Empirical evaluation of sequential, parallel and distributed implementations of k-means clustering

Andrej Perković
89201045@student.upr.si
Faculty of Mathematics, Natural
Sciences and Information Technologies
University of Primorska
Glagoljška ulica 8
SI-6000 Koper, Slovenia

Aleksandar Tošić
aleksandar.tosic@upr.si
Faculty of Mathematics, Natural
Sciences and Information Technologies
University of Primorska
Glagoljška ulica 8
SI-6000 Koper, Slovenia

ABSTRACT

In this paper we present a sequential, parallel and distributed implementation of the infamous k-means clustering algorithm. We perform extensive testing of all three implementations on state the art hardware, and show the performance benefits of paralellization. The research was inspired by a use-case of reverse logistics optimisation of wood in Germany, which translates to a facility location problem. K-means is an heuristic approach that renders surprisingly good results compared to mathematical modelling approaches, which are usually not feasible in large inputs as they belong to the class of NP-hard problems.

KEYWORDS

k-means, Clustering, MPI, Parallel computing

ACM Reference Format:
Andrej Perković and Aleksandar Tošić. 2022. Empirical evaluation of sequential, parallel and distributed implementations of k-means clustering. In *Proceedings of Student Computing Research Symposium (SCORES'22).* ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The concept of reusing waste wood is not a new concept [4]. Over the years, the concept of reusing or recycling waste wood has been gaining increasing attention both from academia and industry participates. The potential of reusing waste wood has multiple benefits such as positive environmental effects as less trees need to be cut in order to meet the demand of raw materials. Currently, in most countries waste wood is collected but rarely sorted or decontaminated. Both of these processes are required before reusing waste wood. The process of sorting is important to filter out wood not suitable for reuse, which is mainly due to size constraints, and type of wood (hardwood/softwood). The decontamination process involves some mechanical cutting and grinding of parts of the suitable waste wood to remove unwanted objects (nails, screws, etc..) and chemical compounds such as adhesives. However, both of these processes are inherently costly, and the capital and operational expenses need to be justified by the added value obtained by reusing waste wood as oppose to buying new raw material [1]. To achieve this goal, legislation must both subsidise the transition to circular economy and at the same time impose restrictions or taxes on excessive CO_2 emissions [14].

Due to lack of investment and motivation by market participants, most of the waste wood is burned for energy and put into landfill where burning is not an option due to heavy contamination. Sometimes, this is done by accumulation sites directly.

For a successful implementation, new facilities for sorting and decontamination must be built. These facilities would then offer recycled wood to the market to fund their operational expenditure. The placement of such facilities is a logistics optimisation problem commonly known as the Facility Location Problem (FLP). Existing research is mostly focused on mathematical modeling and linear programming to find the near optimal positioning of facilities considering all constraints [1, 3]. However, due to the computational complexity of the problem, such solutions are not scalable for large logistic networks such as the entire EU zone.

K-means clustering has been heavily explored as a heuristic approach to solving large problems where linear programming solvers become infeasible [9]. Clustering is a simple and powerful principle for making sense of large swats of data. It is becoming more and more important in today's data-intensive and data-driven society [7]. *K*-means clustering is a rudimentary algorithm for achieving this goal. It has been one of the researchers' favorite, with a plethora of variations and tweaks [16]. It is widely studied, with the most notable work coming from Lloyd [10], Forgey [5], Friedman and Rubin [6] and MacQueen [11].

The crux of the algorithm are it's two steps - the *assignment* (binding) *step* and the *update step*. In the former, we assign each point to the "closest" cluster centroid. In the latter, we recalculate the centroid of the cluster. Definition of closeness depends on the choice of the algorithm. The algorithm stops once there are no changes in the binding. Recently, clustering has been used to improve runtime of Mixed-Integer Linear Models (MILP) to give the solver a better initial state then random [2].

