2018

# Design and Implementation of an Efficient Parallel Feel-the-Way Clustering Algorithm on High Performance Computing Systems

Weijian Zheng
*Indiana University - Purdue University, Indianapolis*, zheng273@purdue.edu

Fengguang Song
*Indiana University Purdue University Indianapolis*, song412@purdue.edu

Dali Wang
*Oak Ridge National Laboratory*, wangd@ornl.gov

Report Number:
18-002

# Design and Implementation of an Efficient Parallel Feel-the-Way Clustering Algorithm on High Performance Computing Systems

Weijian Zheng
Department of Computer Science
Purdue University
Indianapolis, Indiana
zheng273@purdue.edu

Fengguang Song
Department of Computer Science
Indiana University-Purdue University
Indianapolis, Indiana
fgsong@cs.iupui.edu

Dali Wang
Oak Ridge National Laboratory
Oak Ridge, Tennessee
wangd@ornl.gov

## Abstract

*This paper proposes a Feel-the-Way clustering method, which reduces the synchronization and communication overhead, meanwhile providing as good as or better convergence rate than the synchronous clustering methods. The Feel-the-Way clustering algorithm explores the problem space by using the philosophy of "crossing an unknown river by feeling the pebbles in the riverbed." A full-step Feel-the-Way clustering algorithm is first designed to reduce the number of iterations. In the full-step Feel-the-Way algorithm, each process runs a number of L steps before synchronizing its local solutions with other processes. This full-step algorithm can significantly decrease the number of iterations compared to the k-means clustering method. Next, we extend the full-step algorithm to a sampling-based Feel-the-Way algorithm to achieve higher performance. Furthermore, we prove that the proposed new algorithms (both full-step and sampling-based Feel-the-Way) can always converge. Our empirical results demonstrate that the optimized sampling-based Feel-the-Way method is much faster than the widely used k-means clustering method as well as providing comparable costs. A number of experiments with synthetic datasets, real-world datasets of MNIST, CIFAR-10, ENRON, and PLACES-2 show that the new parallel algorithm can outperform the k-means by up to 235% on a high performance computing system with 2,048 CPU cores.*

## 1. Introduction

Machine learning is a primary mechanism to extract information and insights from big data. It has recently generated large amounts of momentous results in both academia and industry. Owing to the large data volume, exponential rate of data generation and extreme-scale data processing, high performance computing (HPC) systems have become the standard platform to solve machine learning problems at extreme scales. However, achieving high scalability for parallel and distributed machine learning algorithms is still a challenging task.

This paper targets the k-means clustering method, which is one of the most widely used big data analytics methods today [1], [2]. It partitions a set of data points into $k$ clusters by the following two steps: 1) assign each data point to its closest center, and 2) recompute new centers as the means of the just assigned points. The process is repeated for a number of iterations until it converges. To achieve high performance on extreme-scale systems with millions of CPUs, it is crucial to reduce the communication cost and synchronization cost of a parallel algorithm. Researchers are conducting extensive studies to design communication-reducing and synchronization-reducing machine learning algorithms [3], [4], [5], [6]. These algorithms may reduce the communication cost and synchronization cost to speed up the execution time for each iteration. However, they sometimes cause a slower convergence rate (i.e., requiring more iterations), lower quality of solutions, or even divergence [7], [8], [9], [10]. Consequentially, the overall execution time becomes longer since the increased number of iterations can overwhelm the benefits of improved performance per iteration.

Intrigued by this problem, we start to explore whether we can design a set of new machine learning algorithms that can not only *reduce the synchronization cost* but also achieve a *better convergence rate*. To that end, we design a new algorithm named "Feel-the-Way" algorithm. The goal is to achieve the best of the two worlds: The new algorithm attains minimized synchronizations meanwhile achieving the same or faster convergence rate.

We design a Feel-the-Way *Clustering* algorithm to achieve the goal. The basic idea is that each computing process behaves like a person who is trying to "cross an unknown river by touching or feeling the pebbles/rocks in the riverbed." In the design, the current rock is the safe "harbor" where the person can always come back and try a new direction again. Note that many people are crossing the same river and will synchronize from time to time. Following the idea, we design the Feel-the-Way algorithm.

In the Feel-the-Way algorithm, each process has a subset of the input dataset, and runs $L$ local clustering steps before synchronizing its local solution with other processes' local solutions. Each process keeps track of its local solution (i.e., $k$ centers) for every local step from 1 to $L$. At the end of the $L$-th step, all processes' local solutions are merged together to determine a new global solution. Next, every process uses the new global solution as a starting point, and repeats the same steps until it converges. The rationale behind the idea is that

*This material is based upon research supported by the Purdue Research Foundation, by the NSF Grant# 1522554, and by the U.S. Department of Energy (DOE), Office of Science, Advanced Scientific Computing Research.*

the solution from the first step is always a "decent" candidate for the global solution. If one of the second, third, ..., and $L$-th steps has a better optimization cost than the first step, it implies that the Feel-the-Way algorithm has successfully reduced synchronizations and communications, meanwhile achieving a better cost than the original synchronous algorithm. In general, the best of the $L$ candidates will be at least as good as the original synchronous algorithm in terms of the optimization cost and the convergence rate. To understand it, considering an extreme case, if all the local steps between the second and the $L$-th make the cost worse than the first step, the final overall cost can still be close to the synchronous first-step-only clustering method because the first step is the "winner" which is then adopted by the Feel-the-Way algorithm.

Our first attempt of realizing the new algorithm shows good results. Experiments in Section 6.2 demonstrate that using $L$=5 can reduce the average number of iterations from 61 to 10 with the MNIST dataset. However, it is not trivial to implement the algorithm in practice efficiently. To make it run as fast as possible, we need to solve the following challenge: the time spent on the second to the $L$-th local steps must be significantly small to reduce the total execution time. Otherwise, the saved iterations may be overwhelmed by the time of the second to $L$-th local steps. Hence, we consider the second to $L$-th local steps as an overhead we pay in order to decrease the number of iterations.

To reduce the overhead incurred by the second to $L$-th steps, we use a variety of sampling methods to avoid re-clustering all the data points. Based on our analysis, not every point is reassigned in the next iteration. Five sampling methods have been designed and compared with each other: 1) Random sampling, 2) a new Max-min sampling, 3) a new coefficient of variation (CV) sampling, 4) *Heap* sampling [11], and 5) a new Reassignment-history-aware sampling method. These sampling algorithms have different time complexities and distinct efficiencies on the convergence rate. To integrate the Feel-the-Way clustering algorithm with the sampling algorithm, we also design a new approach to computing the sampling-based optimization cost efficiently for every local $i$-th step ($i \in \{2, \ldots, L\}$). All the sampling methods are controlled by a sampling rate $r$, and each local step may have a different sampling rate. Finally, we call the new algorithm "sampling-based Feel-the-Way algorithm". By contrast, the original algorithm (i.e., without sampling) is called "full-step Feel-the-Way algorithm".

We first design and develop sequential algorithms for the full-step Feel-the-Way algorithm and the sampling-based Feel-the-Way algorithm, respectively. The sequential algorithms partition the input dataset into a number of data blocks, each with an equal number of data points. Then the algorithms apply $L$ local steps to each data block. Every local step executes the k-means clustering method for one iteration upon the data block. After all data blocks have been computed, the data blocks' centers will be merged to get new global centers. Given the new global centers, we repeat the same process. Note that the sampling-based algorithm only computes a subset of points during the second to $L$-th local steps, but computes all points in the first local step.

