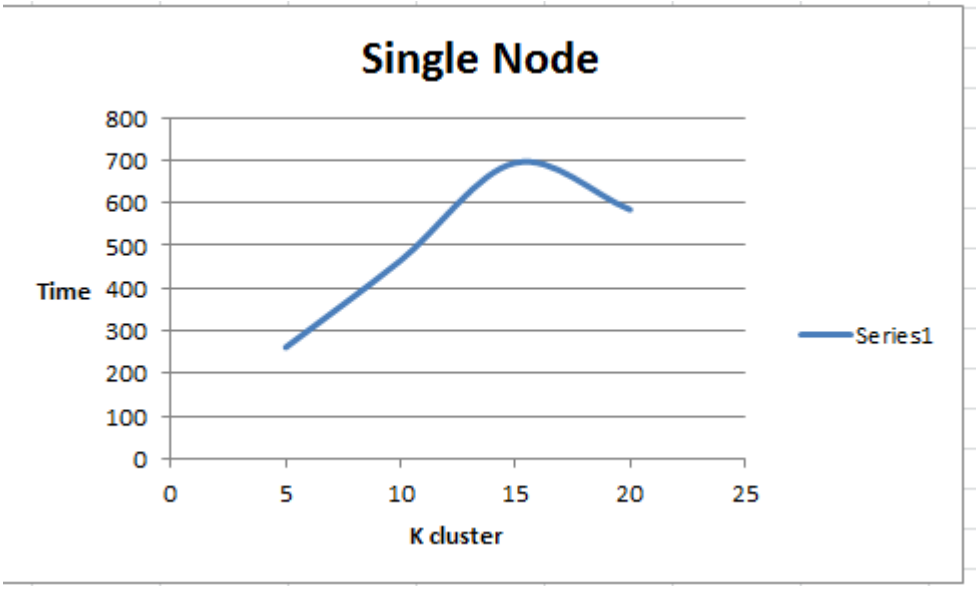
Table 7: Single Node k-Cluster = 15

	Percentage
Cluster 1	0.76 %
Cluster 2	6.75 %
Cluster 3	8.43 %
Cluster 4	9.19 %
Cluster 5	2.02 %
Cluster 6	7.04 %
Cluster 7	6.05 %
Cluster 8	15.23 %
Cluster 9	3.22 %
Cluster 10	12.02 %
Cluster 11	2.32 %
Cluster 12	7.23 %
Cluster 13	3.75 %
Cluster 14	7.09 %
Cluster 15	8.89 %
Time	694 seconds
Iterations	6

Table 8: Single Node k-Cluster = 20

	Percentage
Cluster 1	12.54 %
Cluster 2	0.9 %
Cluster 3	1.09 %
Cluster 4	7.56 %
Cluster 5	2.41 %
Cluster 6	1.77 %
Cluster 7	2.28 %
Cluster 8	2.24 %
Cluster 9	1.28 %
Cluster 10	2.46 %
Cluster 11	0.97 %
Cluster 12	2.3 %
Cluster 13	6.64 %
Cluster 14	4.52 %
Cluster 15	25.03 %
Cluster 16	6.75 %
Cluster 17	2.24 %
Cluster 18	8.53 %
Cluster 19	5.94 %
Cluster 20	2.54 %
Time	584 seconds
Iterations	5

Figure 3: Relation between k mean and time running in single node parallel K-mean



4.2.2 Multiple Node VMs

Table 9: Multiple Node k-Cluster = 5

	Percentage
Cluster 1	67.737 %
Cluster 2	9.0515 %
Cluster 3	0.0000 %
Cluster 4	15.978 %
Cluster 5	7.2322 %
Time	351 seconds
Iterations	3

