Improving K-means Clustering Using Speculation

Stefan Igescu[1], Viktor Sanca[1,*], Eleni Zapridou[1] and Anastasia Ailamaki[1,2,†]

[1]EPFL, Lausanne, Switzerland
[2]Google, Sunnyvale, USA

Abstract

K-means is one of the fundamental unsupervised data clustering and machine learning methods. It has been well studied over the years: parallelized, approximated, and optimized for different cases and applications. With increasingly higher parallelism leading to processing becoming bandwidth-bound, further speeding up iterative k-means would require data reduction using sampling at the cost of accuracy.

We examine the use of speculation techniques to expedite the convergence of the iterative k-means algorithm without affecting the accuracy of the results. We achieve this by introducing two cooperative and concurrent phases: one works on the overall input data, and the other speculates and explores the space faster using sampling. At the end of every iteration, the two phases vote and choose the best centroids according to the objective function. Our speculative technique reduces the number of steps needed for convergence without compromising accuracy and, at the same time, provides an effective mechanism to escape local minima without prior initialization cost through resampling.

Keywords

K-Means, Speculative Execution, Sampling, AI for DB, Clustering, Parallelization, Algorithm Co-Design, Analytics

1. Introduction

K-means clustering is an unsupervised machine learning algorithm used for grouping data points into a predefined number of clusters. It is one of the well-known and commonly used clustering algorithms, and despite being formulated in the 1960s, it is still widely applicable and practical due to its simplicity and effectiveness[1].

K-means. Given a dataset X with d-dimensional real entries, $X \subset \mathbb{R}^d$, and a positive integer k, find a set of centroids (means) C, $C \subset \mathbb{R}^d$ such that the objective function I minimizes the within-cluster sum of squares:

$$I(C, X) = \sum_{x \in X} min_{c \in C} \|x - c\|^2 \qquad \text{(Objective function)}$$

This process is iterated until no points change their assignments or the number of assigned iterations runs out. Well-known Lloyd's [2] variant is described in Algorithm 1.

Algorithm 1 K-means Clustering

1: Initialize k centroids randomly
2: **repeat**
3: Calculate the distance between each data point and the centroids
4: Assign each data point to the nearest centroid
5: Update the centroids by computing the mean of all points assigned to each cluster
6: **until** The centroids no longer move or a maximum number of iterations is reached
7: Return the final clusters and centroids

We can distinguish two main steps: the **assignment step**, which includes lines 3 and 4, and the **update step**, which includes line 5. As this is an iterative algorithm, there is a dependency between the assignment step and the update step in both directions:

1. To update the centroids, we need the assignments of all data points to centroids,
2. To update the assignments of all data points, we need the information about centroids.

By processing all the data points, K-means has an overall time complexity of $O(n \cdot d \cdot k)$ and space complexity of $O(n \cdot (k+d))$ where n is the number of data points, d is the data dimensionality, and k is the set number of clusters. In the case of large datasets, the cost is dominated by the input cardinality. Even if the data were partitioned and processed in a data-parallel memory-bound way, the algorithm's complexity and the required number of steps for convergence would remain the same.

To address the increasingly higher data volumes, and

Joint Workshops at 49th International Conference on Very Large Data Bases (VLDBW'23) — Workshop on Applied AI for Database Systems and Applications (AIDB'23), August 28 - September 1, 2023, Vancouver, Canada
*Corresponding author.
†Work done entirely at EPFL.
‡GitHub Repository.
stefan.igescu@epfl.ch (S. Igescu); viktor.sanca@epfl.ch (V. Sanca); eleni.zapridou@epfl.ch (E. Zapridou); anastasia.ailamaki@epfl.ch (A. Ailamaki)
https://viktorsanca.com (V. Sanca)
0000-0002-4799-8467 (V. Sanca); 0000-0002-5025-6835 (E. Zapridou); 0000-0002-9949-3639 (A. Ailamaki)

lower the computation time and memory capacity requirements, a possible approach is to take a random sample of the given dataset. Still, sampling trades off accuracy for a more compact data representation, which comes at a cost, as random sampling can lead to bias, inadvertently skipping data points that would change the clustering. A random sample cannot always and in an unsupervised fashion without further information represent the entire dataset. As a consequence, the final centroids that are found might not be the ones that correctly represent the initial dataset and minimize the objective function. Selecting the sample size dictates the tradeoff between execution time and centroid quality. The more we want to improve performance, the smaller the sample size has to be, sacrificing how representative the resulting centroids are. Furthermore, more elaborate sampling schemes to find a biased sample might introduce overheads that are not compatible and maskable by the algorithm throughput [3].

Ideally, while observing the iterative nature of the algorithm and the mutual dependencies in the steps, let's suppose there exists an oracle that informs the next step at no cost about the centroids it found, avoiding full data access. This would allow full data skipping with just the cost of assignment of data points to centroids. Let's suppose next that this oracle manages to look in the future over several next steps and propose with some uncertainty what the centroids after several iterations should be. This would lead not only to data skipping but full iteration skipping, in an ideal scenario reducing the algorithm to a single step only and fast-forwarding all the work.

In practice, such approaches exist under the umbrella term of speculation, as in hardware in microprocessors [4], and have also recently been applied in the domain of complex analytical queries [5]. The key characteristic of speculation is to achieve seamless and fast speculative steps and have a lightweight mechanism to detect and repair wrong proposed decisions while avoiding performance degradation.

We take the idea of speculation, tailor it for iterative K-means, introduce a novel way of guiding the objective function to converge in fewer steps without sacrificing the resulting quality, and provide the following contributions by:

- formulating speculative K-means algorithm in section 3 and show how to break iterative step dependencies,
- bridging speculative actions which are approximate with exact execution phase in section 4, and demonstrate cooperative, iterative dependency reconstruction,
- showing that the speculative phase is enhanced by sampling in section 5, which allows the speculative algorithm to quickly explore subsets of data and escape local minima,
- evaluating our approach across datasets in section 6, showing that speculative k-means converge on average in fewer steps towards the optimal solution, often finding a better solution than methods with specialized centroid initialization phases.