After verifying the correctness of the sequential algorithms, we develop a *parallel* sampling-based Feel-the-Way algorithm using the hybrid MPI/Pthread programming model. In the parallel implementation, every thread works on its own subset of data blocks, and merges centers with other threads. We conduct a number of experiments with synthetic data sets and real-world datasets of MNIST, CIFAR-10, ENRON, and PLACES-2 on HPC systems. The experimental results show that the introduced *reassignment-history-aware sampling method* is more effective than the other sampling methods, and the new parallel Feel-the-Way algorithm can outperform the k-means algorithm significantly. To the best of our knowledge, this paper makes the following contributions:

1) We propose a full-step Feel-the-Way algorithm and a sampling-based Feel-the-Way algorithm to reduce synchronizations at the same time achieving as good as or better convergence rates.
2) We design a variety of sampling methods, among which we introduce the most effective reassignment-history-aware sampling method.
3) We prove that the sampling-based Feel-the-Way algorithm converges. We empirically show that the convergence rate is faster with the Feel-the-Way algorithm.
4) Our experiments with both sequential and parallel implementations using real-world datasets demonstrate that the Feel-the-Way method can provide faster performance than the k-means method.

## 2. Related work

Existing machine learning libraries such as Mahout [12], Spark MLlib [13], Google TensorFlow [14], GraphLab [15] and Amazon AML [16] are general-purpose Big Data ecosystems. They target high productivity by using Python or Java on Cloud platforms (e.g., Hadoop). For instance, Alex et al. found that Spark has significant runtime overhead (e.g. inter-stage barrier, task start delay, etc.) compared to MPI-based software [17]. Differently, this paper targets high performance [17], [18], and focuses on designing a new parallel clustering algorithm on HPC systems using the hybrid MPI/Pthread programming model.

Numerous parallel ML algorithms on distributed memory systems have been designed and developed: Some are strictly synchronous but lead to significant communication cost, while the others use relaxed or lazy synchronizations but cannot guarantee the same convergence rate as the the original synchronous algorithms [7], [8], [9], [19], [20], [21]. Xing et al. study machine learning applications that use the stochastic gradient descent (SGD) algorithm, and adaptively change the SGD *step size* to compensate for errors caused by stale synchronizations. Although we share the same philosophy of reducing synchronizations, our work focuses on clustering methods (i.e., not SGD related), and designs a distinct Feel-

the-Way algorithm to minimize both synchronization cost and the number of iterations.

Mini-batch k-means clustering is a sampling-based clustering algorithm [22]. It randomly selects a small number of data points from each batch. In this work, we test five different sampling methods, and incorporate them (as a submodule) into the new Feel-the-Way algorithm. Note that our Feel-the-Way algorithm is a hybrid of sampling algorithm and ordinary clustering algorithm, and can achieve the same optimization cost as the original k-means. Kurban et al. design the *heap* method and use it to the k-means clustering method [11]. We compare the *heap* method with our *reassignment-history based* method, and show that the reassignment-history-aware method can result in a better convergence rate than the heap method.

## 3. Full-Step Feel-the-Way Algorithm

This section introduces the proposed full-step Feel-the-Way algorithm, which works on blocks of data points and applies local optimizations to each individual block iteratively. Each block of data points will be optimized by a number of $L$ local steps without triggering any communications. The algorithm consists of three parts: 1) The local optimization function that iteratively optimizes each data block; 2) the merge function that derives the global clustering centers from different blocks; and 3) the main function that simply controls when to stop the algorithm. The rest of the section will introduce the three functions in details.

### 3.1. Data structure

We use a *blocked* data layout to store the input dataset. In a machine learning application, each data point can be represented by a vector with $m$ attributes. Assuming we have $n$ data points, a dataset can be viewed as an n-by-m matrix. In our algorithm, we divide the matrix into blocks of rows, where each block stores a set of $b$ consecutive data points. This blocked data structure is used by both full-step and sampling-based Feel-the-Way algorithms.

### 3.2. Local optimization function

The local optimization function is in charge of reducing each data block's optimization cost. Its idea is as follows: we run one step of the synchronized algorithm (e.g., the original k-means), then we run $L$-1 steps of asynchronous steps upon each block. The additional $L$-1 steps on each data block are supposed to reduce each block's SSE (Sum of Squared Error) cost monotonically (Theorem 5.2 in Section 5 will prove it).

Algorithm 1 shows the pseudocode of the local optimization function. The function goes through three stages. *Stage 1*: Set the current data block's clustering seed as the newly merged global centers. *Stage 2*: Use the current data block's points to improve the block's clustering cost. In stage 2.1, we find the closest center for each data point; In stage 2.2, we compute each block's local centers' sizes and sums of coordinates; In stage 2.3, we get new local centers based on step 2.2. *Stage 3*: The local optimization function returns if a number of $L$ local steps have been applied to the current data block.

**Algorithm 1** Function of Local Optimization.

```
1: /* Local optimization on one block */
2: local_optimization_block(points, g_centers,
3:       g_centers_size, membership, L)
4: l = 0 /* l is the current step */
5: while l ≤ L − 1 do
6:     blk_cost_old = blk_cost_new; blk_cost_new = 0;
7:     blk_local_sum = 0; blk_local_size = 0;
8:     ▷ stage 1: Use the global centers as a new seed for 0 step
9:     if (l = 0) then
10:        centers ← g_centers;
11:    end if
12:    ▷ stage 2: Use all local points to improve centers
13:    for each point i in points do
14:        ▷ stage 2.1 find the closest center for point i
15:        (dist, new_center) ← find_nearest_center
16:        (points[i], centers);
17:        membership[i] = new_center;
18:        ▷ stage 2.2 update each block's partial sum and size
19:        blk_local_sum[new_center] += points[i];
20:        blk_local_size[new_center]+=1;
21:        blk_cost_new += dist;
22:    end for
23:    ▷ stage 2.3: Recalculate new centers and new cost
24:    for each center c do
25:        centers[c] = blk_local_sum[c] / blk_local_size[c];
26:    end for
27:    if (l = 0) then
28:        1st_step_cost ← blk_cost_new;
29:    end if
30:    l++;
31: end while
32: ▷ stage 3: Returns if L steps are finished
33: return {blk_local_sum, blk_local_size, 1st_step_cost}
```

### 3.3. The merge function

After calling the local optimization function, each data block has its own set of centers. Therefore, $N$ data blocks will have $N$ set of centers. Given each data block's *local_sum*, *local_size* and a new cost *local_cost*, the merge function will add them together to get the *global_sum* and *global_size*, respectively. The new global centers are then computed by dividing *global_sum* by *global_size*.

### 3.4. The main function

Algorithm 2 shows the main function of the full-step Feel-the-Way algorithm. It calls the previous local optimization and merge functions. As shown in the algorithm, it first selects $k$ points as the initial seed. Next, it iteratively optimizes each data block, and aggregates each clustering center's size and coordinate sum (see lines 12-20). After all the data blocks have been processed once, the new global centers can be computed (see lines 22-24).