In this paper we present an open-source parallel and distributed implementation of the k-means algorithm. We evaluate our implementation on the aforementioned use case using data obtained from the statistical office in Germany. Our initial data-set contains 10.000 accumulation sites, which accumulate used wood. The dataset was prepared by statistically estimating the amount of wood that should

SCORES'22, October 6, 2022, Ljubljana, Slovenia

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

accumulate in an area based on the population density, and average waste wood created per inhabitant. Both of the aforementioned statistics were obtained from the statistical office of Germany. Each site is located using GPS coordinates and distances are computed using the Cosine-Haversine Formula [13].

2 IMPLEMENTATION

To tackle the problem at hand, we decide to use the weighted k-means clustering algorithm. It's different form the classical version in that it takes into consideration the capacity of the facility, not just the Euclidean distance. This way the calculation of the centroid of a cluster is biased towards the larger collection facilities of that cluster, which reflects practical needs of placing the treatment facilities closer to places with more significant accumulation of wood. Hence, we get the coordinates of the new centroid in the following way:

$$C_j(x) = \frac{\sum\limits_{x_i \in C_j} x_i \cdot w_i}{\sum\limits_{x_j \in C_j} w_j}$$

The new y coordinate of the centroid is calculated analogously.

In the assignment step, we assign to each data point the closest centroid. The Euclidean distance formula for calculating the distance is used here. We assign point p_i to cluster C_j if it holds that:

$$j = \arg\min_l \{\sqrt{(p_i(x) - C_l(x))^2 + (p_i(y) - C_l(y))^2}\}$$

For the initialization, we decided to go with the Frogy method of randomly choosing k points as initial cluster centorids from the given data set, which is proven to be the best one for the simple k-means [8].

2.1 Parallel mode

This mode was implemented using the thread pool principle with the help of `Executors` class [12]. This allows for dynamic control of threads, so there is less possibility for human error. Synchronization was done with the `CountDownLatch` instances called `barrierBind` and `barrierUpdate`, reinitialized before each assignment and update step, respectively.

For the binding step, each thread gets an approximately equally sized chunk of `Site` collection to process. To do this, all threads need to know location of current centroids. They can safely share this since they only read the values in this step. There is no critical sections here regarding sites, since threads are accessing partitions of the `Site` list, i.e. disjoint sets. On the other hand, the parallel program can get into the race condition in the part of the code where a `Site` object is appended to a cluster's list of the given objects. For this reason, we performed the insertion of sites to appropriate lists sequentially after the computations are done. One would argue that the `ArrayList.add()` method could have been made synchronous, but there are situations where even this can fail. Moreover, doing this sequentially has a negligible influence on the performance.

In this mode, the stopping condition is returned as the result value of the function `bindCluster()`. If it is `true`, only then do we enter the block of code that initiates the update step.

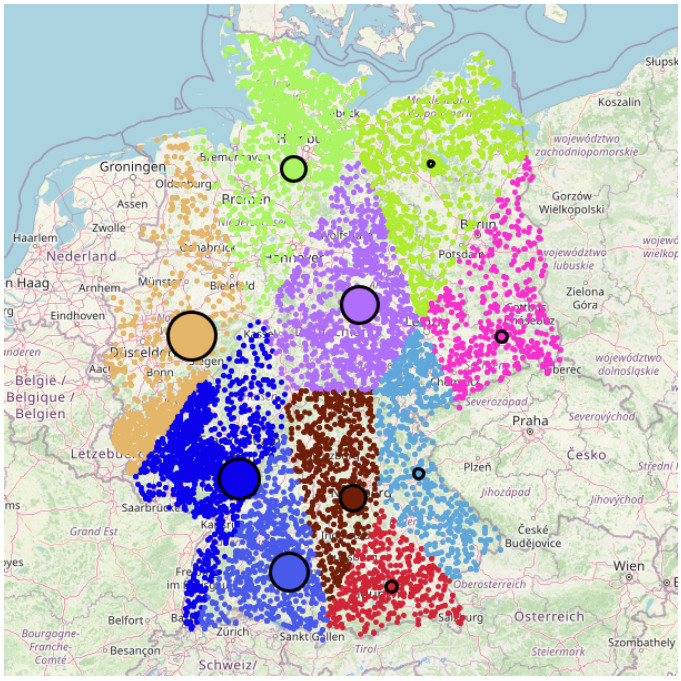

Figure 1: An example of the output results using MPI with 10 clusters of 10 000 sites.

