Staleness-Reduction Mini-Batch K-Means

Xueying Zhu, Jie Sun, Zhenhao He, Jiantong Jiang, and Zeke Wang

***Abstract*— K-means (km) is a clustering algorithm that has been widely adopted due to its simple implementation and high clustering quality. However, the standard km suffers from high computational complexity and is therefore time-consuming. Accordingly, the mini-batch (mbatch) km is proposed to significantly reduce computational costs in a manner that updates centroids after performing distance computations on just a mbatch, rather than a full batch, of samples. Even though the mbatch km converges faster, it leads to a decrease in convergence quality because it introduces staleness during iterations. To this end, in this article, we propose the staleness-reduction mbatch (srmbatch) km, which achieves the best of two worlds: low computational costs like the mbatch km and high clustering quality like the standard km. Moreover, srmbatch still exposes massive parallelism to be efficiently implemented on multicore CPUs and many-core GPUs. The experimental results show that srmbatch can converge up to 40×–130× faster than mbatch when reaching the same target loss, and srmbatch is able to reach 0.2%–1.7% lower final loss than that of mbatch.**

***Index Terms*— Clustering, K-means (km), machine learning, staleness-reduction.**

I. Introduction

K-MEANS is a very popular clustering algorithm and has been widely adopted in machine learning areas for tasks like image segmentation [1], data mining [2], crime prediction [3], and so on. It aims to divide data points into a predefined number of clusters so that data points in the same cluster are more similar to each other than to data points in other clusters.

The standard k-means (km) was first proposed among all kinds of km algorithms. The process of km contains the initialization step, the assignment step, and the update step. In the initialization step, it picks several samples from the dataset randomly and takes these samples as the centroids of clusters. In the assignment step, each data point in the dataset is sampled and assigned to its nearest centroid. In the update step, each cluster centroid is updated to be the centroid of all the samples assigned to it in the latest assignment step. The assignment step and the update step make up an iteration. These two steps run iteratively until the centroids stop changing. We can see that in km, centroids are only updated once after performing distance computation between all centroids and all data points, which brings km high clustering quality[1] and also massive computational complexity. As a result, the practice of km especially on large datasets is either time-consuming or computing-consuming.

To reduce computational costs, Sculley proposes the mini-batch (mbatch) km algorithm [4] that in the assignment step, it computes the distances on a mbatch, rather than a full batch, of samples per iteration, and then in the update step, each cluster centroid is updated to be the centroid of samples assigned to it from all passed assignment steps, instead of just from the nearest assignment step. It's worth noting that for mbatch, multiple samples which are used to update centroids in a certain iteration may match the same data point because data points can be sampled repeatedly to build different mbatchs. We define *assignment result* of a data point as the centroid id that the data point is assigned to. An assignment result is generated when a data point is sampled and assigned to a centroid in the assignment step. Besides, we use the word *contribute* to describe the operation that the corresponding sample of an assignment result is used to update centroids in a way we describe above for km and mbatch. As such, mbatch enjoys a relatively fast convergence rate due to more frequent centroid updates while suffering from a relatively low clustering quality because many stale assignment results which belong to the same data point contribute to new centroids. For example, suppose data point $\vec{a}$ is sampled three times in different iterations in total. The sample in the first iteration is assigned to the first centroid, the sample in the 100th iteration is assigned to the second centroid, and the sample in the 200th iteration is assigned to the third centroid. In this case, mbatch uses three different assignment results of $\vec{a}$ to update centroids in the 200th iteration. The assignment results of $\vec{a}$ in the first iteration and the 100th iteration are staler compared with the assignment in the 200th iteration. However, mbatch still allows these stale assignments to contribute to the latest centroids, which decreases the clustering quality. To quantify the level of the participation of stale assignment results, we

Manuscript received 7 January 2023; accepted 15 May 2023. This work was supported in part by the Program of Zhejiang Science and Technology under Grant 2022C01044; in part by the National Key Research and Development Program of China under Grant 2022ZD0119301; in part by the Fundamental Research Funds for the Central Universities under Grant 226-2022-00151; and in part by the Key Laboratory for Corneal Diseases Research of Zhejiang, Starry Night Science Fund of Zhejiang University Shanghai Institute for Advanced Study under Grant SN-ZJU-SIAS-0010. *(Corresponding author: Zeke Wang.)*

Xueying Zhu, Jie Sun, and Zeke Wang are with the Collaborative Innovation Center of Artificial Intelligence, Department of Computer Science, Zhejiang University, Hangzhou 310027, China (e-mail: zhuxueying@zju.edu.cn; jiesun@zju.edu.cn; wangzeke@zju.edu.cn).

Zhenhao He is with the Systems Group, Department of Computer Science, ETH Zurich, 8092 Zurich, Switzerland (e-mail: zhe@inf.ethz.ch).

Jiantong Jiang is with the Department of Computer Science and Software Engineering, The University of Western Australia, Crawley, WA 6009, Australia (e-mail: jiantong.jiang@research.uwa.edu.au).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2023.3279122.

Digital Object Identifier 10.1109/TNNLS.2023.3279122

[1]The clustering quality is qualified by loss. The loss equals the sum of squared distances between each data point in the dataset and the corresponding centroid which is the closest one to the data point. The lower the loss is, the better the clustering quality is.

first define an *epoch* as a certain number of iterations within which all data points in the dataset have been traversed once in mathematical expectation and define the *epoch gap* as the number of epochs between where the assignment result is generated and where it contributes to the centroids. Based on these two definitions, we then introduce a concept called *centroid staleness*. The centroid staleness is defined to be the average epoch gap of assignment results contributing to the centroids. In each iteration of an epoch, the centroid set updated has its own centroid staleness. Intuitively, the higher the level of the participation of stale assignment results is, the higher the average epoch gap is and the higher the centroid staleness is.

To alleviate mbatch's inferior clustering quality caused by centroid staleness, we propose the staleness-reduction mbatch (srmbatch) km algorithm that keeps performing distance computations on a mbatch of samples per iteration like mbatch while maintaining a lower centroid staleness compared with mbatch. The key idea of srmbatch is to leverage mbatch to frequently update centroids, while only allows the most recent assignment results of data points in the dataset to participate in centroid update. As such, srmbatch can achieve a high clustering quality due to less centroid staleness while keeping a fast convergence rate. Meanwhile, srmbatch still exposes massive parallelism to be efficiently implemented on multicore CPUs and many-core GPUs.

In this article, our contributions are as follows.

1) We propose the k-means srmbatch algorithm (Section III) to achieve a high clustering quality like km (Section III-C) and low computational costs like mbatch (Section III-B). Both two characteristics are proved by theoretical analysis and experimental results.
2) We theoretically analyze the centroid staleness of km, mbatch, and srmbatch. We prove by equations that the centroid staleness of srmbatch is lower than that of mbatch. We qualitatively analyze the relationship between the centroid staleness and the clustering quality to prove that lower centroid staleness leads to higher clustering quality.
3) We conduct extensive experiments to verify the efficiency of srmbatch. Experimental results show that srmbatch achieves a higher clustering quality, faster convergence speed, and almost the same massive degree of parallelism compared with mbatch.

This article is organized as follows. In Section II, we introduce the km algorithm and the mbatch algorithm in detail. In Section III, we put forward srmbatch and theoretically analyze its centroid staleness compared with km and mbatch. In Section IV, we organize experiments to compare the convergence performance among km, mbatch, and srmbatch.

II. Background

In this section, we analyze the advantages and the disadvantages of both km and mbatch. Before diving into the details of each algorithm, we emphasize the definition of the epoch which is used frequently in the following content. We define an epoch as a certain number of iterations within which all data points in the dataset have been traversed once in mathematical expectation.

Algorithm 1 Standard K-Means

```
Input  : $K$: number of clusters,
         $\vec{X}$: dataset,
         $N$: number of data points in the dataset $\vec{X}$,
         $T$: number of iterations.
Output: $\vec{C}$: cluster centroid set.
   /* 1, Initialization Step */
1  $\vec{C} \leftarrow K$ samples picked randomly from $\vec{X}$
2  $\vec{v} \leftarrow 0$ /* Per-iteration counter for current iteration */
3  $\vec{S} \leftarrow 0$ /* Per-iteration sum for current iteration */
4  for $i = 0$ to $T - 1$ do
       /* 2, Assignment Step */
5      for $j = 0$ to $N - 1$ do
           /* Function f for the closest centroid */
6          $c \leftarrow f(\vec{C}, \vec{X}[j])$
           /* Update per-iteration counter */
7          $\vec{v}[c] \leftarrow \vec{v}[c] + 1$
           /* Update per-iteration sum */
8          $\vec{S}[c] \leftarrow \vec{S}[c] + \vec{X}[j]$
       /* 3, Update Step */
9      for $j = 0$ to $K - 1$ do
           /* Update centroids */
10         $\vec{C}[j] \leftarrow \vec{S}[j]/\vec{v}[j]$
           /* Update per-iteration counter & sum */
11         $\vec{v}[j] \leftarrow 0$
12         $\vec{S}[j] \leftarrow 0$
```

A. Standard km

The standard km algorithm is widely adopted in real applications because of its high clustering quality and simple implementation. The detailed flow of km, as illustrated in Algorithm 1, consists of three steps: 1) the *initialization* step, where it picks K random centroids for the clusters (Line 1); 2) the *assignment* step, where each sample is assigned to its closest centroid by comparing squared Euclidean distances (Lines 6–8); and 3) the *update* step, where each centroid is recalculated to be the element-wise mean of the samples assigned to it (Line 10). The algorithm runs for a fixed number of iterations, or until the centroids become stable.