**Algorithm 2** Full-Step Feel-the-Way Algorithm

```
1: /* m : number of points , n_b: number of blocks */
2: /* k centers , L: the limit for number of local steps */
3: Feel_the_Way_clustering(points, m, n_b, k, L)
4: /* Set the initial k centers */
5: pre_g_cost_new = MAXIMUM
6: repeat
7:     /* Store the previous iteration's info */
8:     g_size_old = g_size_ne; g_size_new = 0;
9:     g_sum_new = 0;
10:    pre_g_cost_old = pre_g_cost_new;
11:    pre_g_cost_new = 0;
12:    for each block i ← 0 to n_b - 1 do
13:        /* Run local optimization on each block */
14:        (local_sum, local_size, pre_local_cost) ←
15:        local_optimization_block(i-th block, g_centers,
16:        g_size_old, membership, L);
17:        /* Merge each block's coordinate sum into global sum */
18:        (g_sum_new, g_size_new, pre_g_cost_new) ←
19:        merge(local_sum, local_size, pre_local_cost);
20:    end for
21:    /* Compute new global centers */
22:    for each center c do
23:        g_centers[c] = g_sum_new[c] / g_size_new[c];
24:    end for
25: until pre_g_cost_new ≥ pre_g_cost_old
```

# 4. Sampling-Based Feel-the-Way Algorithm

In this section, we first analyze the performance of the full-step Feel-the-Way algorithm. Motivated by the unscalable performance of the full-step algorithm, we then introduce a sampling-based Feel-the-Way algorithm. We design and compare five different types of sampling methods. Moreover, we introduce a new method to compute the new SSE cost given only a small portion of reassigned data points.

## 4.1. Motivation for a new sampling-based algorithm

We evaluate the performance of the full-step Feel-the-Way algorithm on a synthetic dataset SYN (described in Table 4). Figure 1.a shows that the full-step Feel-the-Way algorithm can significantly reduce the number of iterations. For instance, k-means has 74 iterations while the full-step Feel-the-Way (when $L$=5) has 17 iterations. However, in Figure 1.b, we find that the full-step Feel-the-Way is not faster than k-means when $L$=4 and 5, even though Feel-the-Way has a less number of iterations than k-means. The reason is that the time spent on the second to *L-th* local steps is expensive, which is eventually larger than the benefit of saved iterations. Later in Lemma 5.5, we will prove that the full-step Feel-the-Way algorithm becomes slower than k-means if it takes more than $\frac{m}{L}$ iterations to converge, where $m$ is the number of iterations that the k-means algorithm takes.

Therefore, we hope to design a new algorithm, which can spend little time on the local steps (from the second to the $L$ steps), at the same time reducing the number of iterations. Our in-depth analysis of tracking data points transitions between different clusters finds that, not all data points have been reassigned to a distinct cluster. Also, fewer and fewer data points are reassigned from the first iteration to the last iteration. Intuitively, if we could just pick up those reassigned points (and skip those unchanged points) to compute, the Feel-the-Way algorithm can certainly run much faster. Hence, we introduce different sampling schemes to the original Feel-the-Way algorithm's local steps to intelligently select a small portion of data points to compute clustering, instead of considering all data points (as done by the full-step algorithm).

## 4.2. Design of the sampling-based Feel-the-Way

The main function of the sampling-based Feel-the-Way algorithm is the same as the full-step Algorithm 2 except for the local optimization function.

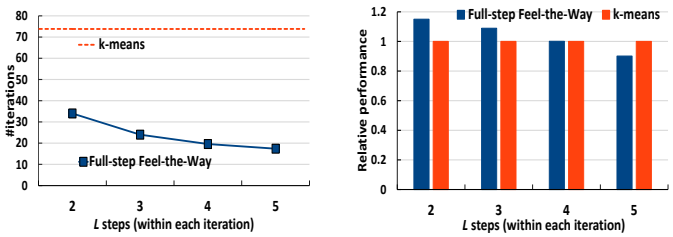
The new local optimization function is displayed in Algorithm 3. In lines 7-9, the sampling algorithm checks whether it has visited enough data points or not. In line 10, only the sampled points will be selected to compute clustering. From line 11 to line 18, each sampled point is reassigned to its closest center, and those consequentially affected cluster's coordinate sum and number of points are updated accordingly. Please note that this algorithm must consider all the data points as stated in line 10 when the algorithm executes the first local step.

The essence of the Feel-the-Way algorithm is to use the first local step to emulate the original synchronous k-means algorithm, and use the following $L-1$ local steps to improve the cost of the first local step. Moreover, the sampling algorithm utilizes the first local step to compute the global SSE cost for all blocks of data points, and determines which points should be sampled based upon the reassignment statistics of the first local step.

## 4.3. Development of various sampling methods

We design and develop five different sampling methods. These sampling methods depend on one of two types of information: 1) distance of the data point to each clustering center (near or far), and 2) the history of previous steps that reassign certain points to different centers.

The five sampling methods are as follows:




(a) SYN (iterations) (b) SYN (speedup)

Fig. 1: Comparison between the full-step Feel-the-Way algorithm and the k-means algorithm.

**Algorithm 3** Sampling Version of the Local Optimization Function

```
1: /* Local optimization on one block */
2: local_optimization_block(points, g_centers,
3:       g_centers_size, membership, L, max_sample_points)
4: /*same as lines 4-11 of Algorithm 1) */
5: ▷ stage 2: Use sampled local points to improve centers
6: for each point i in points do
7:     if (sampled_points > max_sample_points) then
8:         break;
9:     end if
10:    if (point i is sampled or local_step = 0) then
11:        sampled_points = sampled_points + 1;
12:        ▷ stage 2.1 find the closest center for point i
13:        (dist, new_center) ← find_nearest_center
14:        (points[i], centers);
15:        membership[i] = new_center;
16:        ▷ stage 2.2 update each block's partial sum and size
17:        blk_local_sum[new_center] += points[i];
18:        blk_local_size[new_center]+=1;
19:        if (l != 0 ) then
20:            blk_local_sum[old_center] -= points[i];
21:            blk_local_size[old_center]-=1;
22:        end if
23:    end if
24: end for
25: /*same as local optimization function (i.e., Alg. 1) lines 23-33*/
```

1) *Random sampling*: It randomly selects a subset of data points.
2) *Max-min sampling*: For each data point, we compute its distances to the k clustering centers, respectively. Then we compute its *Max-min* (i.e., $\frac{max\ distance}{min\ distance}$). Those points that have smaller Max-min values will be selected as the sampling points. It is based on our assumption that if a data point is equally close to all k centers, it is more likely this point will change its closest center.
3) *Coefficient of Variation (CV)*: CV is defined as $\frac{\text{standard deviation}}{mean}$. For each data point, there are k distances to k centers. This method computes the CV of those k distances. A small CV reveals that the distances of the data point to all centers are almost the same. Hence, the data points that have smaller CVs will be selected by assuming that a point with comparable distances to all centers is likely to be reassigned soon.
4) *Heap sampling*: Heap based sampling is inspired by the work of Kurban et al. [11]. The heap based method maintains a heap data structure for each center to store the distances between the center and its data points. The farthest data point is stored at the top of the heap (similar to heap sorting). When doing sampling, we always select the points that are at the upper levels of the heap. The heap method assumes that the points that are far from their belonging center will switch to a new center more quickly.
5) *Reassignment-history-aware sampling*: It is a new sampling method proposed by this work. Reassignment-history-aware sampling tries to keep track of which data points have been reassigned in the past. If a data point is assigned to a new center in the previous step, we consider it as a sampling point candidate by assuming it will change center again in the current step. Suppose there are $m$ data points that have been assigned to new centers in the previous step, we will randomly pick $s$ sample points from the set of $m$ points. $s$ is either a constant or decided by a sampling ratio. When $m$ becomes too small (e.g., less than 20), we will use all $m$ points as the sampling points.