Figure 4: Clustering Percetage for k = 5

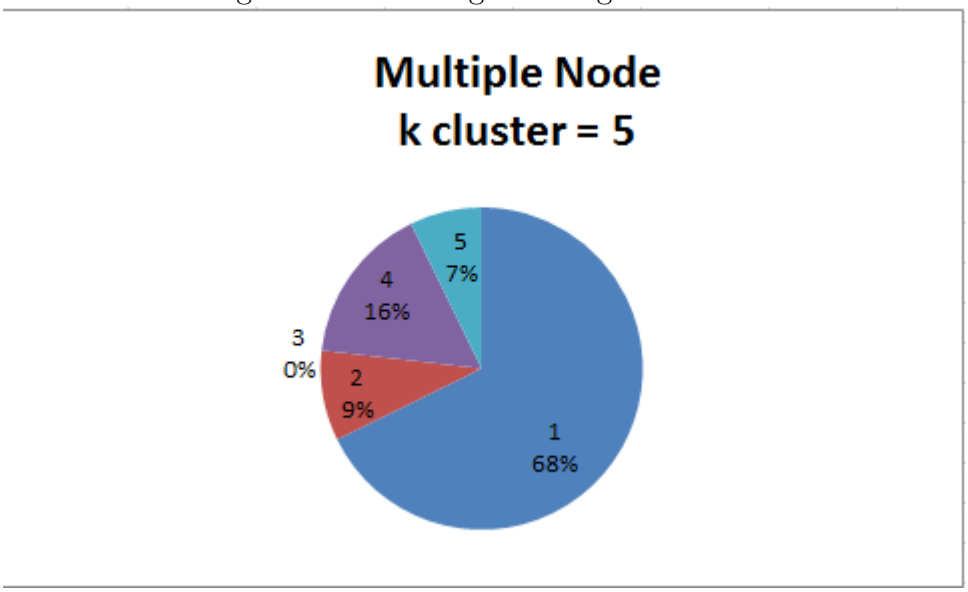


Table 10: Multiple Node k-Cluster = 10

	Percentage
Cluster 1	8.189 %
Cluster 2	7.323 %
Cluster 3	7.190 %
Cluster 4	21.242 %
Cluster 5	3.322 %
Cluster 6	5.756 %
Cluster 7	3.042 %
Cluster 8	9.209 %
Cluster 9	25.029 %
Cluster 10	9.693 %
Time	894 seconds
Iterations	9

Figure 5: Clustering Percentage for k = 10

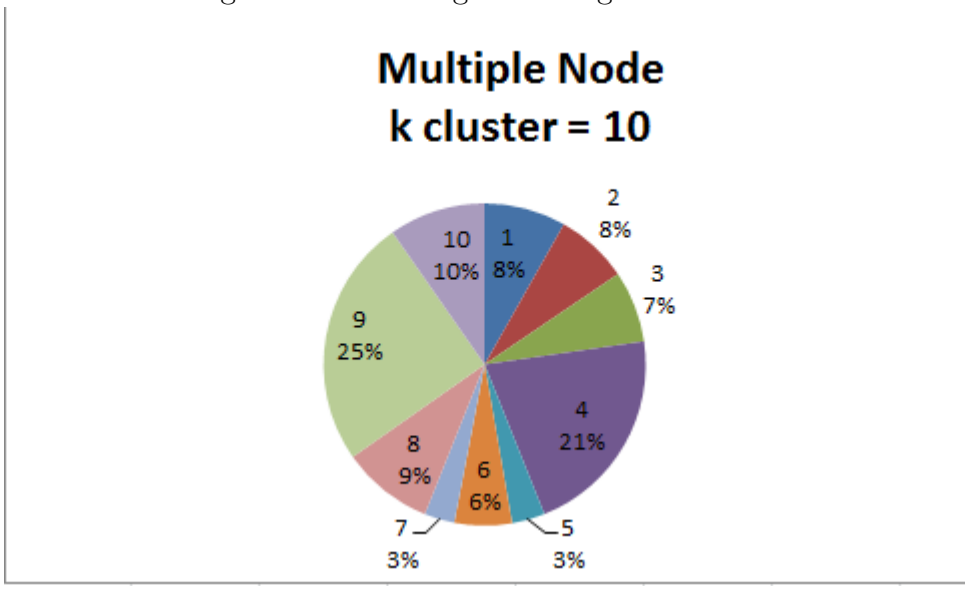


Table 11: Multiple Node k-Cluster = 15

	Percentage
Cluster 1	18.499 %
Cluster 2	3.002 %
Cluster 3	3.784 %
Cluster 4	7.199 %
Cluster 5	5.191 %
Cluster 6	2.265 %
Cluster 7	6.207 %
Cluster 8	9.142 %
Cluster 9	6.323 %
Cluster 10	15.231 %
Cluster 11	5.270 %
Cluster 12	1.854 %
Cluster 13	6.083 %
Cluster 14	6.194 %
Cluster 15	3.748 %
Time	521 seconds
Iterations	5