The detailed flow of km within two epochs is shown in Fig. 1. Based on the definition of the epoch, for km, because of its full batch iteration mode, an epoch is equivalent to an iteration. It's obvious in Fig. 1 that km accumulates samples into $\vec{S}$ and $\vec{v}$ within each epoch (Lines 7 and 8) and cleans up $\vec{S}$ and $\vec{v}$ at the end of the epoch (Lines 11 and 12) after calculating new centroids (Line 10). As a result, only each sample's most recent assignment is accumulated into $\vec{S}$ and $\vec{v}$

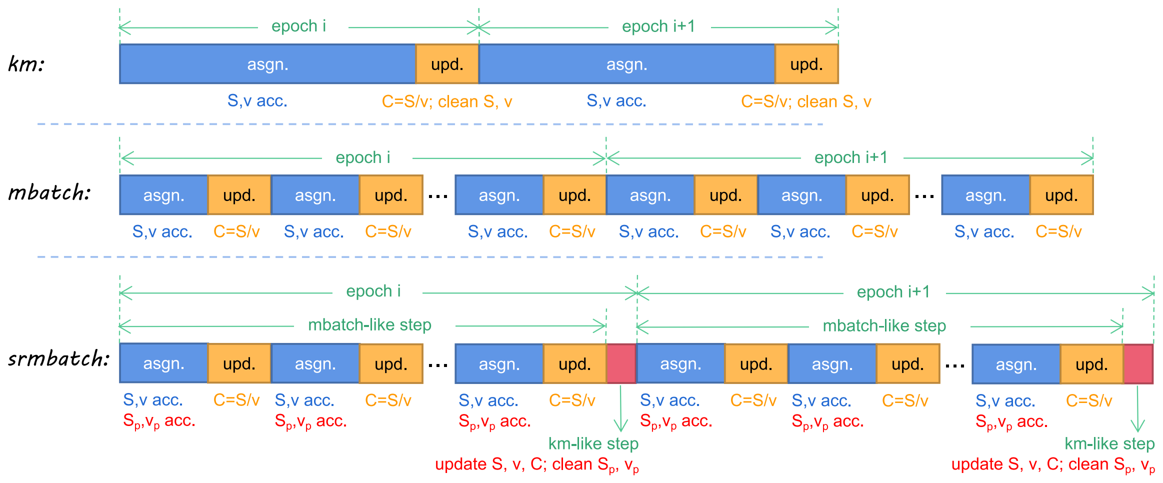


Fig. 1. Detailed flow of km, mbatch, and srmbatch within two sequential epochs. (acc. = accumulate samples, asgn. = assignment, upd. = update.)

and then contributes to the centroids at the end of this epoch, which introduces no staleness into the centroids.

The **advantage** of km is that km is an efficient clustering algorithm that can converge to a low loss because of its full-batch iteration mode with zero centroid staleness.

The **disadvantage** is that km inevitably takes either a long time or a large number of computational resources to calculate distances between a full batch of samples and all centroids before updating centroids once. This issue becomes worse when dealing with large datasets.

B. Mini-Batch K-Means

The mbatch km algorithm applies mbatch optimization to the standard km clustering [4]. Essentially, instead of updating centroids once within each epoch, mbatch updates centroids for each mbatch so it can update centroids multiple times within one epoch.

Algorithm 2 illustrates the flow of the mbatch algorithm. The main differences between mbatch and km are as follows. Before each iteration, a certain number of samples are randomly picked to form a mbatch (Line 5). Then, it calculates distances between samples in the mbatch with all centroids and assigns each sample to its closest centroid (Lines 7–9). After all samples in the mbatch have been assigned, each cluster centroid is updated by the mean of all the samples assigned to it, namely, all the samples accumulated in $\vec{S}[j]$ (j is the corresponding index of each centroid) (Line 11).

As Fig. 1 shows, the $\vec{S}$ and $\vec{v}$ keep accumulating samples during all iterations in mbatch (Lines 7–9). Because mbatch uses random sampling with replacement, the whole dataset is mathematically expected to be traversed once in each epoch, namely, N/B iterations, according to the definition of the epoch. Since $\vec{S}$ and $\vec{v}$ are never cleaned up between iterations or epochs, assignment results from previous epochs are always kept in $\vec{S}$ and $\vec{v}$, which introduces staleness into centroids.

The **advantage** of mbatch is that mbatch optimization leads to a faster convergence rate. The mbatch of samples are picked randomly with replacement from the dataset, so they are supposed to have the same statistic characteristics as the full batch, indicating that mbatch will converge to some

Algorithm 2 Mini-Batch K-Means

```
Input  : $K$: number of clusters,
         $\vec{X}$: dataset,
         $N$: number of data points in dataset $\vec{X}$,
         $B$: mini-batch size,
         $T$: number of iterations.
Output: $\vec{C}$: cluster centroid set.
   /* 1, Initialization Step */
1  $\vec{C} \leftarrow K$ samples picked randomly from $\vec{X}$
2  $\vec{v} \leftarrow 0$ /* Iterative counter */
3  $\vec{S} \leftarrow 0$ /* Iterative sum */
4  for $i = 0$ to $T-1$ do
       /* 2, Assignment Step */
5      $\vec{M} \leftarrow B$ samples picked randomly from $\vec{X}$
6      for $j = 0$ to $B-1$ do
           /* Function f for the closest
              centroid */
7          $c \leftarrow f(\vec{C}, \vec{M}[j])$
           /* Update iterative counter */
8          $\vec{v}[c] \leftarrow \vec{v}[c] + 1$
           /* Update iterative sum */
9          $\vec{S}[c] \leftarrow \vec{S}[c] + \vec{M}[j]$
       /* 3, Update Step */
10     for $j = 0$ to $K-1$ do
           /* Update centroids */
11         $\vec{C}[j] \leftarrow \vec{S}[j]/\vec{v}[j]$
```

extent. The distance calculations associated with a mbatch are significantly fewer than that with a full batch, so mbatch enjoys a faster convergence rate.

The **disadvantage** of mbatch is that mbatch can converge to a lower quality clustering solution, because mbatch uses each data point's multiple assignment results, including the most recent one and the stale ones, to update centroids. Intuitively, stale ones may not be the same as the most recent one but all of them are kept in sum $\vec{S}$ and counter $\vec{v}$ (Lines 7–9) that are used to update centroids for the next iteration (Line 11), so it introduces centroid staleness. More mathematical analysis is in Section III-C.

III. Staleness-Reduction Mini-Batch K-Means

In this section, we first propose the k-means srmbatch algorithm, which aims at achieving both low computational costs and high clustering quality. The key idea of srmbatch is to mainly follow mbatch to update centroids with a mbatch of samples while constraining the staleness of centroids to stay low; that is, there are no more than 2 most recent assignments of each data point in the dataset contribute to centroids. Then, we present the concrete algorithm and related analysis.

A. srmbatch Algorithm

Algorithm 3 illustrates the flow of srmbatch, which is similar to mbatch regarding more frequent centroid updating than km, and which is similar to km regarding the low staleness, as shown in Fig. 1. In the following, we present the detailed flow of srmbatch, including three steps: initialization, mbatch-like, and km-like.

1) Step 1 (Initialization Step): The dataset $\vec{X}$ is shuffled and partitioned to N_b mbatchs (Lines 1 and 2), all of which will be sequentially used for centroid updating.

Each iteration processes a mbatch, with N_b iterations per epoch. According to the definition of epoch in Section II, in srmbatch, an epoch equals exactly N_b iterations because the dataset is traversed once during N_b iterations. The motivation behind our sampling approach is to guarantee that each data point in a dataset is only assigned once in srmbatch in each epoch, which is consistent with km that scans the entire dataset for each iteration/epoch. The cluster centroid set $\vec{C}$ is chosen to be K samples picked randomly from $\vec{X}$ (Line 3).

2) Step 2 (mbatch-Like Step): The srmbatch algorithm is iteratively evaluated in E epochs (Line 8). In each epoch, N_b mbatchs are scanned by srmbatch, one mbatch per iteration (Line 9). In each mbatch of B samples, srmbatch processes the kth sample as follows. First, we determine the closest cluster, whose index is c, by comparing distances between the sample and K cluster centroids $\vec{C}$ (Line 11). Second, we update the cth elements of iterative counter $\vec{v}$ and sum $\vec{S}$ using the kth sample $\vec{M}_j[k]$ (Line 12). Third, we update the cth elements of per-epoch counter $\vec{v}_p$ and sum $\vec{S}_p$ using the kth sample $\vec{M}_j[k]$ (Line 13). After processing a mbatch, srmbatch recalculates $\vec{C}$ to be $\vec{S}$ divided by $\vec{v}$ element-wise (Lines 14 and 15). The updated $\vec{C}$ would be immediately used in the next mbatch, similar to mbatch.