*Time complexity of the sampling methods:* In the above Max-min, CV, and Heap sampling methods, we use the Quick-Select algorithm [23] to find the first $s$ numbers (either largest and smallest) in an array. QuickSelect has a time complexity of $O(logn)$. Given $k$ centers and each data block with $b$ data points, we summarize the time complexities of all the sampling methods in Table 1 (proofs are skipped here). Here, we assume that each sampling method needs to select $s$ sample points from each data block.

TABLE 1: Time complexity of different sampling methods.

| Random | Max-min | Heap | CV | Reassignment-history-aware |
|---|---|---|---|---|
| $O(s)$ | $O(s \log b)$ | $O(sk \log \frac{b}{k})$ | $O(s \log b)$ | $O(s)$ |

## 4.4. An efficient way to compute the new SSE cost in the sampling algorithm

For each local step $l$ ($l \in \{1 \dots L-1\}$) for which sampling is needed, we will compute the new SSE cost at the end of the $l$-th local step. Certainly we cannot recompute every point's distance to its center because of its expensive cost, which would make the sampling algorithm have the same time complexity of the full-step algorithm.

In order to compute a new SSE cost by considering only the small subset of sampling points, we divide the SSE cost computation into two parts: 1) SSE cost related to the sampled points, and 2) SSE cost related to the upsampled points.

Suppose $V$ represents all the points in a data block, and $S$ and $U$ represent sampled points and upsampled points, respectively, where $V = S \cup U$.

$cost(S)$ (i.e., the cost of the sampled data points) is the SSE cost of the sampled point. It is computed when each sampled points is reassigned to its closest center.

Next, the SSE cost of unsampled points $cost(U)$ is computed as follows. We utilize the old SSE cost of unsampled data points $cost(U)_{old}$ to compute the new cost of unsampled data points. Specifically, given a set of centers $C$ from the previous local step, after re-clustering, we get a new set of centers $C'$. For unsampled data point $U$, we also record the coordinates sums and the number of data points attached to each center in previous step as $sum$ and $size$. Let $\sigma_i = C'_i - C_i$, where $C_i$ is the $i$-th center. The corresponding computation formula is as follows: $cost(U) = cost(U)_{old} - 2\sum_{i=1}^{k}(sum_i - size_i \cdot C_i) \cdot \sigma_i + \sum_{i=1}^{k}(size_i(\sigma_i \cdot \sigma_i))$.

Eventually, $cost(V) = cost(S) + cost(U)$.

## 5. Theoretical Analysis

In this section, we will analyze the proposed Feel-the-Way algorithm, and provide theorems and lemmas to answer two questions: 1) Does the sampling-based Feel-the-Way algorithm always converge (subsection 5.1)? 2) In what conditions the Feel-the-Way algorithm will be faster than the k-means method (subsection 5.2)?

### 5.1. Convergence analysis

Clustering problems study how to partition a set of data points $T$—which has a number of $n$ points in a $d$-dimension space ($\Re^d$)—into $k$ groups with the minimum cost. Let $\mu$ be a data point in $\Re^d$, and the result of clustering is a membership vector $M$ and a set of centers $C$.

Since we use a blocking data structure, $T$ is divided into a number $\frac{|T|}{b}$ of blocks. Assume $V$ is a block with $b$ data points and there are $L$ local steps within one global iteration. Given block $V$, in each local step $l \in [0, L-1]$, the local optimization function is invoked to minimize the block $V$'s own SSE cost $cost(V, C')$, where $C'$ is the current best centers of the block. We also know that $cost(V, C') = \sum_{\mu \in V} dist(\mu - C'_{M(\mu)})$. $C'_{M(\mu)}$ is the closest center among $C'$ to the point $\mu$.

Given two data points x and y, the squared Euclidean distance between them, $\Delta(x, y)$, can be computed as follows: $\Delta(x, y) = \sum_{i=1}^{d}(x_i - y_i)^2 = (x - y) \cdot (x - y)$ (i.e., a dot product).

**Lemma 5.1.** *Given an initial set of centers $C$ ($|C| = k$) for block $V$, after reassignment of each data point in $V$ to its closest center, we get a new membership $M'$. Suppose membership $M'$ implies a new set of centers $C'$, the new centers $C'$ always has a cost that is equal to or less than that of the old centers $C$. That is, $cost(V, C, M') \geq cost(V, C', M')$.*

*Proof:* Let $\sigma_i = C'_i - C_i$, where $C_i$ is the $i$-th center.

$$
\begin{aligned}
&cost(V, C, M') = \sum_{\mu \in V} \Delta(\mu, C_{M'(\mu)}) \text{ /*by definition of SSE cost*/} \\
&= \sum_{\mu \in V} (\mu - C'_{M'(\mu)} + \sigma_{M'(\mu)}) \cdot (\mu - C'_{M'(\mu)} + \sigma_{M'(\mu)}) \\
&= \sum_{\mu \in V} (\Delta(\mu, C'_{M'(\mu)}) + 2(\mu - C'_{M'(\mu)}) \cdot \sigma_{M'(\mu)} + (\sigma_{M'(\mu)})^2) \\
&= cost(V, C', M') + 2 \sum_{m=1}^{k} (\sigma_m \cdot \sum_{\mu \in V, M'(\mu) = m} (\mu - C'_m)) \\
&\quad + \sum_{\mu \in V} (\sigma_{M'(\mu)})^2 = cost(V, C', M') + \sum_{\mu \in V} (\sigma_{M'(\mu)})^2
\end{aligned}
\tag{1}
$$

Note that $\sum_{\mu \in V, M'(\mu) = m}(\mu - C'_m)$ is zero since $C'_m$ is a center of a cluster of points (i.e., the arithmetic mean position of all the points in the $m$-th cluster).

□

**Lemma 5.2.** *Let $cost(V, C^l, M^l)$ and $cost(V, C^{l+1}, M^{l+1})$ be the computed SSE cost for block $V$ before and after the $l$-th local step, then $cost(V, C^l, M^l) \geq cost(V, C^{l+1}, M^{l+1})$.*

*Proof:* In the $l$-th local step, every data point will be reassigned to its closest center such at a lower cost is achieved and a new membership of the points is formed, because each reassigned point's distance (to its new center) has decreased. Thus, $cost(V, C^l, M^l) \geq cost(V, C^l, M^{l+1})$.

The new membership $M^{l+1}$ leads to a new set of centers, denoted as $C^{l+1}$. Based on Lemma 5.1, $cost(V, C^l, M^{l+1}) \geq cost(V, C^{l+1}, M^{l+1})$. Putting two inequalities together, $cost(V, C^l, M^l) \geq cost(V, C^{l+1}, M^{l+1})$.

□

Lemma 5.2 shows that each data block's SSE cost will decrease monotonically in each local $l$-th step. Lemma 5.2 is still correct even if we only consider reassignment of a set of sampled points, for which every point in the proof refers to every sampled point and the unsampled points do not affect (i.e., increase/decrease) the SSE cost.