We provide a novel perspective for tuning a well-established unsupervised learning algorithm and an initial blueprint for using speculation in iterative algorithms.

2. Background and Related Work

K-means clustering is a widely used unsupervised machine learning algorithm for data partitioning and clustering, and this section provides an overview of the basic K-means algorithm, the existing initialization techniques, the approximations which can be done to improve performance, and the parallelization methods. We outline the desirable characteristics of these approaches and describe speculation for breaking dependencies in analytical queries [5].

2.1. Clustering quality: inertia

Inertia measures how well the K-means algorithm clustered a dataset. Inertia I is defined as the sum of squared distances of data points X to their closest cluster center X and weighted if required (w):

$$I(C, X) = \sum_{x \in X} min_{c \in C}\ w_{x,c} \cdot ||x - c||^2 \qquad \text{(Inertia)}$$

It is exactly the objective function (Equation Objective function) of the K-Means algorithm, where a good model is one with low inertia for a given number of clusters k, as naturally, as the number of clusters increases the inertia would decrease.

2.2. K-means++: improved initialization

K-means is sensitive to centroid initialization. In its classical form, the algorithm starts by randomly assigning a set of centroids, which can lead to suboptimal solutions when centroids are not carefully selected. To mitigate this, it is important to use an appropriate initialization strategy [6].

One such approach is K-means++ [7], which elects initial cluster centroids using sampling based on an empirical probability distribution that reflects the contribution

of the points to the overall inertia. This helps find the global optimum and produce better quality clusters than random initialization. In Algorithm 2, we describe the steps for the centroid initialization.

Algorithm 2 K-means++ Initialization

1: Choose one center uniformly at random among the data points.
2: **for** each data point x not chosen yet **do**
3: Compute $D(x)$, the distance between x and the nearest center that has already been chosen.
4: **end for**
5: Choose one new data point at random as a new center, using a weighted probability distribution where a point x is chosen with probability proportional to $D(x)^2$.
6: **repeat**
7: Steps 2 to 5
8: **until** k centers have not been chosen
9: Return the initial centers have been chosen

The initialization involves the computation of the distances between each data point and the nearest center. This is consequentially an expensive operation repeated every time a new centroid is sampled. Therefore, all the aforementioned benefits come to an initial time-cost tradeoff and cost to pay before running the iterative clustering phases of K-Means. This can be prohibitive in particular applications, especially in the case of large datasets.

2.3. Sampling: Mini-batch and online K-means

Sampling-based approximations aim to improve performance, usually execution time or memory constraints, by using a randomized subset of the initial input data to reduce the data volume. In addition, these algorithms are practical when dealing with large datasets that would be too costly to process fully in-memory in one go [8].

Mini-batch K-means is an iterative algorithm that uses a subset of the data to update the cluster centers. At each K-means iteration, a new mini-batch is sampled [9]. Online K-means is a streaming algorithm that updates the cluster centers using one data point at a time [10].

These methods reduce the execution time or the amount of memory needed to run K-means clustering. However, this comes at a cost to the accuracy of the final centroids found. The fact we are not using the entire dataset to find the centroids leads to exploring a subspace of possible centroids and could lead to solutions with higher inertia. Intuitively, when an unbiased sample is taken, data populations that clustering was intended to discover in the first place might be lost. Finally, elaborate in-memory sampling schemes might be impractical and costly, especially if the replaced computation via data volume reduction does not match the sampling overhead and overall throughput [3], or require workload-aware and efficient online strategies [11].

Therefore, sampling uniformly at random provides a data reduction and speedup strategy at the cost of impacting the non-uniform distribution of clusters in the original dataset.

2.4. Parallelization

With increasingly large datasets and system improvements such as larger memory capacities, and parallel and distributed architectures, one approach for addressing K-means on large datasets is to adapt it to use available hardware.

Distributed frameworks such as Spark [12, 13] and MapReduce [14] take advantage of distributed computational resources to process large datasets quickly and efficiently while keeping the dataset in memory in a scale-out manner.

On the other hand, multi-processor setups and the introduction of GPUs with higher memory capacities were driving the formulation of scale-up implementations, which exploit available data and task parallelism and high-bandwidth data access [15, 16].

Conceptually, the main goal of parallelization is to efficiently use abundant and available resources. Still, as in the case of data parallelism, Amdahl's law suggests that an increase in resources often results in diminishing returns [17]. Considering all the bottlenecks, such as data exchange, data distribution disbalance, and communication, especially in the critical sections, may lead to resources remaining idle or underutilized. Adding more resources does not necessarily lead to proportionally faster execution.

Finally, parallelization does not change the algorithmic complexity of the task. For example, the data-parallel formulation conceptually just partitions the input data over multiple workers, with defined synchronization overhead. In the case of iterative algorithms and K-Means in particular, there is a dependency between two steps (Algorithm 1), where the order and sequence of iterations must be executed one at a time, necessitating a full pass over the data to proceed to the next step.

2.5. Speculative Execution for Analytics

Speculation for analytics [5] was introduced to strike a balance between data and task parallelism and demonstrate that idle hardware resources can be redirected to breaking dependencies in complex queries that would predominantly use data parallelism.

This novel processing paradigm accelerates interdependent queries using speculation through approximate query processing and subsequent repair mechanism to achieve results equivalent to purely data-parallel execution. The speculative technique used allows for relaxing dependencies between queries, increasing task parallelism, and reducing end-to-end latency. It works by detecting dependencies between outer and inner subqueries and creating two separate execution paths: one for speculative execution and one for validation/repairs. The speculative execution path executes the outer query by resolving any cross-query conditions with the help of a branch predictor. At the same time, the validation/repair execution path performs three steps: (i) it fully computes the inner query, (ii) it validates the speculative decisions, and (iii) it repairs the results in case of mispredictions. This technique decouples the inner and outer queries and allows them to run in parallel or share data and operators if they process rows from the same table.