For the update step, we call the method `updateCentroid()`, which very simply creates a `Runnable` for each cluster and sends it to the executor.

2.2 Distributed mode

This mode is more "independent", or better put, "self-contained" [12]. We implemented it with the `MPJ express` library [15]. Due to the nature of message passing and `MPJ`'s underlying implementation in `C`, it is not possible to pass complex structures between processes, like `Cluster` and `Site`. We can only natively send the primitive types. For this reason, we decided to "serialize" the `Cluster` and `Site` arrays. We transform them into `double` arrays, `centroidBuffer`, represented with orange circles in Figure 2, and `siteBuffer`, represented with blue circles, respectively. For `Cluster` transformation, we extract the *id*, *latitude* and *longitude* of each instance. Hence, for cluster i, we have its *id* at position $3i$, its *latitude* at position $3i+1$ and its *longitude* at position $3i+2$ in `centroidBuffer`. Similarly for the sites, we store the *id* at position $5i$, *latitude* at $5i+1$, *longitude* at $5i+2$, *weight* at $5i+3$ and *clusterID* at $5i+4$ in the `siteBuffer` for the site i. Process 0 acts as the master and completes the setup, that is the transformation into `double` arrays. After that, we start looping. Processes loop as long as the stopping condition is not satisfied. To check this, there is a separate buffer for flags, represented with red and green circles in Figure 2. Each process gets a flag with the help of the `Scatter` function to signalize whether it registered a change in its binding step.

Since the first step in the algorithm is assignment, the coordinator first broadcasts `centroidBuffer` and then scatters the `siteBuffer`.

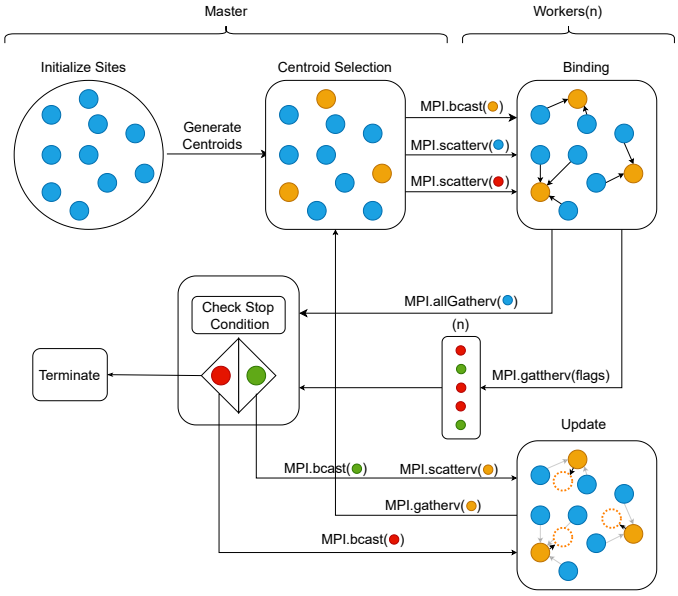

Figure 2: An outline of the distributed algorithm's workflow using MPI.