**Lemma 5.3.** *Suppose an input dataset $T$ is divided into $P$ data blocks. Each data block $V_i$ has its own clustering membership $M_{V_i}$ and a set of $k$ centers $C_{V_i}$. After merging all the blocks' local centers into a global set of centers $C_T$, $cost(T, C_T, M_T) \leq \sum_{i=1}^{P} cost(V_i, C_{V_i}, M_{V_i})$.*

*Proof:* Let $\sigma_i = C_{T(i)} - C_{(i)}$, where $C_{T(i)}$ is the $i$-th global center. $C_{(i)}$ here is approximately equal to the $i$-th center of each block. $\mu$ is a single data point.

$$
\begin{aligned}
&\sum_{i=1}^{P} cost(V_i, C_{V_i}, M_{V_i}) = \sum_{i=1}^{P} \sum_{\mu \in V_i} \Delta(\mu - C_{V_i(M_{V_i}(\mu))}) \\
&= \sum_{i=1}^{P} \sum_{\mu \in V_i} ((\mu - C_{T(M_{V_i}(\mu))} + \sigma_{M_{V_i}(\mu)}) \\
&\quad \cdot (\mu - C_{T(M_{V_i}(\mu))} + \sigma_{M_{V_i}(\mu)})) \\
&= \sum_{i=1}^{P} cost(V_i, C_T, M_{V_i}) + \sum_{i=1}^{P} \sum_{\mu \in V_i} (\sigma_{M_{V_i}(\mu)})^2 \\
&\quad + 2 \sum_{m=1}^{k} (\sigma_m \cdot \sum_{i=1}^{P} \sum_{\mu \in V_i, M_{V_i}(\mu) = m} (\mu - C_{T(m)})) \\
&= cost(T, C_T, M_T) + \sum_{i=1}^{P} \sum_{\mu \in V_i} (\sigma_{M_T(\mu)})^2
\end{aligned}
\tag{2}
$$

Please note that $\sum_{i=1}^{P} \sum_{\mu \in V_i, M_{V_i}(\mu) = m}(\mu - C_{T(m)})$ is zero since $C_{T(m)}$ is a center of a cluster points based on membership $M_{V_i}$.

□

**Theorem 5.4.** *Let $cost(T, C_T^{i-1}, M_T^{i-1})$ and $cost(T, C_T^i, M_T^i)$ be the computed global SSE cost for all data points $T$ before and after the $i$-th global iteration, then $cost(T, C_T^{i-1}, M_T^{i-1}) \geq cost(T, C_T^i, M_T^i)$.*

*Proof:* Assume there are $l$ local steps inside one global iteration. Let $C_V^{i(l-1)}$ and $M_V^{i(l-1)}$ be the local center and membership of $V$ after $l$-th step within $i$-th iteration. $M_T^{i(l-1)}$

is the membership of all data points copied from $M_{V_m}^{i(l-1)}$. Let $C_T^i$ be the merged global center after $i$-th iteration. $M_T^i$ is the membership of all data points implied by $C_T^i$

$$
\begin{aligned}
& cost(T, C_T^{i-1}, M_T^{i-1}) = \sum_{m=1}^{P} cost(V_m, C_{V_m}^{i(0)}, M_{V_m}^{i(0)}) \\
& \geq \sum_{m=1}^{P} cost(V_m, C_{V_m}^{i(l-2)}, M_{V_m}^{i(l-2)}) \text{ /* According to lemma 5.2 */} \\
& \geq \sum_{m=1}^{P} cost(V_m, C_{V_m}^{i(l-2)}, M_{V_m}^{i(l-1)}) \text{ /* Find the closest center */} \quad (3) \\
& \geq cost(T, C_T^i, M_T^{i(l-1)}) \qquad \text{/* According to lemma 5.3 */} \\
& \geq \sum_{m=1}^{P} cost(V_m, C_T^i, M_T^i) \qquad \text{/* Find the closest center */} \\
& = cost(T, C_T^i, M_T^i)
\end{aligned}
$$

Thus, $cost(T, C_T^{i-1}, M_T^{i-1}) \geq cost(T, C_T^i, M_T^i)$

□

Theorem 5.4 shows that the algorithm's global cost is monotonically decreasing, and the sampling-based Feel-the-Way algorithm converges.

### 5.2. Speedup analysis for the full-step and sampling-based Feel-the-Way algorithms

**Lemma 5.5** (Full-step)**.** *Assume it takes $m$ and $n$ iterations for the k-means and the full-step Feel-the-Way algorithm to converge. Let $T_{k-means}$ and $T_{full-step}$ be the computation time of the k-means and full-step Feel-the-Way algorithm, respectively. If $n \leq \frac{m}{L}$, then $T_{full-step} \leq T_{k-means}$.*

*Proof:* Assume each iteration takes time t,

$$
\begin{aligned}
& T_{k-means} \geq T_{full-step} \implies t \times m \geq t \times n \times L \\
& \implies m \geq n \times L
\end{aligned} \quad (4)
$$

Thus, when $n \leq \frac{m}{L}$, $T_{full-step} \leq T_{k-means}$.

□

**Theorem 5.6** (Sampling-based)**.** *Assume it takes $m$ and $n$ iterations for k-means and the sampling-based Feel-the-Way algorithm to converge. Suppose the sampling Feel-the-Way algorithm uses a sampling ratio of $r$. Let $T_{k-means}$ and $T_{sample}$ be the computation time of the k-means and sampling-based Feel-the-Way algorithm, respectively. If $n \leq \frac{m}{1+(L-1)r}$, then $T_{sample} \leq T_{k-means}$.*

*Proof:* Assume the time for each iteration is t

$$
\begin{aligned}
& T_{k-means} \geq T_{sample} \implies t \times m \geq (t + (L-1) \times r \times t) \times n \\
& \implies m \geq n \times (1 + (l-1) \times r)
\end{aligned} \quad (5)
$$

Thus, $T_{sample} \leq T_{k-means}$ when $n \leq \frac{m}{1+(l-1)r}$. □

Theorem 5.6 reveals that the speedup of the sampling-based Feel-the-Way over k-means is determined by the reduced number of iterations, the number of local steps, and the sampling ratio. For instance, if we choose $L = 2$ and a small $r$, the sampling-based algorithm should be faster than k-means. Experimental results in Section 6.2 will demonstrate the speedup of the sampling-based Feel-the-Way algorithm, in which we set $r$=1%.

## 6. Experimental Results

In this section, we will show three sets of experimental results on both synthetic and real-world datasets to evaluate the performance of our algorithms and implementation: 1) In the first set of experiments, we evaluate the effectiveness of different sampling methods; 2) In the second set of experiments, we compare the performance of k-means, full-step and sampling-based Feel-the-Way algorithms in terms of wallclock time and number of iterations; and 3) In the last set of experiments, we evaluate the scalability of our parallel implementation using thousands of CPU cores. Note that in all of the experiments, we use double precision floating point numbers and four-bytes integers to do the computations.

**Computing Platform:** We performed experiments on the BigRed II system located at the Indiana University. BigRed II is a Cray XE6/XK7 HPC system, which consists of 1,020 compute nodes. Each compute node has two 16-core CPUs and 64GB of memory. Table 2 shows detailed information of the system.

TABLE 2: The BigRed II Supercomputer System.

| | |
|---|---|
| Nodes | 1,020 (max 2048 cores per job) |
| Memory per node | 64 GB |
| Processors per node | 2 |
| Cores per processor | 16 |
| Processor | AMD Opteron 6380 2.5GHz |
| Interconnect | Cray Gemini |
| MPI | Cray-MPICH 7.2.5 |

**Datasets:** We use four real-world datasets, and two synthetic datasets. Table 3 shows the information of the four real-world datasets, which are *MNIST*, *CIFAR-10*, *ENRON*, and *PLACES-2*. MNIST is a well-known dataset for hand-writing digit recognition [24]. CIFAR-10 is used for object recognition with 60,000 color images [25]. PLACES-2 is generated from *Places 2* [26], which is a large collection of images from different scenarios. ENRON is a dataset of documents and was initially used for the email classification research [27].