3) Step 3 (km-Like Step): At the end of an epoch, srmbatch updates cluster centroids $\vec{C}$ to constrain the centroid staleness (Lines 16–19). Particularly, $\vec{C}$ is recalculated to be $\vec{S}_p$ divided by $\vec{v}_p$ in an element-wise approach (Line 17), and $\vec{S}$ and $\vec{v}$ are set to $\alpha * e$ multiplied by $\vec{S}_p$ and $\vec{v}_p$, respectively (Line 18). It guarantees that the new centroid set for the next epoch only contains the most recent assignments from the current epoch and that the centroid set during the next epoch contains no more than two most recent assignments of each data point. Finally, $\vec{S}_p$ and $\vec{v}_p$ are set to 0 for the next epoch (Line 19). This centroid update is the reason why, at any time, there are

Algorithm 3 Staleness-Reduction Mini-Batch K-Means

```
Input  : $K$: number of clusters,
         $B$: mini-batch size,
         $\vec{X}$: dataset,
         $N$: number of data points in dataset $\vec{X}$,
         $E$: number of epochs,
         $\alpha$: hyperparameter.
Output: $\vec{C}$: cluster centroid set.
   /* 1, Initialization Step */
1  $N_b \leftarrow N/B$ /* number of mini-batches in a dataset */
2  $\vec{M}_i (i = 0, 1, 2, \ldots, N_b - 1) \leftarrow$ Shuffle and partition $\vec{X}$ to $N_b$ mini-batches
3  $\vec{C} \leftarrow K$ samples picked randomly from $\vec{X}$
4  $\vec{v} \leftarrow 0$ /* Iterative counter */
5  $\vec{S} \leftarrow 0$ /* Iterative sum */
6  $\vec{v_p} \leftarrow 0$ /* Per-epoch counter for current epoch */
7  $\vec{S_p} \leftarrow 0$ /* Per-epoch accumulator for current epoch */
8  for $e = 1$ to $E$ do
       /* 2, mbatch-like Step */
9      for $j = 0$ to $N_b - 1$ do
           /* 1) Assignment Step */
10         for $k = 0$ to $B - 1$ do
               /* Function $f$ for the closest centroid */
11             $c \leftarrow f(\vec{C}, \vec{M}_j[k])$
               /* Update iterative counter & sum */
12             $\vec{v}[c] \leftarrow \vec{v}[c] + 1, \vec{S}[c] \leftarrow \vec{S}[c] + \vec{M}_j[k]$
               /* Update per-epoch counter & sum */
13             $\vec{v_p}[c] \leftarrow \vec{v_p}[c] + 1, \vec{S_p}[c] \leftarrow \vec{S_p}[c] + \vec{M}_j[k]$
           /* 2) Update Step */
14         for $k = 0$ to $K - 1$ do
15             $\vec{C}[k] \leftarrow \vec{S}[k]/\vec{v}[k]$ /* Update centroids */
       /* 3, km-like Step (Re-update Step) */
16     for $j = 0$ to $K - 1$ do
17         $\vec{C}[j] \leftarrow \vec{S_p}[j]/\vec{v_p}[j]$ /* Update centroids */
           /* Re-update iterative counters & sums */
18         $\vec{v}[j] \leftarrow \alpha * e * \vec{v_p}[j], \vec{S}[j] \leftarrow \alpha * e * \vec{S_p}[j]$
           /* Re-update per-epoch counters & sums */
19         $\vec{v_p}[j] \leftarrow 0, \vec{S_p}[j] \leftarrow 0$
```

at most two assignment result of each data point in the dataset that contribute to the centroid set, hence reducing staleness, as specified in the introductory summary of Section III.

TABLE I

COMPLEXITY OF COMPUTATIONAL OPERATIONS IN SRMBATCH (ALGORITHM 3) WITHIN ONE EPOCH. (D IS THE NUMBER OF DIMENSIONS OF SAMPLES)

Operations	Complexity
Distance calculation (Line 11) (Ignore square root calculations)	$= N * K * D$ times of subtraction $+ N * K * D$ times of multiplication $+ N * K * (D-1)$ times of addition
Iterative counter & sum updating (Line 12)	$N + N * D$ times of addition
Per-epoch counter & sum updating (Line 13)	$N + N * D$ times of addition
Centroid updating (Lines 15)	$N_b * K * D$ times of division
Centroid updating (Lines 17)	$K * D$ times of division
Iteration counter & sum re-updating (Line 18)	$2 * K + 2 * K * D$ times of multiplication
Per-epoch counter & sum re-updating (Line 19)	$K + K * D$ times of value assignment

The coefficients α and e (Line 18) are used to tune the weight of $\vec{S}_p$ and $\vec{v}_p$. The intuition behind introducing e as a coefficient is that the cluster centroids in srmbatch should tend to become stabler when e increases like mbatch in which $\vec{S}$ and $\vec{v}$ keep accumulating samples to guarantee convergence stability. Hyperparameter α is introduced to manually control the proportion of assignment results from the last epoch in the current epoch's $\vec{S}$, $\vec{v}$, and $\vec{C}$. A larger α means more contribution that the last epoch makes to the centroid set in the current epoch.

4) Comparison to mbatch: As illustrated in Fig. 1 and Algorithm 3, the flows of srmbatch and mbatch are similar to each other. Their differences are shown as follows.

First, they have different sampling mechanisms. srmbatch shuffles the dataset at the beginning and samples data points sequentially to build a mbatch, while mbatch samples data points randomly from the dataset to build a mbatch.

Second, they have different centroid update mechanisms. Compared to mbatch that only updates centroids at the end of each iteration, srmbatch adds one more step, i.e., km-like step, that uses the most recent assignments at the end of current epoch to update cluster centroids before the next epoch begins. As such, the next epoch can have low centroid staleness as km because km-like step deprecates stale assignment results.

B. Theoretical Analysis of Computational Costs

We mainly analyze the computational costs of km, mbatch, and srmbatch on distance calculation for centroid update in one iteration. It's worth noting that for all three algorithms, not only distance calculation but also other operations like centroid updating, the iterative counter($\vec{v}$) updating, the iterative sum($\vec{S}$) updating, and so on cause computational costs. Table I shows the complexity of all computational operations in srmbatch within one epoch. We can find that compared with the cost of distance calculation, the cost of other operations can be ignored. For km and mbatch, operations beyond distance calculation are less so can be ignored as well.

For km, it always updates centroids with a full batch of samples, so it calculates distances $N * K$ times to update centroids once. For mbatch and srmbatch, we compare them under the same mbatch size. For mbatch, it always updates centroids with a mbatch of samples so it calculates distances $B * K$ times to update centroids once. For srmbatch, it calculates distances $B * K$ times to update centroids once as well because it uses a similar mbatch strategy. When B is far smaller than N, (1) always holds which means mbatch and srmbatch take far less computational costs to update centroids once than km does

$$K * B \ll N * B. \tag{1}$$

C. Theoretical Analysis of Clustering Quality

In this section, our goal is to theoretically compare the centroid staleness of km, mbatch, and srmbatch, so as to verify srmbatch's efficiency in reducing centroid staleness compared with mbatch. In the following, we first give the formal definition of centroid staleness, followed by the theoretical analysis, second compare the centroid staleness of km, mbatch, and srmbatch, third analyze the relation between the centroid staleness and the clustering quality, and finally, make discussions about the hyperparameter α. for srmbatch

1) Definition of Centroid Staleness:

Definition 1 (Epoch Gap): The epoch gap of an assignment result is defined as the number of epochs between where the assignment result is generated and where it contributes to the centroid set. $G_{i,j}$ means the epoch gap of an assignment result that is generated in one iteration of the ith epoch and that contributes to the centroid set in one iteration of the jth epoch as shown in the following equation:

$$G_{i,j} = j - i. \tag{2}$$

Given that assignment results from the same ith epoch have the same epoch gap $j - i$ when they contribute to the centroid set in the jth epoch, $W_{i,j}$ is the sum of epoch gaps of all the $N_{i,j}$ assignment results that are from the ith epoch and that contribute to centroid set in the jth epoch, as shown in (3). $N_{i,j}$ has different forms for different algorithms and we analyze it in Section III-C2

$$W_{i,j} = \sum_{j=1}^{N_{i,j}} G_{i,j} = \sum_{j=1}^{N_{i,j}} (j - i) = N_{i,j}(j - i). \tag{3}$$

Definition 2 (Centroid Staleness): Centroid staleness is a scalar and is defined to be the average epoch gap that the assignment results contributing to the centroid set in a certain iteration. The centroid set calculated out in each iteration has its own centroid staleness. $L_{x,y}$ means the centroid staleness that the centroid set from the xth iteration of the yth epoch has, as shown in (4). The reason why i increases from 1 to $j-1|_{j=y}$ instead of to $j|_{j=y}$ is that assignment results from the yth epoch do not introduce centroid staleness to the centroid set in the same yth epoch ($G_{i,j}|_{i=j=y} = 0$, $W_{i,j}|_{i=j=y} = 0$)

$$L_{x,y} = \sum_{i=1}^{i=j-1} P_{i,j}(x)W_{i,j}\Big|_{j=y}. \tag{4}$$

Equation (4), $P_{i,j}(x)$ is the proportion that each assignment result from the ith epoch takes among all assignment results that contribute to the centroid set in the xth iteration of the jth epoch. It's a function of x and has different forms for different algorithms. We analyze $P_{i,j}(x)$ in Section III-C2. The reason why we do not introduce an independent weight coefficient for each assignment result from the ith epoch separately is that for all three algorithms, assignment results from the same epoch have the same proportion.