Speculation allows better resource utilization through data and task parallelism and subsequently opens opportunities to use techniques such as scan sharing or shared-query execution. Still, a lightweight speculation and repair mechanism is required to exploit proposed desirable optimization opportunities.

3. Speculative K-Means

Taking into account the desirable characteristics of the K-means algorithm, an improved algorithm would:

- O1: reduce the number of steps and time needed to converge,
- O2: achieve equal or better centroid result quality,
- O3: avoid costly initialization while escaping local minima.

These objectives are seemingly contradictory and mutually conflicting. O1 would imply an approximate solution that conflicts with O2, while O3, in conjunction with O1, would result in approximate and potentially significantly reduced result quality. There is a clear tradeoff between the resulting quality and necessary (pre)computation time [7].

To bridge these objectives, we propose a speculative formulation where in conjunction with the exact execution, a speculative step will be happening that satisfies the objectives and terminates with equal (or better) cluster characteristics, i.e., lower inertia as a measure of clustering quality (Equation Inertia) We refer to Algorithm 1 and **assignment step (A)** in lines 3 and 4, and **update step (B)** in line 5, and the existence of strict cyclical dependency between A and B that we want to break and allow more quickly traversing the chain of iterations towards the final solution.

Speculative K-Means (Figure 1, Algorithm 3) combines the regular (slow) and speculative (fast) execution in a task-parallel manner. Fast exploration is based on sampling, managing to finish multiple iterations until synchronization with the single slow iteration. The key component to speculation is merging and repairing at the end of every slow iteration. At that point, centroids from fast execution (from an unreliable oracle) are compared with slow execution using inertia (subsection 2.1), and better centroids are selected.

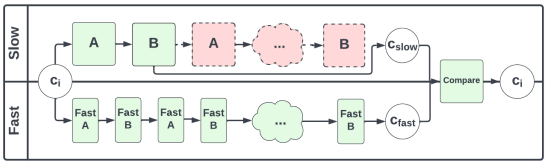

Figure 1: Conceptual Design of Speculative K-means

The fast worker allows no-regret exploration and potentially avoids traversing the entire chain to reach the final solution by skipping some intermediate steps using the sample-based prediction of clusters.

When the slow execution finishes processing its stage, the fast one is interrupted. Both the fast and slow execution propose the centroids they found, and the best ones are chosen by computing their inertia over the entire dataset (i.e., objective function). The centroids which lead to lower inertia are selected. This procedure is repeated until convergence or algorithm termination, starting from the newly selected centroids.

Algorithm 3 K-means Clustering using Speculation

1: Initialize centroids c_i randomly
2: Create two workers: slow and fast
3: Run 1 Assignment stage and 1 Update stage on the slow execution starting from c_i
4: Run as many as possible stages on a sample of the dataset on the fast execution starting from c_i
5: When slow execution finishes, stop fast execution. Get the proposed centroids c_{slow}, c_{fast}
6: Compute the inertia $I(X, c_{slow})$, $I(X, c_{fast})$ on the entire dataset X
7: $c_i = argmin_c(I(X, c_{slow}), I(X, c_{fast}))$ select centroids minimizing the inertia.
8: **while** not (centroids no longer move | a maximum number of iterations is reached) **do**
9: repeat 2 to 7
10: **end while**
11: Return the final clusters and centroids

We satisfy the objectives (Algorithm 3) using collaboratively:

- O1: sampling-based speculation to guide the objective function (lines 4, 5),
- O2: merge and vote on better centroids along the slow step (lines 6, 7),
- O3: space exploration by dropping-out points using sampling (line 4).

This approach is not slower to converge than a standard K-means in terms of steps, and it produces better clusters than approximate methods thanks to the centroid voting and consolidation, which we demonstrate in section 4.

Furthermore, it can converge closer to the global minimum without costly centroid initialization as in K-means++, as the centroids are resampled each time during the fast execution. Since the K-means quality is strictly conditioned by its initialization, by resampling the centroids from which the fast execution starts each time, we study the effect of avoiding local optima through random space exploration in section 5.

4. Exploitation: Fast and Accurate

Firstly, we are interested in the qualitative behavior of speculative K-means against vanilla K-means (Lloyd's algorithm) and K-Means++ as an approach with a better initialization strategy. We observe and compare the evolution of the clusters' inertia against the computed estimate of the optimal clustering and its respective inertia. This will be essential to understand if our method is reducing the inertia faster than existing methods and to verify where our method is converging with respect to the global minimum and, in particular, if it can reach it.

As the fast execution (Figure 1) performs several assignments and update steps, allowing it to converge faster, the slow execution computes the result equivalent to vanilla K-Means on the entire dataset. This means that the fast execution can suggest centroids that are closer to the final solution, reducing the total number of steps needed to converge and, consequently, the overall time of execution. At the same time, the quality of the final solution is still comparable to the case when working with the entire dataset, thanks to the slow execution, which can find the best (naive) position of the centroids to reduce the inertia in case of bad speculation.

4.1. Breaking the K-means dependencies

The vanilla (naive) K-means Algorithm 1 consists of two main steps: the assignment step and the update step. The assignment step includes lines 3 and 4, while the update step includes line 5. The two steps are dependent on each other, as the assignment of data points depends on the current cluster centers, which are, in turn, updated based on the assignments. This creates a cyclical pattern in the K-means algorithm.

We can group the K-means steps into stages, each containing an Assignment step (A) and an Update step (B), with many of these stages repeated in the K-means algorithm until convergence. One way to speculate is to try to run parallel the Assignment and the Update step, thus reducing the overall time of execution of a single stage. In particular, we would speculate the result of the Assignment (or the Update) allowing the consequent Update (or the Assignment) to start running concurrently. Finally, we could perform a repair stage once the correct result of the Assignment (or the Update) is available.