We also use two synthetic datasets in our experiments. Table 4 shows the two synthetic datasets, which are named *SYN* and *SYN-Large*. We generate the synthetic datasets using different Gaussian distributions. In particular, the larger dataset of SYN-Large is generated to measure the scalability of our parallel implementation with many CPUs.

For each experiment, we use five different seeds, and report the average of the measured performances as our performance result. Also, all the Feel-the-Way experiments have comparable SSE costs to the k-means algorithm.

### 6.1. Comparison of Various Sampling Methods

As described in Section 4.3, the sampling-based Feel-the-Way algorithm deploys a sampling approach to the 2nd to

TABLE 3: Specifications of the real-world datasets.

| | MNIST | CIFAR-10 | ENRON | PLACES-2 |
|---|---|---|---|---|
| K | 10 | 10 | 10 | 50 |
| #Data points | 10,000 | 60,000 | 38,400 | 1,024,000 |
| #Coordinates | 784 | 3,072 | 28,102 | 1,024 |
| Dataset size | 17MB | 626MB | 24.7MB | 3.25GB |

TABLE 4: Specifications of the synthetic datasets.

| | SYN | SYN-Large |
|---|---|---|
| K | 10 | 10 |
| #Data points | 6,400 | 204,800 |
| #Coordinates | 1,024 | 2,048 |
| Dataset size | 50.48MB | 3.23GB |

the $L$-th steps, while computing the whole set of points for the first step. Since the effectiveness of the sampling method determines the performance of sampling-based Feel-the-Way, we compare and evaluate five different sampling methods: 1) Random, 2) Max-min, 3) Coefficient of Variation (CV), 4) Heap, and 5) Reassignment-history-aware method (in short, *Reassign-hist*).

In the rest of the subsection, we use the Reassignment-history-aware method as an example to explain how a sampling ratio will affect the convergence rate and execution time. Next, we use a selected small ratio of $1\%$ to compare five sampling methods.

**Effect of the sampling ratio.** We use the parameter of *sampling ratio* to control how many data points should be considered to compute new clustering centers. If the sampling ratio is $p\%$, $p\%$ of the total number of points will be accessed by the algorithm.

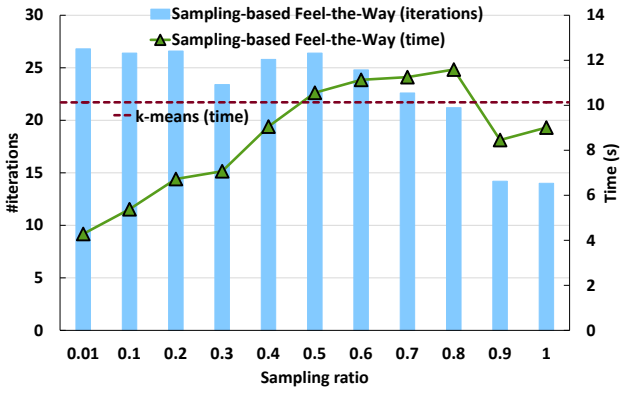



Fig. 2: Effect of the sampling ratio with the MNIST dataset using the Reassignment-history-aware sampling method.

Figure 2 shows how the number of iterations and the execution time will change as the sampling ratio increases from $1\%$ to $100\%$ for the MNIST dataset. Here we use the Reassignment-history-aware sampling and set $L = 4$ to test the Feel-the-Way algorithm. Regarding the number of iterations, when the sampling ratio = $1\%$, Feel-the-Way takes 27 iterations. Then, the number of iterations drops slowly from 27 to 21 as the ratio rises from $1\%$, $10\%$ to $80\%$. Eventually, when the sampling ratio is equal to $90\%$, we see a significant drop in the number of iterations. This experiment tells us that given a specific sampling method, making the sampling ratio smaller does not necessarily degrade the number of iterations greatly (e.g., $20\%$, $10\%$, $1\%$ have nearly the same number of iterations).

On the other hand, the execution time increases more quickly than the speed in which the number of iterations decreases (except for $90\%$ and $100\%$). For instance, the execution time jumps quickly from 4 seconds to 12 seconds when the ratio rises from $1\%$ to $80\%$. This is because a larger sampling ratio implies a larger time complexity for each local step $l$ ($l = 2, 3, 4$). Theorem 5.6 has proved a formula to describe the relationship between time and the number of iterations given a specific $L$.

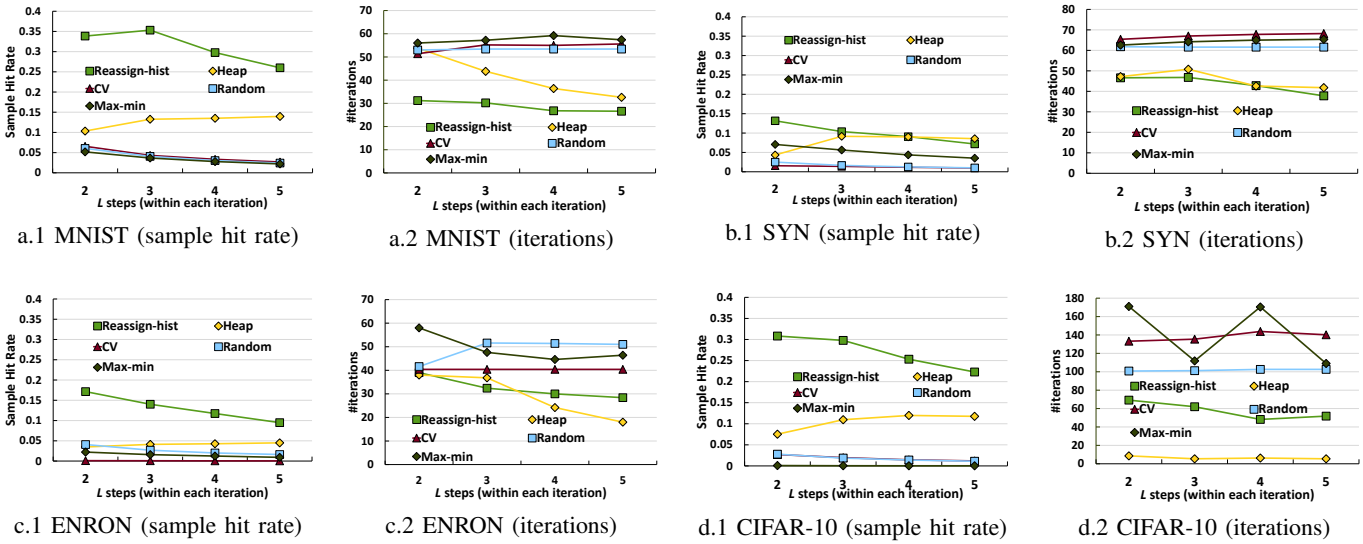
Since our goal is to optimize the execution time, we choose to use a small sampling ratio of $1\%$ by bringing in two benefits: 1) It can lead to a small overhead (around $1\%$) to compute most local steps $l \in \{2 \dots L\}$; and 2) In combination with the first full step of $l = 1$, the sampling-based Feel-the-Way algorithm can achieve a less number of iterations than k-means (i.e., 27 iterations versus k-means' 61.4 iterations, on average).