Figure 6: Clustering Percentage for k = 15

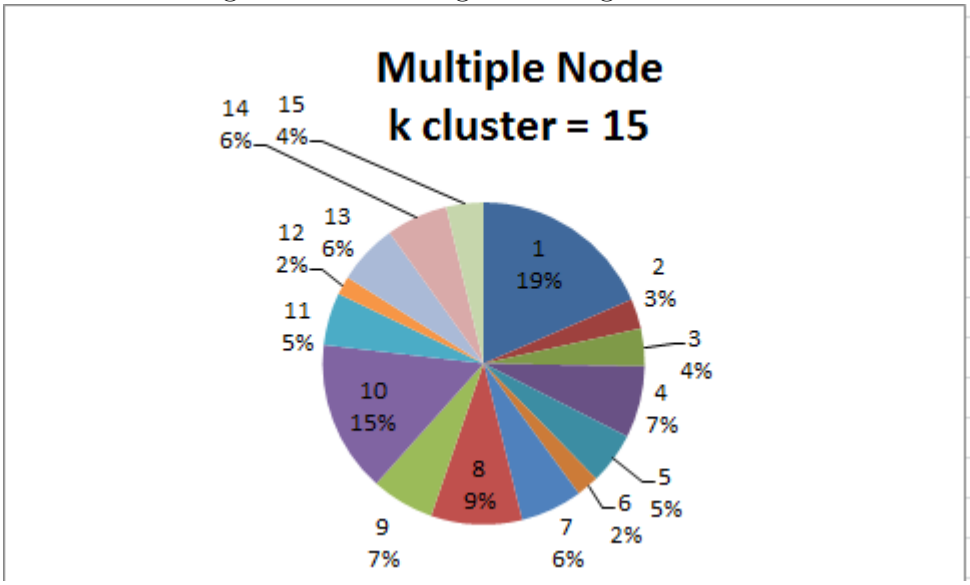
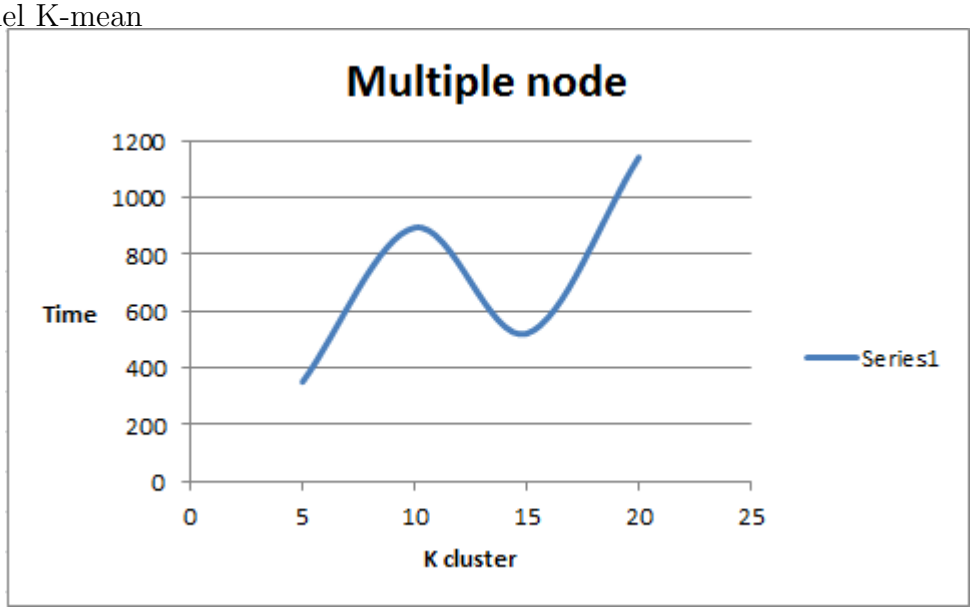


Table 12: Multiple Node k-Cluster = 20

	Percentage
Cluster 1	9.307 %
Cluster 2	5.085 %
Cluster 3	19.417 %
Cluster 4	3.051 %
Cluster 5	1.590 %
Cluster 6	1.729 %
Cluster 7	4.503 %
Cluster 8	2.971 %
Cluster 9	3.393 %
Cluster 10	3.152 %
Cluster 11	1.770 %
Cluster 12	6.049 %
Cluster 13	1.635 %
Cluster 14	5.717 %
Cluster 15	2.684 %
Cluster 16	7.023 %
Cluster 17	2.441 %
Cluster 18	1.153 %
Cluster 19	13.857 %
Cluster 20	3.463 %
Time	1144 seconds
Iterations	11

Figure 7: Relation between k mean and time running in Multiple node parallel K-mean



4.3 Performance

Table 13: k Cluster = 5

	Unparallel	Single Node	Multiple Node
Time	33.831	261	351
Iteration	3	2	3

Table 14: k Cluster = 10

	Unparallel	Single Node	Multiple Node
Time	49.456	465	894
Iteration	8	4	9

Table 15: k Cluster = 15

	Unparallel	Single Node	Multiple Node
Time	73.933	694	521
Iteration	10	6	5

Table 16: k Cluster = 20

	Unparallel	Single Node	Multiple Node
Time	121.307	584	1144
Iteration	17	5	11

5 Conclusion

- From our sample runs, we have concluded that the time of multiple node cluster is greater than the single node cluster due to the overhead communications between nodes in the multiple machines.
- The time of each run doesn't depend on the number of clusters but on the iterations.
- There will be more iterations and the results are more accurately if the ϵ is smaller.