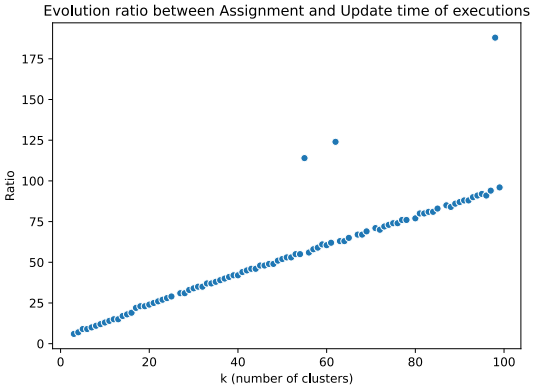

Figure 2: Evolution of ratio between the execution time of Assignment and Update steps.

After a study of the time of executions (Figure 2), the Assignment step takes more time than the Update step. In particular, the ratio between the time of an Assignment step and the time of the Update step increases with k. Consequently, the parallelization of Assignment and Update in each stage is inefficient since the gain of added task parallelism is irrelevant when $t_{Assignment} \gg t_{Update}$.

Therefore, we must use an approach that allows fast speculation (and repair) when implementing iterative speculation. Instead of running Assignment and Update concurrently for each stage, we run two independent and concurrent executions of K-means (Fast and Slow, Figure 1). Both executions start with the same given centroids, but: the slow execution processes one stage consisting of one Assignment stage and one Update step using the full dataset, while the fast execution runs on a sample of the dataset and is able to process multiple stages in the same amount of time as the slow execution. As soon as the slow execution finishes, the fast execution is interrupted, guaranteeing that they both take the

same amount of time. Finally, we compare the centroids proposed by both executions, compute the overall objective function over the entire dataset given both proposed centroids from Fast and Slow execution, and pick the centroids with the lowest inertia. This step can be done in a single scan of the data, overlapping it with the mandatory step of the Slow execution path.

The fast execution of the K-means algorithm is used to speculate and predict an approximation of future centroids that the slow execution may encounter by traversing a speculative path. If the prediction is correct, it will allow skipping several stages of the K-means algorithm, breaking the cyclic dependency. This is possible since it works on a subset of the dataset, allowing it to execute faster and dive deeper into the algorithm, guiding the objective faster to convergence. However, as shown in section 2.3, the approximate approach can lead to less accurate results when applied to the full dataset due to using an unbiased data sample. To address this issue, Slow execution is deployed to compute the result using the entire dataset, ensuring that Speculation K-means will converge toward a solution that considers the entire dataset. Thus, the two executions have different purposes, with the fast execution being used to speculate and the slow execution being used to ensure accuracy.

In Figure 1, the green shapes represent the steps executed, whereas the red ones indicate the steps we would save in case the speculation of the Fast execution suggests a set of centroids with lower inertia.

The main difference between the speculative execution used in [5], discussed in section 2.5, and the Speculation K-means technique is that in the former, the speculation is used to break the dependency between the inner and outer query to increase parallelism, while in the latter, the speculation is used to predict possible future results (i.e., centroids). Therefore, we are breaking the dependencies in a different sense: we are traversing the entire chain of dependencies to arrive at the final result. Additionally, the correction phase in our approach ensures an accurate convergence of a sequence of actions in an iterative algorithm, rather than just the result of dependent subqueries.

In Figure 3, we compare vanilla K-Means, K-Means++ (with better initialization), and speculative K-means as proposed until now. All start from the same initial centroid, except K-Means++, which has an additional step of initial centroid computation which is not represented. After this, they compute $k = 8$ clusters on the OpenML [18] Dataset 23395 [19]. OpenML Dataset 23395 and run for a maximum of 50 stages. Additionally, the optimal inertia achievable is estimated by running 50 times K-means++ with different initial centroids and then picking the solution with minimum inertia. The run with speculation is more efficient, as it reaches convergence approximately after 10 stages, while the vanilla implementation takes

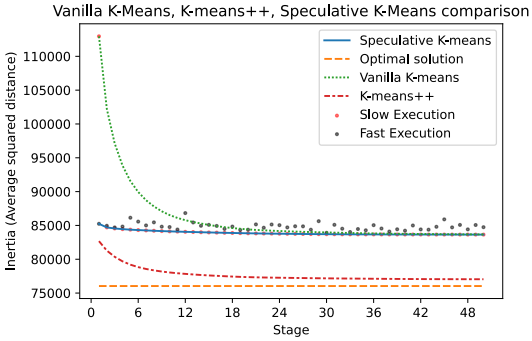

Figure 3: Evolution of the inertia of two runs starting from the same initial centroids: one based on Vanilla K-means, the other on Speculation K-means, and K-Means++. The Figure illustrates the estimated optimal achievable inertia and the inertia of the centroids suggested by the slow run and the fast run at each stage.

30 stages. Furthermore, the accuracy of the final result is the same for both executions, showing how the slow execution and the correction phase ensure a comparable final inertia.

4.2. Reconstructing the dependencies

As the algorithm approaches convergence, the accuracy and effectiveness of the speculation carried out by the fast execution decrease, as shown in Figure 3. In the initial stages, the inertia of the centroids proposed by the fast execution is significantly smaller than that of the centroids of the slow execution. Still, it becomes comparable in the following 3/4 stages. Additionally, as we approach convergence, the inertia of the fast execution's centroids is constantly greater than that of the slow execution. This is because the last stages of K-means before convergence need the entire datasets to compute the exact position of the centroids, and we cannot do this by using a sample of it. This leads us to question what to do with the fast execution and if we can improve its prediction even during the last stages before convergence. This is especially noticeable, as in this formulation speculative approach cannot offer better result quality, similar to K-means++.