**Comparison of the five sampling methods.** Next, we test which sampling method can provide the best performance. Our evaluation is based on two metrics: 1) Accuracy. If a sampling method selects $s$ data points, we measure how many points of the selected points have really been assigned to a different cluster (e.g., $h$ points switched clusters). We use $\frac{h}{s}$ to represent the accuracy of the sampling method. We call it "Sampling Hit Rate". The higher the sampling hit rate, the better the sampling method is. The second metric is 2) the number of iterations needed by the Feel-the-Way clustering algorithm to eventually converge.

In Figure 3, we show the experimental results with four datasets: MNIST, SYN, ENRON, and CIFAR-10. For each dataset, we evaluate different sampling methods' *sampling hit rate* and *number of iterations*, respectively.

As shown in Figure 3 a.1, the Reassignment-history-aware method has the best sampling hit rate of 35% when $L = 2$ and 3. The hit rate lowers to 30% and 25% when $L = 4$ and 5. The second best sampling method is the Heap method, which has a sampling hit rate of 15%. The other three methods (Random, CV, and Max-min) only achieve sampling hit rates that are less than 5%. From the perspective of convergence rate, Figure 3 a.2 shows that the Reassignment-history-aware method is the fastest one, taking around 30 iterations. The Max-min sampling method takes 60 iterations, and the CV, Random sampling methods take 51 and 53 iterations. The Heap sampling method converges slowly when $L = 2$, but starts to converge faster when $L$ is bigger. Note that the sampling method that has a higher hit rate is more likely to converge faster.

Figure 3 b.1 shows the SYN dataset's sampling hit rate. When $L > 2$, the heap method is as good as the the reassignment-history-aware method. The Random and Max-




a.1 MNIST (sample hit rate) a.2 MNIST (iterations) b.1 SYN (sample hit rate) b.2 SYN (iterations)

c.1 ENRON (sample hit rate) c.2 ENRON (iterations) d.1 CIFAR-10 (sample hit rate) d.2 CIFAR-10 (iterations)

Fig. 3: Sampling methods comparison on different datasets.

min methods have the lowest sampling hit rate. Correspondingly, Figure 3 b.2 shows that the Reassignment-history-aware method and Heap method have the smallest number of iterations.

Similarly, Figure 3 c.1 and c.2 show the experimental results for the ENRON dataset. The Reassignment-history-aware method has a significantly higher hit rate than the other sampling methods. Please note that when $L = 5$, the Heap method takes less iterations to converge than the Reassignment-history-aware method, but its SSE cost is 4% larger than the reassignment-history-aware.

Figure 3 d.1 and d.2 show the experimental results for the CIFAR-10 dataset. Again, The Reassignment-history-aware method is better than the Heap method, which is better than the CV, Max-min, and Random methods. However, Figure d.2 shows an exception, where the Heap sampling method has the smallest number of iterations. For this case, we find that its SSE cost is larger than that of the Reassignment-history-aware method by 11%. Therefore, the Heap method takes fewer iterations to reach a less optimal solution.

In general, the Reassignment-history-aware sampling method is the most effective method with a higher sampling hit rate and a less number of iterations in most cases. In the following experiments, we choose to use this method to show the performance of the Feel-the-Way algorithm.

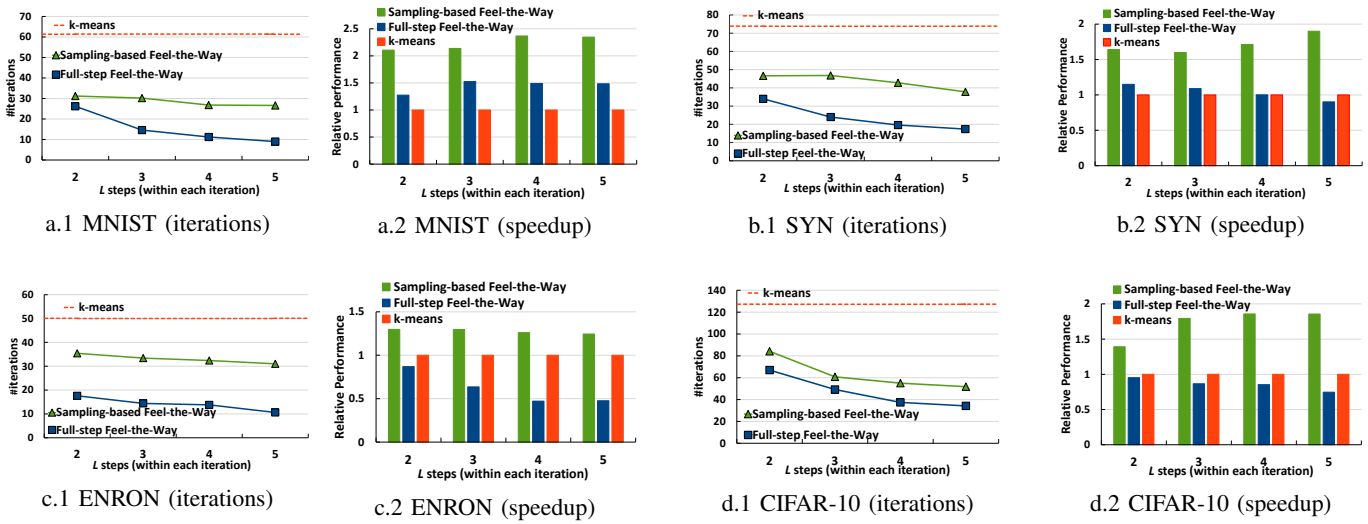
## 6.2. Sequential performance of the full-step and sampling-based Feel-the-Way algorithms

In the second set of experiments, we compare the performance of k-means, full-step Feel-the-Way, and sampling-based Feel-the-Way clustering algorithms. We use three real-world datasets and one synthetic dataset to compare the different clustering algorithms.

Figure 4 shows four groups of subfigures: *a*, *b*, *c*, and *d*, which correspond to four datasets, respectively. The first subfigure in a group shows the number of iterations (i.e., convergence rate), while the second subfigure shows the relative performance speedup over the baseline program of k-means.

Figure 4 a.1 and a.2 show the performance comparison on MNIST. In a.1, the full-step Feel-the-Way method has the least number of iterations because it is able to utilize its $L$ number of local steps to improve the cost. Sampling-based Feel-the-Way has the second least number of iterations due to using a small portion (i.e., 1%) of sampled points. As an example, when $L$ is equal to 5, sampling-based and full-step Feel-the-Way take 27 and 9 iterations to converge. By contrast, the k-means clustering method takes 61 iterations to converge on average. In Figure 4 a.2, we show the performance speedup of the Feel-the-Way algorithms relative to k-means. *Speedup* is computed by the division of two algorithms' execution time. The higher the number, the better the performance is. Thanks to the very small sampling overhead of the sampling-based Feel-the-Way algorithm, it is faster than the full-step Feel-the-way by up to 1.7 times although it requires more iterations to converge (as shown in subfigure a.1). Also, it is faster than k-means by up to 2.4 times.