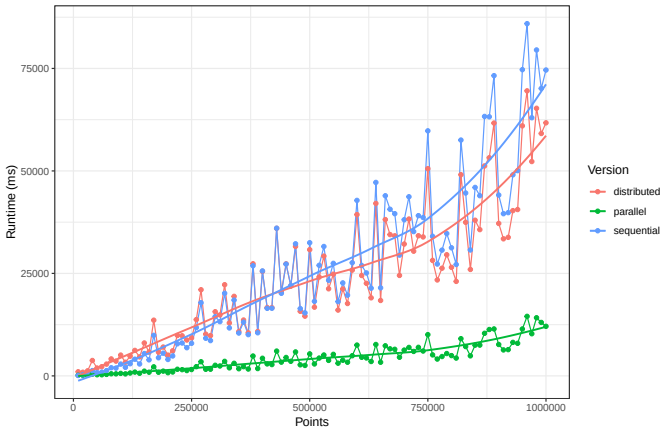

Figure 3: Performance evaluation of all three implementations. The results were obtained by increasing the number of points for each test while keeping the number of centroids at 100

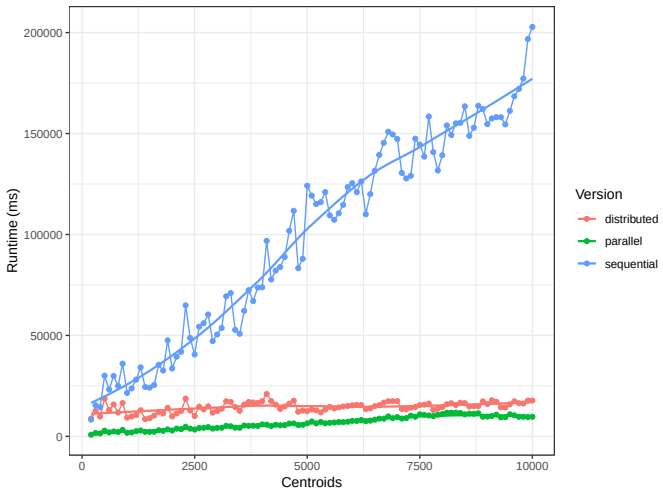

Figure 4: Performance evaluation of all three implementations. The results were obtained by increasing the number of centroids for each test while keeping the number of points at 100.000,00

We used `Scatterv` for the latter, since the number of sites need not be divisible with the number of processes running.

After this, the change flags are returned to the coordinator. It decides whether the stopping condition has been reached and broadcasts that to the workers, which is represented with the *Check Stop Condition* box on the Figure. Based on that, we either terminate the calculations or proceed to the update step.

For the update step, we decided to implement it in the way that all the processes loop through the entire `siteBuffer`, but only perform calculations on instances whose *clusterID* corresponds to the cluster centroid instance the given process is assigned. For this reason, we used the `Allgatherv` on the `siteBuffer` right after the assignment step, but then we used `Scatterv` on `centroidBuffer` to assign approximately equal number of clusters to each process.

3 RESULTS

To evaluate our implementation, all three versions were implemented and tested for performance. All tests were performed on the same hardware namely, two AMD Epyc CPU's with 64 compute cores each, 512GB of RAM running Linux. To test the performance we conduct two separate tests with 20 workers for the concurrent versions. Firstly, we test the impact the number of sites has on performance by fixing the number of centroids and increasing the number of sites. Secondly, we test the impact centroids have on performance by fixing the number of sites.

In Figure 3 we show the scalability of all three implementations. As expected, we observe a reduction in run-time in both parallel and distributed over the serial implementation. In Figure 3, we can see that the communication overhead in distributed computation pays off only for the heavier half of the test cases. In general, shared memory should perform much faster then buffered IO used by the loopback interface.

Figure 4 shows the impact of centroids on performance. As expected, the sequential version scales linearly, while both parallel and distributed see a marginal hit on performance.