2) Comparison of L_e for km, mbatch, and srmbatch:

a) Evaluating $L_{x,y}$ for km: The km algorithm performs centroid update with a full batch of samples ($N = B$) in each epoch, indicating that each data point is sampled exactly once to update $\vec{S}$ and $\vec{v}$ in each epoch (Lines 7 and 8 in Algorithm 1). Moreover, $\vec{S}$ and $\vec{v}$ are cleaned at the end of each epoch (Lines 11 and 12), which means old assignment results from the first, second, $(j-1)$th epochs won't be kept in $\vec{S}$ and $\vec{v}$ in the jth epoch. So there exists in the following equation for $P_{i,j}(x)$:

$$P_{i,j}(x) = \begin{cases} = 0, & i \le j-1 \\ = \dfrac{1}{N}, & i = j. \end{cases} \tag{5}$$

For km, $N_{i,j} = N (i \le j)$ according to the definition of epoch. Substituting (5) into (4), we can achieve $L_{x,y}$ in the following equation:

$$L_{x,y} = 0. \tag{6}$$

b) Evaluating $L_{x,y}$ for mbatch: The mbatch algorithm performs the centroid update at the end of each iteration with a mbatch of samples which are randomly picked with replacement from the entire dataset. According to the definition of the epoch, in mbatch, each data point is mathematically expected to be sampled once to update $\vec{S}$ and $\vec{v}$ within an epoch (Lines 8 and 9 in Algorithm 2). $P_{i,j}(x)$ is as shown in (7). B is the mbatch size, $(j-1)N$ is the number of assignment results from the first $\sim$ $(j-1)$th epochs, and xB is the number of assignment results from the first $\sim$ xth iterations in the jth epoch

$$P_{i,j}(x) = \frac{1}{(j-1)N + xB}. \tag{7}$$

For mbatch, $N_{i,j} = N (i \le j)$ as well according to the definition of the epoch. Therefore, there exists in the following equation for $L_{x,y}$:

$$\begin{aligned} L_{x,y} &= \frac{N}{(y-1)N + xB}\frac{y(y-1)}{2} \\ &\ge \frac{(y-1)N}{(y-1)N + xB}, \quad \text{when } y \ge 2. \end{aligned} \tag{8}$$

c) Evaluating $L_{x,y}$ for srmbatch: The srmbatch algorithm recalculates the centroids to be $\vec{S}_p$ divided by $\vec{v}_p$ in an element-wise manner and reupdates $\vec{S}$ and $\vec{v}$ after each epoch ends (Lines 17 and 18 in Algorithm 3), so that in each iteration, srmbatch always update the centroid set with assignment results from no more than two latest epochs instead of from all previous epochs like mbatch. $P_{i,j}(x)$ for srmbatch satisfies as follows:

$$P_{i,j}(x) = \begin{cases} = 0, & i \le j-2 \\ = \dfrac{1}{\alpha(j-1)N + xB}, & i = j-1 \text{ or } j. \end{cases} \tag{9}$$

Equation (9), $\alpha(j-1)N$ is the value of $N_{i,j}|_{i=j-1}$, namely, the number of assignment results from the $(j-1)$th epoch. The reason why $N_{i,j}|_{i=j-1}$ doesn't equal N is that srmbatch multiplies the assignment results from the $(j-1)$th epoch in the km-like step at the end of the $(j-1)$th epoch with the coefficient[2] $\alpha * (j-1)$. So $L_{x,y}$ satisfies as follows:

$$\begin{aligned} L_{x,y} &= P_{i,j}(x)W_{i,j}\Big|_{i=j-1,j=y} \\ &= P_{i,j}(x)N_{i,j}(j-i)\Big|_{i=j-1,j=y} \\ &= \frac{\alpha(y-1)N}{\alpha(y-1)N + xB} \\ &\le \frac{(y-1)N}{(y-1)N + xB}, \quad \text{when } 0 \le \alpha \le 1. \end{aligned} \tag{10}$$

d) srmbatch versus mbatch: Intuitively, srmbatch has lower $L_{x,y}$ than mbatch because as Fig. 1 shows, srmbatch deprecates assignment results with high epoch gap in $\vec{S}$ and $\vec{v}$ by km-like step at the end of each epoch, while mbatch never cleans up $\vec{S}$ and $\vec{v}$, namely, never drops assignment results with high epoch gap for centroid update. Also, from (8) and (10), we can find that if both algorithms share the same B, $L_{x,y}$ for srmbatch will always be less than or equal to $L_{x,y}$ for mbatch when $0 \le \alpha \le 1$ and $y \ge 2$ hold. In practice, the loss of both mbatch and srmbatch seldom converges within two epochs. So for most cases, srmbatch has lower $L_{x,y}$ with $0 \le \alpha \le 1$ than mbatch does.

e) srmbatch versus km: We can find that when $B \ne N$ for srmbatch, srmbatch and km do not share the same number of iterations within one epoch, so $L_{x,y}$ for srmbatch cannot be compared with $L_{x,y}$ directly with one-to-one correspondence about x, y. However, we can find that $L_{x,y}$ for km is always zero, while $L_{x,y}$ for srmbatch can be close to zero as well when $\alpha \approx 0$ based on (10). km reaches a high clustering quality because $L_{x,y} = 0$ which means there is no stale assignment result impacting its clustering quality. So we consider that srmbatch with $\alpha \approx 0$, namely, $L_{x,y} \approx 0$, can reach a similarly high clustering quality as well because nearly all stale assignment results are deprecated.

3) Relation Between $L_{x,y}$ and Loss: We seek lower $L_{x,y}$ for lower loss. To better analyze the relation between $L_{x,y}$ and the loss, we define $R_{x,y}$ as a set of assignment results that contribute to the centroid set in the xth iteration of the

[2]Both $(j-1)$ here and e in Line 18, Algorithm 3 refer to the same index of the epoch.

yth epoch, and dismantle $R_{x,y}$ as a series of assignment result sets $\{R_{x,y}^{(i)} | 1 \leq i \leq y\}$. $R_{x,y}^{(i)}$ contains assignment results that are generated in the ith epoch and that contribute to the centroid set in the xth iteration of the yth epoch. Because of the definition of epoch, $R_{x,y}^{(i)} (1 \leq i \leq y - 1)$ contains one assignment result of each data point in the dataset. Since all three algorithms iteratively converge because of the nature of km algorithms [5], $R_{x,y}^{(i_1)}$ is generated based on a centroid set with higher loss than $R_{x,y}^{(i_2)}$ when $i_1 < i_2$, namely, when $R_{x,y}^{(i_1)}$ is staler than $R_{x,y}^{(i_2)}$. In other words, $R_{x,y}^{(i_1)}$ deviates from the accurate assignment results more than $R_{x,y}^{(i_2)}$ when $i_1 < i_2$. Besides, when the gap between i_1 and i_2 increases, the gap between the losses of the corresponding centroid sets increases as well. As a result, for $R_{x,y}$, each $R_{x,y}^{(i)}$ causes a centroid set which is calculated out in the xth iteration of the yth epoch shifts from the one with lower loss to the one with higher loss, and the smaller the i is, the severer the centroid shift that $R_{x,y}^{(i)}$ causes is. We use epoch gap to measure how stale an assignment result is when it contributes to the centroid set, and we assume that the effect of stale assignment results on loss is linear,[3] then the centroid staleness $L_{x,y}$, namely, the average epoch gap for assignment results that contribute to the centroid set in the xth iteration of the yth epoch, can measure how severe the overall loss increase caused by centroid shift is.

D. Discussion on the Case $\alpha = 0$

From (9), we can find that if $\alpha = 0$, $L_{x,y}$ reaches the lowest value in srmbatch, meaning that most centroid staleness is removed. It seems tempting and seemingly will lead to the best convergence rate and quality. However, compared with $\alpha \neq 0$, $\alpha = 0$ leads to a more severe sampling distortion problem with which the mbatch optimization comes along.

In particular, $\vec{S}$ and $\vec{v}$ are cleaned at the end of each epoch. So if $\alpha = 0$, a mbatch of samples that are accumulated in $\vec{S}$ and $\vec{v}$ in the first iteration of the next epoch will completely decide centroids by $\vec{S}/\vec{v}$. However, the statistical characteristics of a mbatch of samples usually can not represent the whole dataset. So a new centroid set calculated from just a mbatch of samples may not be able to reduce loss. Not only the first iteration of each epoch has this problem, but also the second, the third, ... until $\vec{S}$ and $\vec{v}$ have accumulated a certain amount of samples. This process repeats at the beginning of each epoch, which may affect the convergence rate and quality.