First, we propose keeping track of the previously predicted centroids in fast execution. Every time a fast execution starts, we sample a different subset of the dataset it will work on. Each subset will lead to a different centroids prediction. Therefore we decide to keep their memory of them using a tracing method, which consists in updating the newly discovered c_{fast} centroids with the c_{fast}^{past} past centroids in the following way:

$$c_{fast} = q \cdot c_{fast} + (1 - q) \cdot c_{fast}^{past} \tag{1}$$

Again, we can see how q plays the role of a hyperparameter which determines how much we should keep track of the past speculated centroids. This type of tracing leads to an exponential decay of the memory of the past centroids. The reason why this method should help the prediction of the fast execution in the late stages of K-means is that it tries to combine the contribution of different speculation passes. Suppose the problem is that a dataset sample cannot represent the entire dataset in the computation of the centroids. In that case, combining the contribution of different samples should mitigate this limitation. In Figure 4, we can see how with tracing, the prediction of the fast execution is more accurate even in the last stages and has lower variance guided by past execution.

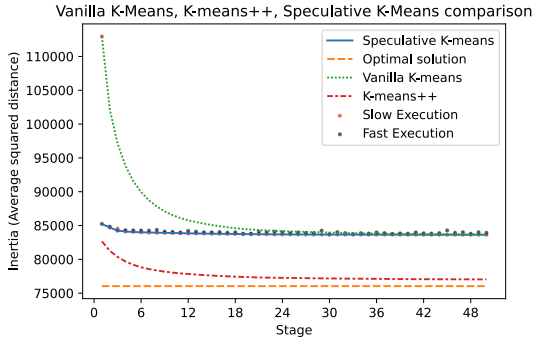

Figure 4: Evolution of the inertia of two runs starting from the same initial centroids: one based on Vanilla K-means, the other on Speculation K-means with tracing $q = 0.5$, without centroid resampling.

An alternative approach is to gradually increase the sample size of the data points that the fast execution algorithm is working with. As previously mentioned in subsection 2.3, there is a trade-off between performance and accuracy when using subsamples in K-means, with smaller sample sizes leading to faster stages but less accurate predictions. At the beginning of the algorithm, it may be beneficial to use a small sample size to quickly explore different speculation paths and arrive at centroid estimates that are close to the final solution. In this phase, the accuracy of the predicted centroids is not as important as being able to delve into the speculation path for numerous stages. However, as we approach convergence, it becomes more crucial to have accurate speculation. In this phase, we need to make small adjustments to the centroids to find their optimal positions, and to do this, we need access to the full dataset. Thus, we can gradually increase the sample size to reduce the number of stages processed by the fast execution (which is not a significant concern at this point since we are close to convergence) and improve the accuracy of our predictions by having an increasing complete view of the data.

While this approach allows fast convergence informed by previous speculative steps, Figure 4 demonstrates that speculation does not yet approach the inertia as that of dedicated centroid initialization (K-Means++).

5. Exploration: Escaping Local Minima

Figure 3 illustrates that both the Speculation K-means and the vanilla version do not converge to the optimal minimum nearly as K-Means++ does in these runs. As discussed in Section 2, this corroborates that the initialization of K-means plays a crucial role in determining the quality of the final clustering. Initialization techniques like K-means++ improve the final results, but have an initial time cost (that we do not include in the comparison). We propose an alternative approach that avoids this initial cost while still leading closer to a global minimum by escaping the initial centroids through fast and randomized space exploration.

Figure 1 and Algorithm 3 show how the fast execution starts from the same centroids as the slow execution. However, by doing this, both executions are conditioned by the first initialization of the centroids. Instead, we modify our speculative approach and fast execution to start from a slightly different set of centroids than the slow one. This allows taking advantage of the fast execution to explore more of the solution space, searching for global minima. At the same time, in the repair and consolidation stage, where we compare the found centroids by computing their inertia with the slow phase, the results will be at least as good as the vanilla approach if more aggressive exploration failed. This method is similar to having a vanilla K-means executed many times from different initial centroids and picking the best-obtained solution. Still, it is implemented only on the fast execution, and it is distributed over many speculative steps.

Each time before starting a fast speculation phase, we sample k data points from the dataset as new initial centroids c_i' and modify the starting centroids of the fast execution by combining the given initial centroids and the sampled ones linearly:

$$c_i = p \cdot c_i + (1 - p) \cdot c_i' \tag{2}$$

The value p determines how intensely we want to perturb and mix the initial centroids: the closer it is to 0, the more random the centroids are, and the more we explore the solution space. This parameter expresses the degree of exploration we want from the fast execution. It can be set statically before the K-means execution to an optimal value which may vary from dataset to dataset, or it can be adapted at run time by adapting the randomness based on

the quality of solutions we are finding stage after stage. In Figure 5 we can see how using the centroid resampling technique with $p = 0.8$ improves the inertia of the final clusters.

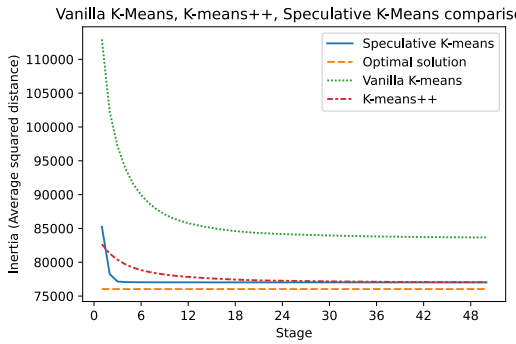

Figure 5: Evolution of the inertia starting from the same initial centroids: one based on Vanilla K-means, the other on Speculation K-means with centroid resampling, and K-Means++ with its own initialization step, $p = 0.8$.

With centroid resampling and starting centroid perturbation, we achieve guided and bounded randomization such that the increased exploration opportunities potentially break the bad initialization without an explicit and expensive initial centroid computation of K-Means++. Still, while this speculative approach is probabilistic, it will not be worse than the vanilla K-means, but it might not yield better results in all cases, conditional on available hyperparameters such as perturbation and sampling rate.