4 CONCLUSIONS AND FUTURE WORK

What we did expect is for the parallel and the distributed to perform much better than the sequential. But what was unexpected is the difference between `MPI`'s performance in Figure 3 and 4. It is almost on par with the performance of the parallel version for the case of fixed number of sites, yet far from it in the case of fixed number

of centroids. We attribute this to caching. Array of sites is much bigger than the latter, so having them fixed could be the reason for the performance kick.

How different caching strategies and the Cluster Configuration of the distributed part, better reflecting the real-world performance of the aforementioned computing, influence on the comparisons made in this paper; additionally, pushing the boundary of the test data size further, beyond what can fit in a single computer's or server's memory, and how does that influence the performance of the two would be the subject of our future work.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the European Commission for funding the InnoRenew CoE project (H2020 Grant Agreement #739574) and the PHArA-ON project (H2020 Grant Agreement #857188) and the Republic of Slovenia (Investment funding of the Republic of Slovenia and the European Union of the European Regional Development Fund) as well as the Slovenian Research Agency (ARRS) for supporting project number J2-2504.

REFERENCES

[1] Michael David Burnard, Črtomir Tavzes, Aleksandar Tošić, Andrej Brodnik, and Andreja Kutnar. 2015. The role of reverse logistics in recycling of wood products. In *Environmental implications of recycling and recycled products*. Singapore : Springer, cop. 2015, 1–30.

[2] Jean-Thomas Camino, Christian Artigues, Laurent Houssin, and Stéphane Mourgues. 2021. MILP formulation improvement with k-means clustering for the beam layout optimization in multibeam satellite systems. *Computers & Industrial Engineering* 158 (2021), 107228.

[3] Péter Egri, Balázs Dávid, Tamás Kis, and Miklós Krész. 2021. Robust facility location in reverse logistics. *Annals of Operations Research* (2021), 1–26.

[4] Bob Falk et al. 1997. Opportunities for the wood waste resource. *Forest Products Journal* 47, 6 (1997), 17–22.

[5] Edward W Forgy. 1965. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *biometrics* 21 (1965), 768–769.

[6] Herman P Friedman and Jerrold Rubin. 1967. On some invariant criteria for grouping data. *J. Amer. Statist. Assoc.* 62, 320 (1967), 1159–1178.

[7] Attri Ghosal, Arunima Nandy, Amit Kumar Das, Saptarsi Goswami, and Mrityunjoy Panday. 2020. A short review on different clustering techniques and their applications. *Emerging technology in modelling and graphics* (2020), 69–83.

[8] Greg Hamerly and Charles Elkan. 2002. Alternatives to the k-means algorithm that find better clusterings. In *Proceedings of the eleventh international conference on Information and knowledge management*. 600–607.

[9] Ke Liao and Diansheng Guo. 2008. A clustering-based approach to the capacitated facility location problem 1. *Transactions in GIS* 12, 3 (2008), 323–339.

[10] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.

[11] James McQueen. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, 1967*. 281–297.

[12] Andrej Perkovic. 2022. *K Means Clustering*. https://github.com/AndrejPer/k_means_clustering

[13] C Carl Robusto. 1957. The cosine-haversine formula. *The American Mathematical Monthly* 64, 1 (1957), 38–40.

[14] Erwin M Schau, Črtomir Tavzes, Igor Gavrić, Iztok Šušteršič, Eva Prelovšek Niemelä, Balázs Dávid, Jaka Gašper Pečnik, and David DeVallance. 2022. Environmental and economic assessment of using wood to meet Paris Agreement greenhouse gas emission reductions in Slovenia. In *E3S Web of Conferences*, Vol. 349. EDP Sciences, 03005.

[15] Aamir Shafi, Bryan Carpenter, and Mark Baker. 2009. Nested parallelism for multi-core HPC systems using Java. *J. Parallel Distributed Comput.* 69, 6 (2009), 532–545. https://doi.org/10.1016/j.jpdc.2009.02.006

[16] Junjie Wu. 2012. *Advances in K-means clustering: a data mining thinking*. Springer Science & Business Media.