On the contrary, none-zero α can ease this problem. According to Line 18, Algorithm 3, if $\alpha \neq 0$, the weight of assignment results from the last epoch increases when e increases no matter how tiny the α is. It makes sense because the cluster centroids in srmbatch should tend to become stabler when e increases like mbatch in which $\vec{S}$ and $\vec{v}$ keep accumulating samples to guarantee convergence stability.

It's hard to judge whether staleness or sampling distortion has a more severe influence on convergence rate in real implementation for mbatch iteration mode, so srmbatch introduces one tuning knob α for the user.

[3]we take a more complex model like a binomial model or an exponential model as future work.

IV. Experiment

In this section, we compare km, mbatch, and srmbatch on three aspects–the hardware efficiency (Section IV-B), the statistical efficiency (Section IV-C), and the end-to-end comparison (Section IV-D). Experiments on the first aspect show that srmbatch achieves almost the same massive degree of parallelism compared with mbatch, and experiments on the other two aspects show that srmbatch achieves a high clustering quality like km and low computational costs like mbatch. We also explore the effect of hyperparameter α on srmbatch (Section IV-E) and give guidance on how to tune α based on experiments. We finally extend our algorithm with k means++ (Section IV-F) to show that our algorithm still works well when it's combined with the other orthogonal optimization for km.

A. Experimental Setup

1) Hardware Platform: We run our CPU code on a 10-core Intel CPU i9-10900X @ 3.70 GHz, with 94 GB/s memory bandwidth and a 19.25 MB L3 cache. The SIMD[4] code leverages the AVX-512 instruction set. We run the GPU code on an NVIDIA Tesla V100 GPU @1.245 GHz with 5120 compute unified device architecture (CUDA) cores and 900 GB/s memory bandwidth.

2) Dataset: We run our experiments with four real-world datasets: Sift [7], Gist [7], Poker [8], and Mnist8m [9], as shown in Table II. These datasets cover a wide range of data points and are representative of clustering tasks. All four datasets are normalized in each dimension by the min-max normalization method.

3) Centroid Initialization: Since the km algorithm itself is sensitive to the initial centroids, for the sake of fairness, we use the same initial centroids in all the experiments for each dataset. The initial centroids are chosen to be K samples randomly picked from the dataset as the related works ([4], [10], [11], [12]) do.

4) Comparison Methodology: To demonstrate the efficiency of srmbatch, we compare the hardware efficiency (Section IV-B), statistical efficiency (Section IV-C), and end-to-end efficiency (Section IV-D) of srmbatch with that of mbatch and km.

B. Hardware Efficiency: Time for Each Epoch

In this section, we demonstrate the hardware efficiency of srmbatch by comparing with mbatch[5] and km on the Sift dataset with $K = 32$, in terms of the time for an epoch with varying mbatch size. For experiments on CPUs, we implement

[4]SIMD stands for "Single Instruction, Multiple Data, and refers to functionality that a lot of computer hardware has for operating on multiple numbers at once [6].

[5]We introduce two mbatch variants. The first one, labeled mbatch (rand), is the standard implementation of mbatch. It randomly picks K samples with replacement for each mbatch, as shown in Algorithm 2. The other one, labeled mbatch (seq), is the variant that sequentially chooses K samples for each mbatch. We introduce mbatch (seq) for two reasons. First, it's used to show that mbatch (rand) suffers from random memory access and random number generation. Second, the only difference between mbatch (seq) and srmbatch is that srmbatch has a staleness reduction part so the comparison between them can show the overhead of the staleness reduction part of srmbatch.

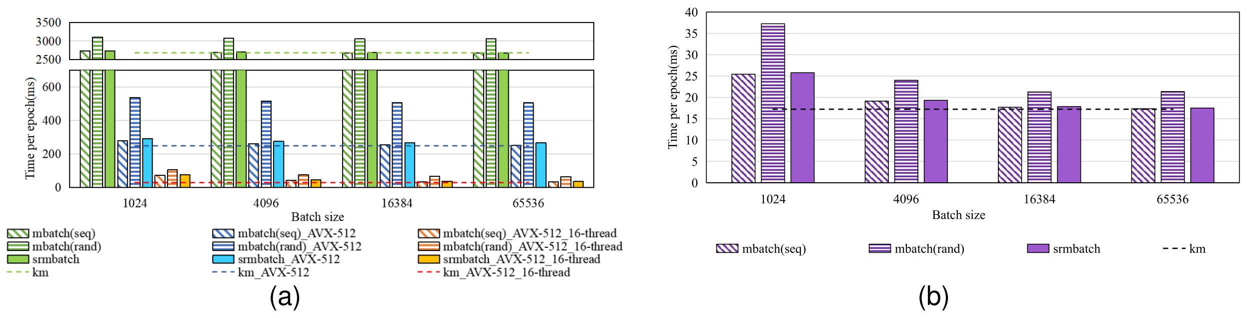


Fig. 2. Comparison of hardware efficiency: time per epoch. (a) CPU. (b) GPU.

TABLE II
EVALUATED DATASETS

Data sets	Samples	Features	Clusters
Sift [7]	1,000,000	128	100
Gist [7]	1,000,000	960	100
Poker [8]	1,000,000	10	10
Mnist8m [9]	81,000,000	784	10

all algorithms in three different configurations which are about the parallelism potentials of the system. The first one only uses O3 compiler option[6]; the second uses O3 and SIMD (AVX-512)[7] for 32-dimension parallelism; the third uses O3, SIMD (AVX-512) for 32-dimension parallelism, and multithread[8] (16-thread) technique for 16-sample parallelism.

For experiments on GPUs, we implement all algorithms with CUDA. The most time-consuming kernel for all algorithms is to calculate the distance between each sample in a mini/full-batch and each centroid in the centroid set. To fully utilize GPU computing power, this kernel includes 256 thread blocks with 32 warps per block (32 threads per warp). Every thread block calculates distances between one sample and 32 centroids concurrently.[9] Every warp in a thread block calculates the distance between one sample and one centroid. Every thread in a warp calculates the squared distance between 1-D of a sample and the same dimension of a centroid.[10]

Time per epoch means how long for a certain algorithm to run over an epoch, rather than over the whole clustering process. Typically, we run 50 epochs and then get the average time for each epoch. Fig. 2 compares the hardware efficiency of srmbatch, mbatch, and km. There is a gap (600–2500 ms) for the vertical axis in Fig. 2. We hide 600–2500 ms of the vertical axis so that all results are shown on an appropriate scale. Based on the experiments, we have three observations:

First, srmbatch takes roughly the same time per epoch as mbatch (seq) and km regardless of AVX-512/16-thread code on the CPUs or code on the GPUs, indicating that srmbatch exposes massive calculation parallelism and that the overhead of the staleness reduction part of srmbatch is trivial. To be specific, for one epoch, the staleness reduction part (Line 13, 17–19, Algorithm 3) needs around 1.3E8 floating point operations (FLOPs) and others need around 1.2E10 FLOPs under the given experimental situations. What's more, the acceleration effect of parallelism is almost stable. To be specific, the standard deviation around the mean of time per epoch is no more than 6 ms for CPU-based experiments and no more than 50 us for GPU-based experiments.

Second, *mbatch (seq)* takes less time per epoch than *mbatch (rand)* on both CPUs and GPUs, indicating the random sampling strategy is more memory-unfriendly. The underlying reason is that random sampling leads to random memory access, which causes memory bandwidth unutilized and thus increases the time per epoch. Besides, as Fig. 2(a) shows, "AVX-512" achieves high calculation parallelism, while only slightly relieving the low bandwidth utilization issue caused by random sampling. We can learn it from two phenomena. First, *mbatch(rand)_AVX-512* needs obviously lower time per epoch than that of *mbatch(rand)*, when other configurations keep. Second, the time difference between *mbatch(rand)_AVX-512* and *mbatch(seq)_AVX-512* is just lower than the time difference between *mbatch(rand)* and *mbatch(seq)* a little. What's more, the multithread tech on CPUs and the multicore tech on GPUs are powerful in increasing both calculation parallelism and random access bandwidth. As a result, we can see that the time difference between *mbatch(rand)* and *mbatch(seq)* decreases a lot when multithread/core tech is used and other configurations keep.

Third, as Fig. 2(b) shows, too small mbatch size like 1024 hurts the performance of srmbatch and mbatch on the GPUs compared with km's performance. It is because (1) a small mbatch size does not expose enough parallelism to saturate thousands of cores on the GPU, while km allows GPUs to fully leverage their high parallelism, and (2) a small mbatch size leads to multiple times of centroid update per epoch, while km only updates centroids once per epoch. Leveraging all three optimization methods on CPUs leads to high calculation parallelism that can decrease the time for distance calculation a lot, so the time for centroid update cannot be ignored, as shown in Fig. 2(a).

[6]O3 is the highest optimization level of the GCC compiler. It includes revising the order of instructions, incomprehensive data parallelism, and so on. It is implemented totally by the compiler, so it is friendly for users while its optimization is unelaborate.

[7]SIMD is short for "single instruction, multiple data." It's widely used for data parallelism and it requires developers to use the specific instruction set like AVX-512.