6. Holistic Evaluation

In this section we evaluate our speculative K-means formulation against vanilla (Lloyd's) K-means and K-means++ with a better initialization step. We focus on comparing the parameters such as inertia and the number of stages before convergence to allow an algorithmic-oriented analysis of speed and quality of clustering.

We implement a prototype of speculative K-Means using Python and compare it against the K-means baselines available in the widely used Scikit-learn [20] machine learning library in Python.

6.1. Datasets

The datasets on which the evaluations are done are retrieved from OpenML [18] an online platform for machine learning research, which provides a database of machine learning datasets. We query the platform for 10 datasets that follow the conditions shown in Table 1. This allows us to report an averaged-out summary of our findings that is not biased over a given dataset, to improve the empirical evaluation using datasets available on the platform.

NumberOfInstances	>	$5 \cdot 10^5$
NumberOfInstances	<	10^7
NumberOfNumericFeatures	>	5
NumberOfNumericFeatures	<	50
NumberOfMissingValues	=	0
NumberOfSymbolicFeatures	=	0

Table 1
OpenML datasets features used in the evaluation.

6.2. Faster convergence with fewer stages

In this section, evaluate how speculative K-means can improve the performance of the vanilla K-means by reducing the number of stages required to reach convergence.

We conduct experiments on 10 datasets with the characteristics listed in Table 1, using both vanilla K-means and speculative K-means with a number of clusters k ranging from 3 to 10. We measure the number of stages each algorithm took to reach convergence, which we defined as the point when the relative difference in inertia between two stages is below 10^{-3} or when the assignments are not changing anymore. Next, we computed the ratio of the number of stages taken by speculative K-means to the number of stages taken by vanilla K-means and plotted a histogram of these ratios. Additionally, we computed the ratio of the inertia of the final centroids obtained from the two algorithms.

In this setup, the goal is to achieve most of the stages ratios below the value of 0.5, and ideally closer to 0. This is because we are trying to reduce the convergence time and to achieve a 50% reduction in overall time, it is, therefore, necessary to reduce the number of stages by at least half. The results, shown in Figure 6, demonstrate that speculative K-means is often successful in reducing the number of stages to less than half the original amount in most cases, with most of the ratios in the range of 0 to 0.5.

On the other hand, the ratio of inertia on the same graph is better when the value approaches 1, showing that the quality of the final centroids obtained using Speculative K-means is not compromised, and may even be improved, despite the increased performance. This suggests that Speculative K-means has the potential to significantly improve the efficiency of the K-means algorithm without sacrificing solution quality.

Previously, we showed that the speculation technique can significantly reduce the number of stages required for convergence in K-means clustering. However, there are some time offsets to consider due to the repair and speculation processes, including sampling a subset of

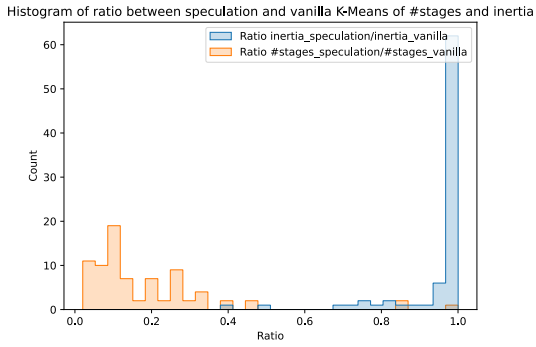

Figure 6: The histograms show the distribution of the ratio of the number of stages in speculative K-means to the number of stages in vanilla K-means (orange) and the ratio of the inertia in speculative K-means to the inertia in vanilla K-means (blue). The speculative execution used $subsample\_size = 0.01$ and tracing 4.2 with $q = 0.5$.

points for the fast execution and the repair step, which is the most time-consuming due to the computation of inertia of centroids from the fast and slow execution threads. This computation is similar in complexity to the Assignment step, the most time-consuming step in K-means, as shown in Figure 2. In a simple implementation, there would be 1 Assignment step for the slow execution and 2 Inertia steps for the repair, totaling 3 steps comparable to the Assignment. This can be reduced to 2 steps by using the results from the inertia computation in one stage to compute the Assignment step in the next stage.

As the Assignment step is the most time-consuming, each stage in Speculative K-means requires approximately double the time of a stage in Vanilla K-means at most. While halving the total number of stages can compensate for the increased time per stage, the overall time of execution would still be the same. Improving the computation of inertia through better parallelization, evaluation using hardware accelerators, using techniques such as scan sharing, or alternative approaches such as avoiding explicit computation of inertia could further improve the performance of Speculative K-means. These approaches remain part of future work and directions.

6.3. Exploration and escaping the local minima

In this section, we show how Speculative K-means can escape local minima by using the resampling centroid technique described in section 5. To demonstrate this, we conducted executions with increasing values of k in a range of values between 3 and 10. We run three versions of K-means: vanilla K-means, K-means++, and Speculative K-means with centroid resampling. For all executions, we also compute the optimal solution and the respective inertia and then calculate the ratio of the inertia of the solution found by each of the three methods to the optimal one. Finally, we plotted histograms of these ratios for all three methods. In these executions, Vanilla and Speculative K-means always started from the same initial centroids, whereas K-means++ used its initialization technique.

