K-means Clustering: Algorithm and Implementation

David Chonev

ACM Reference Format:
David Chonev. 2024. K-means Clustering: Algorithm and Implementation. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

K-means clustering is a simple yet powerful algorithm used to partition a dataset into k clusters, where each data point belongs to the cluster with the nearest mean. This technique is widely used in various applications such as market segmentation, image compression, and anomaly detection.

2 GENERATION OF SITES

The initial sites (centroids) for the K-means algorithm can be generated using the initial dataset. If the number of sites to be generated exceeds the available data points, new sites are generated within the geographical bounds of Europe, ensuring they fall on land and not water. This is achieved by generating random latitude and longitude points within a polygonal boundary that encompasses Europe.

3 IMPLEMENTATION

The K-means algorithm can be implemented using three main functions: calculating new centers, assigning sites to new centers, and checking cluster stability. Calculating new centers involves averaging the coordinates (latitude and longitude) of all sites in a cluster to update the centroid's position, ensuring the centroid better represents the cluster's center. Assigning sites to new centers reassigns each site to the nearest centroid based on distance, forming more accurate groupings. Checking cluster stability compares the current and previous centroids to determine if they have stabilized, signaling the algorithm to stop when clusters no longer change significantly. These functions work iteratively to refine clusters and improve their accuracy.

3.1 Calculate New Centers

The new center for each cluster is calculated by averaging the latitude and longitude of all sites assigned to the cluster. This step ensures that the centroid of each cluster moves to the center of the sites that belong to it, which helps in minimizing the within-cluster variance.

For each $C_i \in$ clusters do:
  sumLatitude = 0
  sumLongitude = 0
  numSites = 0
  For each $S_j \in$ sites do:
    If S_j.getCluster() = C_i then:
      sumLatitude+ = S_j.getLa()
      sumLongitude+ = S_j.getLo()
      numSites+ = 1
  If numSites $\neq$ 0 then:
    C_i.setLa(sumLatitude / numSites)
    C_i.setLo(sumLongitude / numSites)

3.2 Assign Sites to New Centers

Each site is assigned to the nearest cluster center based on the minimum distance. This step is essential for ensuring that each site is grouped with the most appropriate cluster center, helping to form well-defined clusters.

For each $S_j \in$ sites do:
  minDistance = ∞
  closestCluster = null
  For each $C_i \in$ clusters do:
    distance = $\sqrt{(S_j.\text{getLa}() - C_i.\text{getLa}())^2 + (S_j.\text{getLo}() - C_i.\text{getLo}())^2}$
    If distance < minDistance then:
      minDistance = distance
      closestCluster = C_i
  S_j.setCluster(closestCluster)

3.3 Check Cluster Stability

To determine if the clusters have stabilized, we compare the centroids of the new clusters with the old clusters. This step is crucial for determining convergence; if the centroids do not change, the algorithm has converged and the clustering is considered stable.

For each $i \in [0, n)$ do:
  distance = $\sqrt{(\text{newClusters}[i].\text{getLa}() - \text{oldClusters}[i].\text{getLa}())^2 + (\text{newClusters}[i].\text{getLo}() - \text{oldClusters}[i].\text{getLo}())^2}$
  If distance > 0 then return false;
return true;

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

3.4 Sequential Version

The sequential version of the K-means algorithm follows the basic implementation where all steps are performed in a single sequence on one processor. The process involves initializing cluster centers, iteratively assigning sites to the nearest cluster, calculating new cluster centers, and checking for convergence until the centroids stabilize.

Initialize cluster centers randomly
Repeat until convergence:
  Calculate new cluster centers
  Assign sites to nearest cluster centers
  Check if clusters have stabilized

3.5 Parallel Version

The parallel version of the K-means algorithm uses available processors to speed up the computations. The dataset is divided among the available processors, which independently assign sites to clusters and calculate new centers. This approach utilizes the computational power of multiple cores to achieve faster convergence.

Initialize cluster centers randomly
Divide dataset among available processors
Repeat until convergence
  Each processor calculates new cluster centers
  Each processor assigns its sites to nearest cluster centers
  Combine results from all processors

3.6 Distributed Version

The distributed version of the K-means algorithm is designed to work across multiple nodes in a distributed system. The root processor initializes the dataset and distributes it to all processors. Each processor independently assigns sites to clusters, calculates new centers, and checks for convergence. The results are combined and redistributed in each iteration until the clusters stabilize.

Root processor initializes dataset and assigns clusters
Distribute dataset to all processors
Repeat until convergence:
  Each processor calculates new centers for its cluster subset
  Combine new centers and redistribute to processors
  Each processor assigns sites to clusters for it's sites portion
  Combine updated site data and redistribute
  Each processor checks for cluster convergence in its subset
  If any processor detects changes, continue

4 TESTING

4.1 Sites testing

The performance of the K-means algorithm was tested with varying numbers of sites to compare the execution time across different implementation modes: sequential, parallel, and distributed. The results, as illustrated in Figure 1, reveal distinct patterns in the average time per cycle for each mode.