[8]Multithread is another widely used technique for not only data parallelism but also task parallelism. It requires developers to use a specific library like pthreads library.

[9]We use the for-loop to handle the case that the number of centroids is more than 32 for the following experiments.

[10]The for-loop is used to handle the case that the number of dimensions is more than 32 for the following experiments.

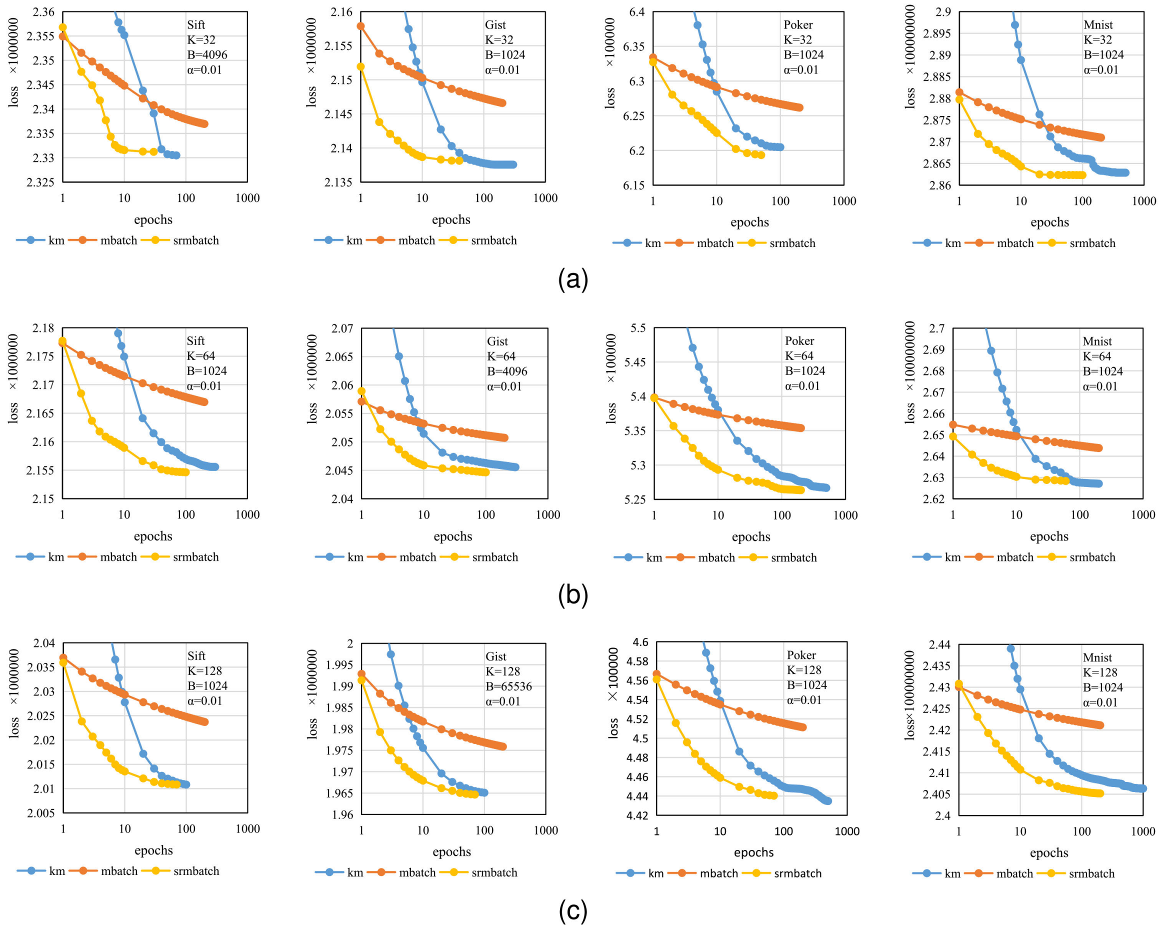


Fig. 3. Comparison of statistical efficiency: loss versus epoch. (a) $K = 32$. (b) $K = 64$. (c) $K = 128$.

C. Statistical Efficiency: Loss versus Epoch

In this section, we examine the statistical efficiency of srmbatch and compare it with km and mbatch. Fig. 3 illustrates the comparison of all four datasets with $K \in \{32, 64, 128\}$, in terms of loss (within-cluster sum of square error) versus epoch.[11] We take the experimental results with $K \in \{32, 64, 128\}$ as representative results for the small/medium/large centroid set, and the other number of clusters leads to similar observations though we do not show them. For each dataset, we choose the most suitable mbatch size from $\{1024, 4096, 16\,384, 65\,536\}$ for mbatch to reach the best clustering solution, while the km algorithm does not have a hyperparameter to tune. For srmbatch, we let the mbatch size of srmbatch be equal to the one that works best for mbatch on each dataset, and set alpha $= 0.01$ of which the efficiency is proved in experiments in Section IV-E.

We make the following observations. First, srmbatch always reaches an excellent clustering solution, in terms of loss, quite similar to a solution that km can achieve within the error range $(-0.15\%, 0.18\%)$, indicating that srmbatch has high clustering quality like km. By the way, it should be noted that the km algorithm gives not a global solution but a local one and there also exists an error in the floating-point calculation so srmbatch may sometimes achieve slightly lower loss than km. Second, srmbatch needs significantly fewer epochs to converge, compared with both km and mbatch. The comparison between srmbatch and km verifies that srmbatch has low computational costs for each time of centroid update because centroids always update with a mbatch of samples like mbatch instead of the full-batch samples like km. The comparison between srmbatch and mbatch indicates the importance of eliminating the stale assignment results from centroids. Although both srmbatch and mbatch use mbatch to update centroids, srmbatch has lower centroid staleness which let it faster reach a low loss.

[11]Regarding an epoch, either km or srmbatch processes each data point in the dataset exactly once, while mbatch processes N/B mbatchs of samples, whose expectation is equal to that of the entire dataset.

D. End-to-End Comparison: Loss versus Time

In this section, we validate that srmbatch outperforms km and mbatch in terms of end-to-end performance on the GPUs for high experiment efficiency with large datasets. The configurations are the same as those in Section IV-C. Fig. 4 illustrates the training loss that varies with the elapsed time for all three algorithms. We observe that srmbatch can converge significantly faster than mbatch, mainly because srmbatch needs a fewer number of epochs to converge to the same clustering solution. Compared with mbatch, srmbatch can converge 40×–130× faster, when reaching the same target loss.

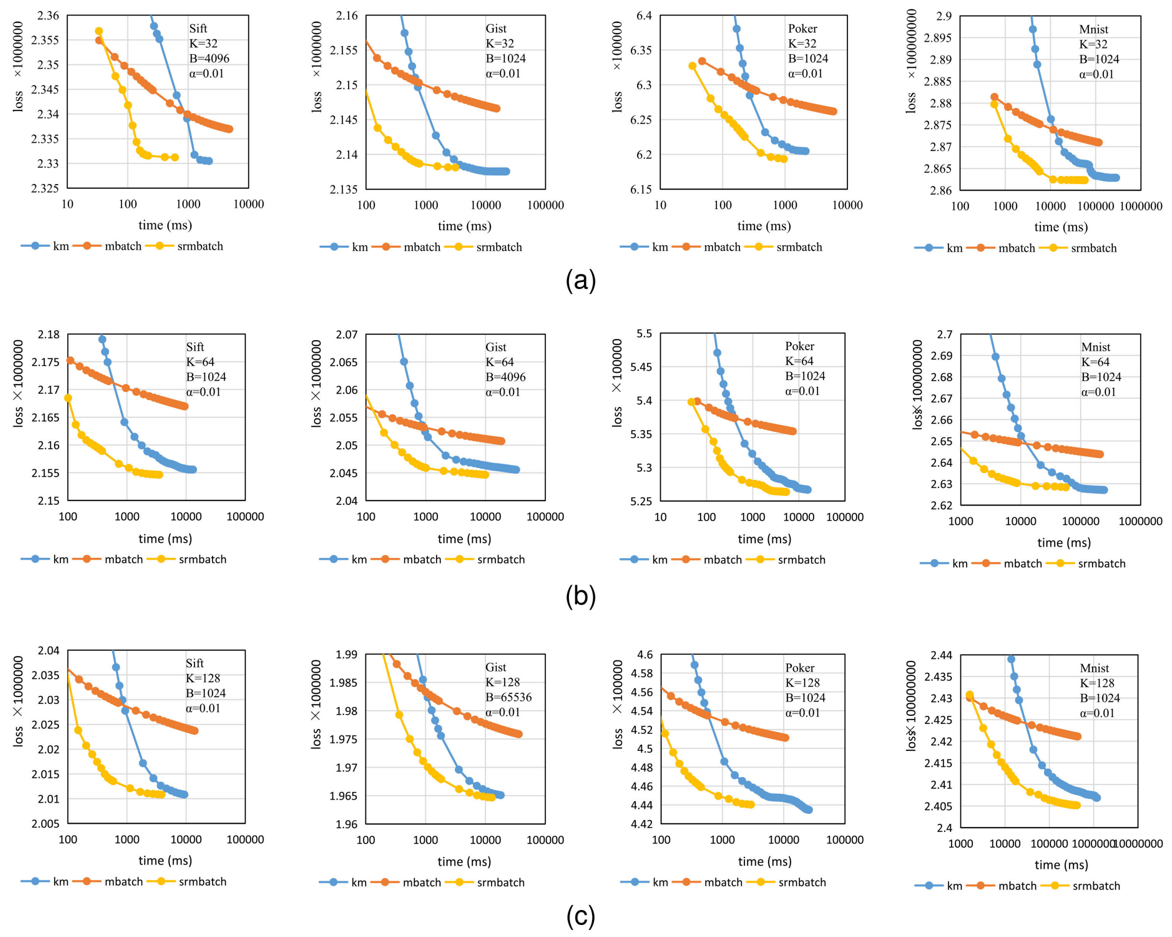


Fig. 4. End-to-end comparison: loss versus time. (a) $K = 32$. (b) $K = 64$. (c) $K = 128$.