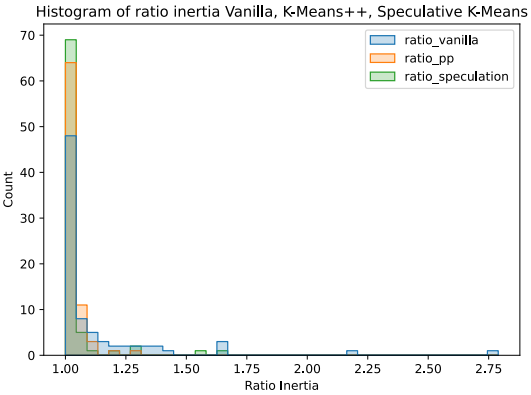

Figure 7: The figure shows the ratio between the inertia of the centroids found by Vanilla, Speculative K-means, and K-means++ (ratio_pp) to the optimal inertia. Speculative K-means used a $subsample\_size = 0.01$ and centroids resampling with $p = 0.5$

As depicted in Figure 7, Speculative K-means had the highest number of runs where the ratio was close to 1, indicating that it was most successful in reaching closer to the global minimum inertia. Then it is followed closely by K-means++ and finally by Vanilla K-means.

These measurements suggest that Speculative K-means, with centroid resampling, is indeed able to escape local minima without an initial time cost. In this particular case, it also outperforms K-means++ due to faster convergence and better exploration.

On the other hand, the same observation holds when the distribution of the number of steps is plotted in Figure 8. While this corroborates that the initialization step is important, as K-means++ requires fewer steps than the vanilla K-means, speculation allows convergence in fewer steps.

6.4. Adversarial dataset evaluation

Finally, we evaluate the performance of speculative K-Means clustering over typical adversarial datasets depicted in Figure 9. K-means algorithm family cannot provide satisfactory clustering and data separation that

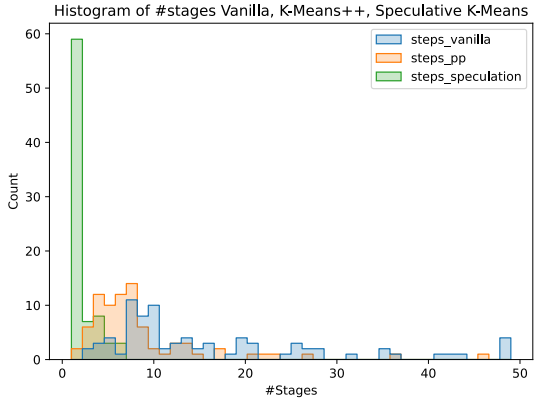

Figure 8: The figure shows the histogram of the number of steps required by Vanilla, Speculative K-means, and K-means++ (steps_pp) to the optimal inertia. Speculative K-means used a *subsample_size* = 0.01 and centroids resampling with $p = 0.5$.

follows the given patterns. Still, using these kinds of datasets allow for exploring the impact of initialization and speculative execution.

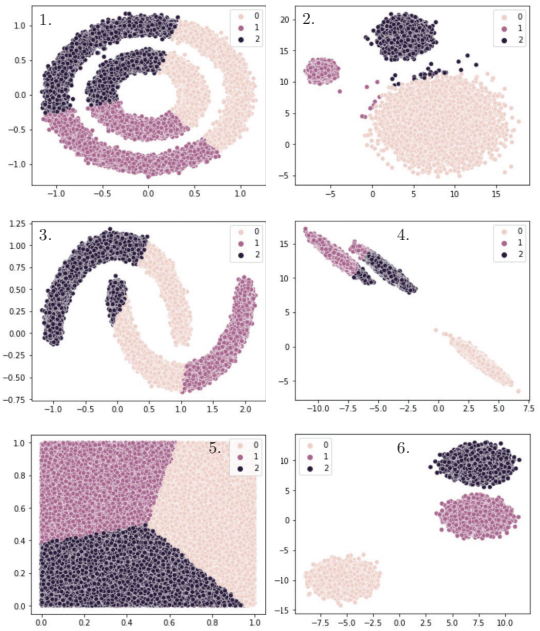

Figure 9: Adversarial/Difficult datasets: 1. noisy circles, 2. varied, 3. noisy moons, 4. anisotropic, 5. no structure, 6. blobs.

We first compare the inertia obtained by Vanilla K-Means, K-Means++, and Speculative K-Means in Figure 10. While the ideal inertia cannot be captured by the objective function of the K-means clustering, the results show that Speculative K-Means is always better or equal to Vanilla K-Means. On the other hand, the default parameters of the exploration mechanism have not always escaped the initial centroids in two cases, in comparison to K-Means++. This motivates a more detailed future case study regarding adaptive sampling and runtime tuning of parameters that we introduced in the speculative version of K-means.

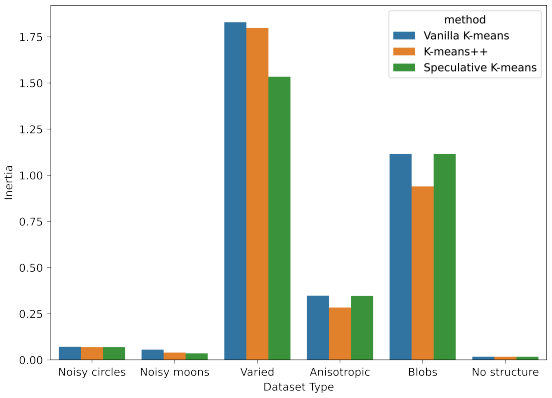

Figure 10: Final inertia comparison over adversarial datasets for different clustering methods.

Still, the number of steps to convergence, respectively after the initialization step for K-Means++, as presented in Figure 11 demonstrates that Speculative K-Means allows consistently faster convergence with fewer iterative steps.

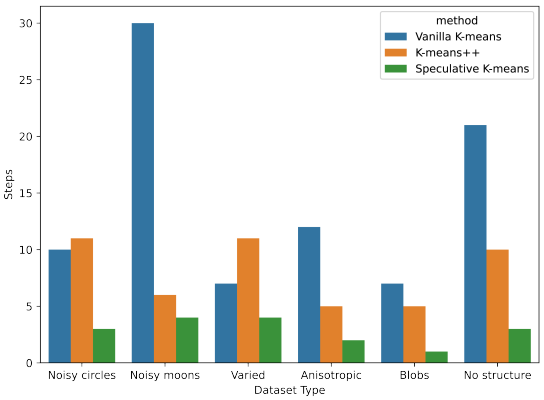

Figure 11: Number of steps to convergence over adversarial datasets for different clustering methods.