Initially, all modes perform similarly with low execution times. However, as the number of sites increases, the parallel version consistently outperforms the others, showing a steady increase in execution time. This efficiency is due to the parallel version's ability to leverage multiple processors effectively, distributing the workload evenly and reducing the overall computation time.

In contrast, the sequential and distributed modes remain close in performance, with the distributed version being the slowest. The increased overhead in communication and synchronization between processors in the distributed mode contributes to this inefficiency. Despite utilizing more cores, the distributed version does not achieve better performance due to the uneven workload distribution among processors. For some processors, the task of comparing cluster subportions with all sites becomes significantly heavier, resulting in longer execution times.

These findings highlight that while parallel processing can significantly optimize performance, distributing tasks across multiple processors does not always yield better results. The balance between computation and communication overhead plays a critical role in the performance of distributed systems.

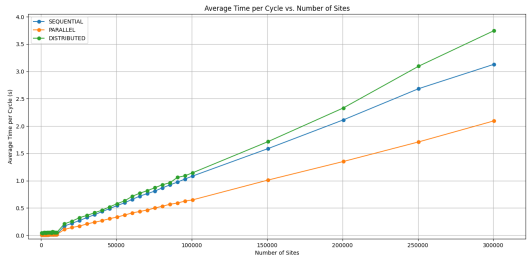

Figure 1: Average Time per Cycle vs. Number of Sites for Sequential, Parallel, and Distributed Modes

4.2 Cluster Testing

The performance of the K-means algorithm was also evaluated by varying the number of clusters. The results are depicted in Figure 2. Initially, all implementation modes—sequential, parallel, and distributed—exhibit similar performance, with negligible differences in execution time.

However, as the number of clusters increases, significant differences in performance emerge. Around the threshold of 100 clusters, the sequential version becomes the slowest among the three modes. The distributed version, while slightly faster than the sequential implementation, still lags behind the parallel version. The parallel implementation consistently demonstrates superior performance, maintaining a lower execution time as the number of clusters increases.

The slower performance of the distributed version can be attributed to the overhead associated with communication and synchronization between processors. Although distributing the workload can balance the computational effort, the coordination required

among processors introduces delays. This contrasts with the parallel implementation, which benefits from shared memory and efficient multi-threading, resulting in faster computations.

These findings reinforce the observation that the parallel version of the K-means algorithm is the most efficient for handling larger numbers of clusters. The distributed version, despite utilizing multiple processors, incurs additional overheads that hinder its performance compared to the parallel implementation.

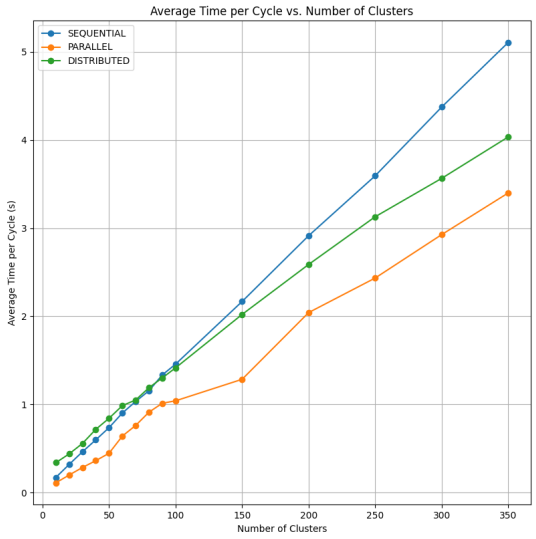

Figure 2: Average Time per Cycle vs. Number of Clusters for Sequential, Parallel, and Distributed Modes

5 SUMMARY

This project explored the implementation and performance of the K-means clustering algorithm across three different modes: sequential, parallel, and distributed. The core functions of the K-means algorithm—calculating new centers, assigning sites to clusters, and checking cluster stability—were implemented and analyzed in these modes.

The sequential version performs all computations in a single sequence on one processor. This straightforward implementation served as a baseline for comparison.

The parallel version leverages multiple processors to speed up computations. By distributing the dataset among available processors, this implementation significantly reduces execution time, especially as the number of sites and clusters increases. The parallel implementation consistently demonstrated superior performance, benefiting from efficient multi-threading and shared memory usage.

The distributed version is designed to work across multiple nodes in a distributed system. Although it balances the computational effort among processors, the overhead of communication and synchronization introduces delays. Consequently, the distributed implementation showed slower performance compared to the parallel version, particularly when the number of clusters increased.

Performance testing revealed that the parallel version of the K-means algorithm is the most efficient, significantly outperforming the sequential and distributed versions as the number of sites and clusters increases. The distributed version, despite utilizing multiple processors, incurs additional overheads that hinder its performance. These findings highlight that effective parallel processing can optimize performance, whereas distributing tasks across multiple processors does not always yield better results due to communication and synchronization overheads.

The study concludes that while all three implementations are viable for K-means clustering, the parallel version offers the best performance efficiency for larger datasets. The balance between computation and communication overhead is crucial in determining the effectiveness of distributed systems. Future work could explore optimization techniques to mitigate the overhead in distributed implementations and further enhance performance.