Moreover, srmbatch can reach 0.2%–1.7% lower final loss than that of mbatch.

E. Effect of Hyperparameter α

In this section, we examine the effect of hyperparameter α on srmbatch's convergence performance. We test the statistical efficiency (loss versus epoch) of srmbatch on all four datasets with the corresponding batch size chosen as shown in Section IV-C. In each case, α traverses the set $\{0, 0.01, 0.1, 1, 10, 100\}$. Fig. 5 illustrates the experimental results, we have three observations.

First, the convergence rate of srmbatch increases while α decreases, since smaller α truly reduces more centroid staleness. Second, however, when α is equal to 0, the final loss of srmbatch tends to increase obviously and sometimes vibrates as well. It means that this configuration lowers the convergence quality. The underlying reason is that even though it eliminates most staleness, it suffers from a severe sampling distortion problem. This problem is introduced by updating new centroids from a mbatch, rather than a full batch, of samples, as mentioned in Section III-D. Third, srmbatch's convergence rate with $\alpha > 1$ is very close to the rate with $\alpha = 1$ and is apparently lower than the rate with $0 \leq \alpha < 1$.

In Section III-C, we analyze that when $y \geq 2$ and $0 \leq \alpha \leq 1$, srmbatch's centroid staleness ($L_{x,y}$) is always lower than or equal to mbatch's. From the results here, we can also find that for srmbatch itself, $0 \leq \alpha < 1$ works better than $\alpha \geq 1$ in the sense of convergence rate because lower alpha causes lower centroid staleness. So to achieve both the high convergence quality and the high convergence rate, $0 < \alpha < 1$ is the recommended hyperparameter range. What's more, from the experiment results, we can observe that $\alpha = 0.01$ usually leads to both a high convergence rate and high convergence quality. Considering the cost of hyperparameter tuning and the algorithm performance, we just recommend setting $\alpha = 0.01$ to achieve good clustering solutions.

F. Extension: Staleness-Reduction km++

We validate that srmbatch can be used together with other orthogonal optimization methods like km++ [13], where km++ optimizes the centroid initialization process, based on the idea that the distance between the initial centroids should be as far as possible and it still shows a great improvement in performance compared with km and mbatch.

Fig. 6 shows the experimental results on four datasets with $K = 32$ and we can have a similar conclusion on other values of K. We observe that srmbatch with km++ still converges the fastest among all three algorithms and its final loss is similar to the loss of km.

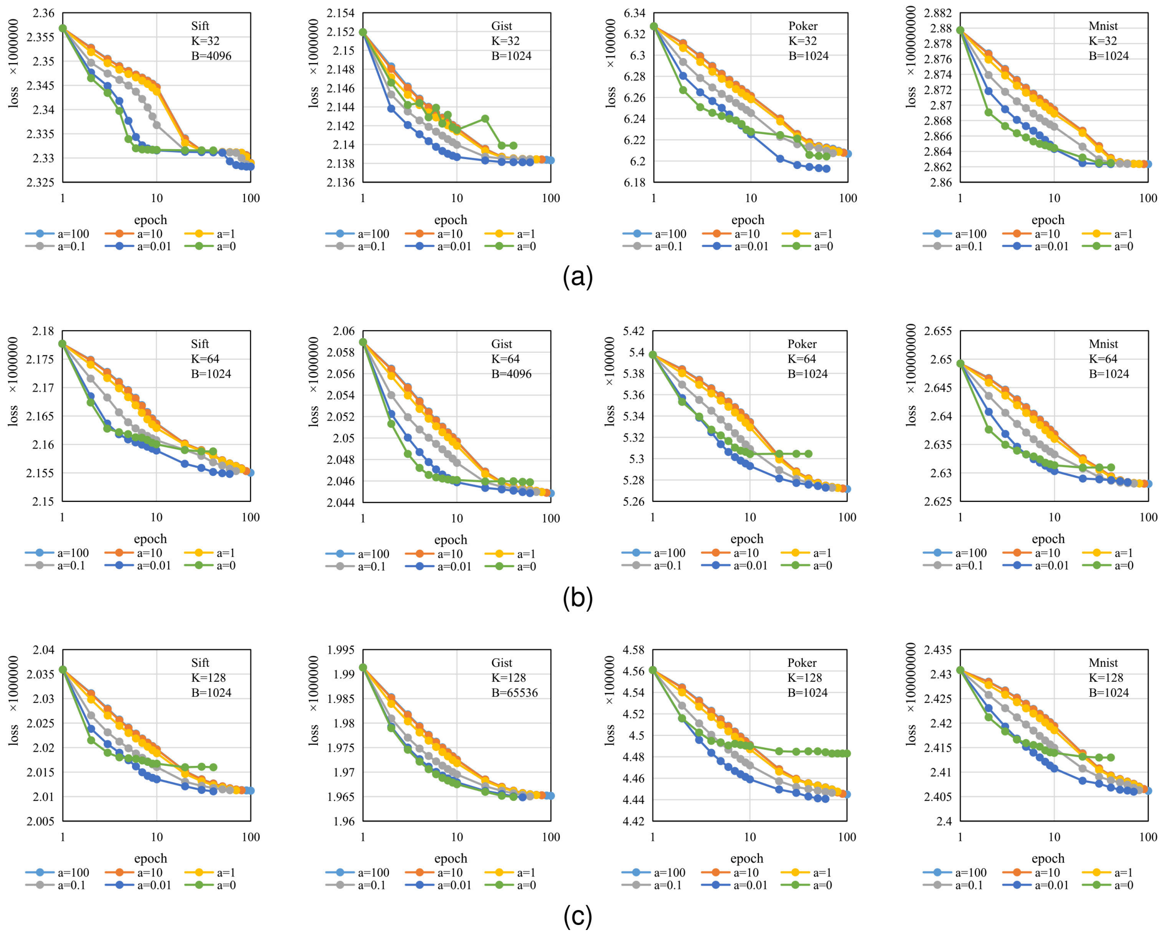


Fig. 5. Effect of Hyperparameter α. (a) $K = 32$. (b) $K = 64$. (c) $K = 128$.

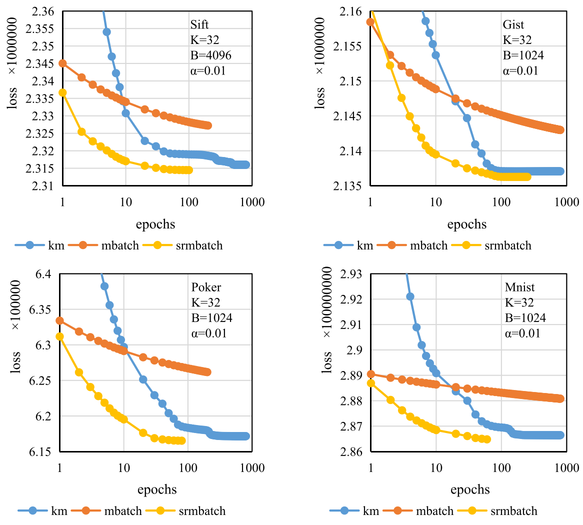


Fig. 6. Performance with km++.

V. Related Work

There are still lots of other studies improving the convergence performance or computation efficiency of km.

A. Distance Calculation Pruning Techniques

Many existing acceleration methods focus on reducing distance computations, including KD-tree-based and triangle-inequality-based approaches. KD-tree-based optimizations [14], [15], [16] enable a fast closest cluster search by storing points in special data structures. Triangle-inequality-based optimizations [17], [18], [19], [20], [21], [22], [23] utilizes cheaper bound computations to avoid calculating expensive certain distances. These techniques are orthogonal to our proposed algorithm.

B. Better Initial Centroids

The km algorithm is highly sensitive to the initial centroids [24], thus a lot of research has been done on the topic of initialization strategies. For example, km++ [13] optimizes the way of centroid initialization based on the idea that the distance between the initial centroids should be as far as possible. Moreover, different variants of km++ have been proposed [25], [26]. However, these methods do not change the process of computing distances and updating centroids, thus can also be easily integrated into our algorithm.