We have extensively evaluated our approach against two baselines (K-means++ and vanilla K-Means) and multiple datasets, and we demonstrated our initial hypothesis that speculation bridges faster convergence without loss of accuracy and allows probabilistic exploration to allow escaping local minima without prior initialization. Future directions include a thorough evaluation of hardware-oriented optimizations, resource allocation, and the impact of speculative execution on the overall parallel execution cost.

7. Conclusion

Despite K-Means being a long-standing and well-studied machine learning algorithm, systems and data analytics-inspired tuning and optimizations such as speculation can bridge strict tradeoffs between different desirable characteristics of algorithm variants. We combine randomized space exploration based on sampling to allow escaping local minima while achieving strictly equal or better results than the original K-Means formulation. We break the cyclic dependency in the iterative K-Means formulation while preserving the original algorithm output quality.

The key characteristic of speculative K-Means is that while phases are concurrently working on optimizing task-local objectives, cooperative merging and lightweight repair designed for speculation for iterative algorithms allows merging and tuning to the desired objective function. We extend speculative execution in data analytics with the first iterative machine learning algorithms and combine initially conflicting but advantageous methods through cooperative algorithmic and system design.

Acknowledgments

We thank the anonymous reviewers for their insightful comments and detailed feedback. This work has been partially supported by Facebook Next-generation Data Infrastructure Research Award (2021).

References

[1] A. K. Jain, Data clustering: 50 years beyond k-means, Pattern recognition letters 31 (2010) 651–666.

[2] S. Lloyd, Least squares quantization in pcm, IEEE transactions on information theory 28 (1982) 129–137.

[3] V. Sanca, A. Ailamaki, Sampling-based aqp in modern analytical engines, in: Data Management on New Hardware, DaMoN'22, Association for Computing Machinery, New York, NY, USA, 2022. URL: https://doi.org/10.1145/3533737.3535095. doi:10.1145/3533737.3535095.

[4] J. L. Hennessy, D. A. Patterson, Hardware-based speculation, in: Computer Architecture, Sixth Edition: A Quantitative Approach, 6th ed., Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2017, pp. 208–217.

[5] P. Sioulas, V. Sanca, I. Mytilinis, A. Ailamaki, Accelerating complex analytics using speculation, 2021. URL: https://www.cidrdb.org/cidr2021/papers/cidr2021_paper03.pdf.

[6] P. Fränti, S. Sieranoja, How much can k-means be improved by using better initialization and repeats?, Pattern Recognition 93 (2019) 95–112. URL: https://www.sciencedirect.com/science/article/pii/S0031320319301608. doi:https://doi.org/10.1016/j.patcog.2019.04.014.

[7] D. Arthur, S. Vassilvitskii, K-means++: The advantages of careful seeding, volume 8, 2007, pp. 1027–1035. doi:10.1145/1283383.1283494.

[8] J. Bejarano, K. Bose, T. Brannan, A. Thomas, K. Adragni, N. K. Neerchal, G. Ostrouchov, Sampling within k-means algorithm to cluster large datasets, UMBC Student Collection (2011).

[9] D. Sculley, Web-scale k-means clustering, in: Proceedings of the 19th International Conference on World Wide Web, WWW '10, Association for Computing Machinery, New York, NY, USA, 2010, p. 1177–1178. URL: https://doi.org/10.1145/1772690.1772862. doi:10.1145/1772690.1772862.

[10] L. Bottou, Y. Bengio, Convergence properties of the k-means algorithms, in: G. Tesauro, D. Touretzky, T. Leen (Eds.), Advances in Neural Information Processing Systems, volume 7, MIT Press, 1994. URL: https://proceedings.neurips.cc/paper/1994/file/a1140a3d0df1c81e24ae954d935e8926-Paper.pdf.

[11] V. Sanca, P. Chrysogelos, A. Ailamaki, Laqy: Efficient and reusable query approximations via lazy sampling, 2023, p. 15. doi:10.1145/3589319.

[12] B. Wang, J. Yin, Q. Hua, Z. Wu, J. Cao, Parallelizing k-means-based clustering on spark, in: 2016 International Conference on Advanced Cloud and Big Data (CBD), 2016, pp. 31–36. doi:10.1109/CBD.2016.016.

[13] I. Kusuma, M. A. Ma'Sum, N. Habibie, W. Jatmiko, H. Suhartanto, Design of intelligent k-means based on spark for big data clustering, in: 2016 international workshop on Big Data and information security (IWBIS), IEEE, 2016, pp. 89–96.

[14] W. Zhao, H. Ma, Q. He, Parallel k-means clustering based on mapreduce, in: Cloud Computing: First International Conference, CloudCom 2009, Beijing, China, December 1-4, 2009. Proceedings 1, Springer, 2009, pp. 674–679.

[15] Y. Zhang, Z. Xiong, J. Mao, L. Ou, The study of parallel k-means algorithm, in: 2006 6th World Congress on Intelligent Control and Automation, volume 2, IEEE, 2006, pp. 5868–5871.

[16] R. Farivar, D. Rebolledo, E. Chan, R. H. Campbell, A parallel implementation of k-means clustering on gpus., in: Pdpta, volume 13, 2008, pp. 212–312.

[17] Rules of thumb in data engineering, in: Proceedings of 16th International Conference on Data Engineering (Cat. No. 00CB37073), IEEE, ????, pp. 3–10.

[18] J. Vanschoren, J. N. van Rijn, B. Bischl, L. Torgo, Openml: networked science in machine learning, SIGKDD Explorations 15 (2013) 49–60. URL: http://doi.acm.org/10.1145/2641190.264119. doi:10.1145/2641190.2641198.

[19] Openml dataset 23395, ???? URL: https://www.openml.org/d/23395.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.