C. Approximated Methods

Approximated methods improve computation efficiency at the cost of a slight loss of accuracy, and thus are in high demand considering the large-scale problems. Besides mbatch [4], there are other approximated methods, including approximated search [27], hierarchical search [28], and subset-based methods [29], [30]. Recent advances in approximated methods include nested mbatch km [10], which accelerates mbatch via the distance pruning technique of [17]. However, this algorithm is quite more complex and is harder to implement, compared with mbatch and our srmbatch. Allowing simple implementation is an outstanding advantage on modern hardware devices like GPUs and field-programmable gate arrays (FPGAs) that expose massive parallelism.

D. Product Quantization (PQ)

PQ encodes vectors into several disjointed sub-vectors [7], which is a common technique for handling large-scale data. For example, PQ km [11] uses PQ codes to realize fast and memory-efficient km clustering. It achieves better accuracy than B km [12], which converts vectors into binary codes [31].

VI. Conclusion

In this article, we propose the k-means srmbatch to achieve both a low computation cost and a high clustering quality. To our knowledge, we're the first to analyze the centroid staleness of the mbatch k-means (mbatch) algorithm and propose srmbatch to improve it. Experiments verify the efficiency of srmbatch on increasing convergence rate and converging to a better clustering result while still keeping hardware efficiency on modern CPUs and GPUs, compared with mbatch. We take a more complex model (e.g., a binomial model or an exponential model) about the effect of stale assignment results on loss as our main future work. Besides, we also believe it is also interesting to modify the form of srmbatch to support streaming data. What's more, we consider that accelerating srmbatch on FPGAs [32] is worth a try as well because of its simple and hardware-friendly workflow.

Acknowledgment

The authors would like to thank anonymous reviewers for their detailed feedback.

References

[1] N. Dhanachandra, K. Manglem, and Y. J. Chanu, "Image segmentation using K-means clustering algorithm and subtractive clustering algorithm," *Proc. Comput. Sci.*, vol. 54, pp. 764–771, 2015.

[2] X. Jiang, C. Li, and J. Sun, "A modified K-means clustering for mining of multimedia databases based on dimensionality reduction and similarity measures," *Cluster Comput.*, vol. 21, no. 1, pp. 797–804, Mar. 2018.

[3] V. Jain, Y. Sharma, A. Bhatia, and V. Arora, "Crime prediction using K-means algorithm," *Global Res. Develop. J. Eng.*, vol. 2, no. 5, pp. 206–209, 2017.

[4] D. Sculley, "Web-scale K-means clustering," in *Proc. 19th Int. Conf. World Wide Web*, Apr. 2010, pp. 1–2.

[5] S. Z. Selim and M. A. Ismail, "K-means-type algorithms: A generalized convergence theorem and characterization of local optimality," *IEEE Trans. Pattern Anal. Mach. Intell.*, vols. PAMI–6, no. 1, pp. 81–87, Jan. 1984.

[6] H. Wilson. (Jul. 10, 2015). *What is SIMD?* [Online]. Available: https://huonw.github.io/blog/2015/07/what-is-simd/

[7] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–128, Jan. 2011.

[8] F. Cattral and R. Oppacher, "Poker hand," UCI Mach. Learn. Repository, MIT Press, Cambridge, MA, USA, 2007.

[9] G. Loosli, S. Canu, and L. Bottou, "Training invariant support vector machines using selective sampling," in *Large Scale Kernel Machines*, L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, Eds. Cambridge, MA, USA: MIT Press, 2007, pp. 301–320. [Online]. Available: http://leon.bottou.org/papers/loosli-canu-bottou-2006

[10] J. Newling and F. Fleuret, "Nested mini-batch K-means," in *Proc. NIPS*, 2016, pp. 1–9.

[11] Y. Matsui, K. Ogaki, T. Yamasaki, and K. Aizawa, "PQK-means: Billion-scale clustering for product-quantized codes," in *Proc. MM*, 2017, pp. 1725–1733.

[12] Y. Gong, M. Pawlowski, F. Yang, L. Brandy, L. Bourdev, and R. Fergus, "Web scale photo hash clustering on a single machine," in *Proc. CVPR*, 2015, pp. 19–27.

[13] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proc. 8th Annu. ACM-SIAM Symp. Discrete Algorithms*, Stanford, CA, USA, 2007, pp. 1027–1035.

[14] G. Di Fatta and D. Pettinger, "Dynamic load balancing in parallel KD-tree K-means," in *Proc. 10th IEEE Int. Conf. Comput. Inf. Technol.*, Jul. 2010, pp. 2478–2485.

[15] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient K-means clustering algorithm: Analysis and implementation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 7, pp. 881–892, Jul. 2002.

[16] P. Sirait and A. M. Arymurthy, "Cluster centres determination based on KD tree in K-means clustering for area change detection," in *Proc. Int. Conf. Distrib. Frameworks Multimedia Appl.*, Aug. 2010, pp. 1–7.

[17] C. Elkan, "Using the triangle inequality to accelerate k-means," in *Proc. ICML*, 2003, pp. 147–153.

[18] G. Hamerly, "Making K-means even faster," in *Proc. SIAM Int. Conf. Data Mining*, Apr. 2010, pp. 130–140.

[19] Y. Ding, Y. Zhao, X. Shen, M. Musuvathi, and T. Mytkowicz, "Yinyang K-means: A drop-in replacement of the classic K-means with consistent speedup," in *Proc. ICML*, 2015, pp. 579–587.

[20] J. Drake and G. Hamerly, "Accelerated K-means with adaptive distance bounds," in *Proc. NIPS*, vol. 8, 2012, pp. 1–4.

[21] T. Bottesch, T. Bühler, and M. Kächele, "Speeding up K-means by approximating Euclidean distances via block vectors," in *Proc. ICML*, 2016, pp. 2578–2586.

[22] R. R. Curtin, "A dual-tree algorithm for fast K-means clustering with large K," in *Proc. SIAM Int. Conf. Data Mining*, 2017, pp. 300–308.

[23] G. Hamerly and J. Drake, "Accelerating lloyd's algorithm for K-means clustering," in *Partitional Clustering Algorithms*. Cham, Switzerland: Springer, 2015, pp. 41–78.

[24] J. M. Peña, J. A. Lozano, and P. Larrañaga, "An empirical comparison of four initialization methods for the K-means algorithm," *Pattern Recognit. Lett.*, vol. 20, no. 10, pp. 1027–1040, Oct. 1999.

[25] O. Bachem, M. Lucic, H. Hassani, and A. Krause, "Fast and provably good seedings for K-means," in *Proc. NIPS*, vol. 29, 2016, pp. 55–63.

[26] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, "Scalable K-means++," *Proc. VLDB Endowment*, vol. 5, no. 7, pp. 622–633, 2012.

[27] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object retrieval with large vocabularies and fast spatial matching," in *Proc. CVPR*, 2007, pp. 1–8.

[28] D. Nistér and H. Stewenius, "Scalable recognition with a vocabulary tree," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, vol. 2, Jun. 2006, pp. 2161–2168.

[29] A. Broder, L. Garcia-Pueyo, V. Josifovski, S. Vassilvitskii, and S. Venkatesan, "Scalable K-means by ranked retrieval," in *Proc. 7th ACM Int. Conf. Web Search Data Mining*, Feb. 2014, pp. 233–242.

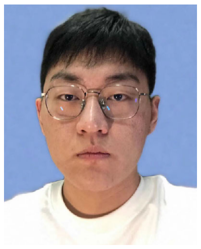
[30] Y. Avrithis, Y. Kalantidis, E. Anagnostopoulos, and I. Z. Emiris, "Web-scale image clustering revisited," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1502–1510.

[31] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin, "Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 12, pp. 2916–2929, Dec. 2013.

[32] Z. He, Z. Wang, and G. Alonso, "BiS-KM: Enabling any-precision K-means on FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Feb. 2020, pp. 233–243.

Xueying Zhu received the bachelor's degree from Zhejiang University, Hangzhou, China, in 2021, where she is currently pursuing the Ph.D. degree.

Her current research interests include in-network computation and SmartNIC.



Jie Sun received the bachelor's degree from Zhejiang University, Hangzhou, China, in 2021, where he is currently pursuing the Ph.D. degree.

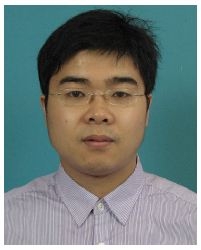
His current research interests include graph neural network and machine learning systems.

Zhenhao He received the bachelor's degree from Tongji University, Shanghai, China, in 2016, and the master's degree from ETH Zurich, Zurich, Switzerland, in 2018, where he is currently pursuing the Ph.D. degree.

His research interests include heterogeneous computing and distributed computing.

Jiantong Jiang received the master's degree from Northeastern University, Shenyang, China, in 2020. She is currently pursuing the Ph.D. degree with The University of Western Australia, Crawley, Australia.

She was a Research Assistant at the School of Software Engineering, Zhejiang University, Hangzhou, China. Her current research interests include high-performance computing and machine learning systems.



Zeke Wang received the Ph.D. degree from Zhejiang University, Hangzhou, China, in 2011.

He is a Research Professor at the Collaborative Innovation Center of Artificial Intelligence, Department of Computer Science, Zhejiang University, China. His current research interests mainly focus on building machine learning systems using heterogeneous devices, e.g., SmartNIC and SmartSwitch.