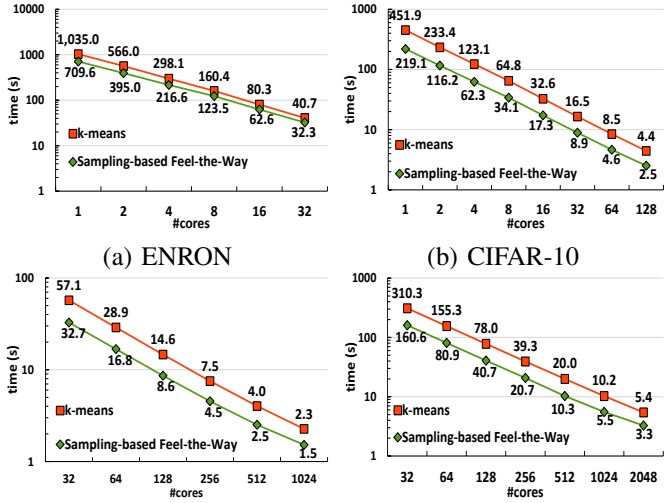
Figure 4 b.1 and b.2 show the performance comparison on the SYN dataset. In the figure, the sampling-based Feel-the-Way algorithm has the second least number of iterations, and achieves the best speedup, which outperforms the k-means method by 1.8 times. In Figure 4 c.2, the full-step Feel-the-way is slower than k-means by 50%, however, the sampling-based Feel-the-way is faster than k-means by 1.3 times. This is because the overhead of the full-step algorithm exceeds the time saved by its less number of iterations for ENRON. In Figure 4 d.2, the sampling-based Feel-the-Way algorithm attains speedups between 1.4 and 1.9 over the k-means method for the CIFAR-10 dataset.




a.1 MNIST (iterations) a.2 MNIST (speedup) b.1 SYN (iterations) b.2 SYN (speedup)

c.1 ENRON (iterations) c.2 ENRON (speedup) d.1 CIFAR-10 (iterations) d.2 CIFAR-10 (speedup)

Fig. 4: Performance comparison on different datasets




(a) ENRON (b) CIFAR-10

(c) SYN-Large (d) PLACES-2

Fig. 5: Strong scalability experiments. The y-axis is shown in the logarithmic scale.

### 6.3. Scalability of the parallel sampling-based Feel-the-Way algorithm

Finally, we perform large scale experiments to evaluate the scalability of our parallel implementation for the sampling-based Feel-the-Way algorithm. In the parallel experiments, we take as input four datasets: ENRON, CIFAR-10, SYN-Large, and PLACES-2.

Figure 5 shows the performance of strong scalability. As we increase the number of CPU cores, the execution time will decrease correspondingly. Figure 5.a displays the execution time of k-means and sampling-based Feel-the-Way on the ENRON dataset. As the number of CPU cores increases from 1 to 32, our parallel Feel-the-Way reduces the execution time from 709.6 to 32.3 seconds while k-means reduces it from 1,035 to 40.7 seconds. Figure 5.b displays the execution time when using the CIFAR-10 dataset. On 128 CPU cores, our parallel implementation is faster than k-means by 176%.

In Figure 5.c, we use 1,024 CPU cores to compute the SYN-Large dataset. From the subfigure c, we can see that the parallel Feel-the-Way implementation reduces the execution time from 32.7 to 1.5 seconds using 1,024 cores, outperforming k-means by 153%. As for the PLACES-2 dataset as shown in (d), the wallclock execution time is decreased from 160.6 to 3.3 seconds when using a number of 2,048 CPU cores. The parallel sampling-based Feel-the-Way implementation is able to outperform the k-means method by 164%.

## 7. Conclusion

In this paper, we seek to design and develop a fast clustering algorithm for large-scale HPC systems. We first design the full-step Feel-the-Way algorithm. Compared to k-means, this algorithm takes significantly less number of iterations by applying local optimization to each data block, meanwhile providing the same cost as the k-means algorithm. Next, to optimize the execution time of the full-step algorithm, we introduce different sampling methods and design a new algorithm called sampling-based Feel-the-Way. By sampling a few useful data points, the new sampling-based algorithm has a much better performance than the full-step algorithm. Five sampling methods are designed and tested, among which the reassignment-history-aware sampling method achieves the best convergence rate. Our future work along this line will design sampling-based parallel Feel-the-Way algorithms for other machine learning and deep learning methods on extreme-scale HPC systems.

## References

[1] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM computing surveys (CSUR)*, vol. 31, no. 3, pp. 264–323, 1999.

[2] J. A. Hartigan and J. Hartigan, *Clustering algorithms*. Wiley New York, 1975, vol. 209.

[3] A. Frommer and D. B. Szyld, "On asynchronous iterations," *Journal of computational and applied mathematics*, vol. 123, no. 1, pp. 201–216, 2000.

[4] J. W. Demmel, L. Grigori, M. F. Hoemmen, and J. Langou, "Communication-optimal parallel and sequential QR and LU factorizations," UTK, LAPACK Working Note 204, August 2008.

[5] R. Bru, L. Elsner, and M. Neumann, "Models of parallel chaotic iteration methods," *Linear Algebra and its Applications*, vol. 103, pp. 175–192, 1988.

[6] F. Song, H. Ltaief, B. Hadri, and J. Dongarra, "Scalable tile communication-avoiding QR factorization on multicore cluster systems," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10)*, 2010, pp. 1–11.

[7] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing, "More effective distributed ML via a stale synchronous parallel parameter server," in *Advances in neural information processing systems*, 2013, pp. 1223–1231.

[8] M. G. Tallada, "Coarse grain parallelization of deep neural networks," in *ACM SIGPLAN Notices*, vol. 51, no. 8. ACM, 2016, p. 1.

[9] W. Zheng, F. Song, and L. Lin, "Designing a synchronization-reducing clustering method on manycores: Some issues and improvements," in *Proceedings of the Machine Learning on HPC Environments*. ACM, 2017, p. 9.

[10] S. Shi, Q. Wang, P. Xu, and X. Chu, "Benchmarking state-of-the-art deep learning software tools," in *7th International Conference on Cloud Computing and Big Data (CCBD)*. IEEE, 2016, pp. 99–104.

[11] H. Kurban and M. M. Dalkilic, "A novel approach to optimization of iterative machine learning algorithms: Over heap structure," in *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 2017, pp. 102–109.

[12] Apache Mahout, "https://mahout.apache.org/," 2017.

[13] MLlib, "http://spark.apache.org/mllib/," 2017.

[14] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[25] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[15] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed GraphLab: a framework for machine learning and data mining in the cloud," *Proceedings of the VLDB Endowment*, vol. 5, no. 8, pp. 716–727, 2012.

[16] Amazon Machine Learining, "https://aws.amazon.com/aml/details/," 2017.

[17] A. Gittens, A. Devarakonda, E. Racah, M. Ringenburg, L. Gerhardt, J. Kottalam, J. Liu, K. Maschhoff, S. Canon, J. Chhugani *et al.*, "Matrix factorizations at scale: A comparison of scientific data analytics in Spark and C+MPI using three case studies," in *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 2016, pp. 204–213.

[18] D. A. Reed and J. Dongarra, "Exascale computing and big data," *Communications of the ACM*, vol. 58, no. 7, pp. 56–68, 2015.

[19] G. Di Fatta, F. Blasa, S. Cafiero, and G. Fortino, "Fault tolerant decentralised k-means clustering for asynchronous large-scale networks," *Journal of Parallel and Distributed Computing*, vol. 73, no. 3, pp. 317–329, 2013.

[20] S. Shi, Q. Wang, P. Xu, and X. Chu, "Benchmarking state-of-the-art deep learning software tools," in *7th International Conference on Cloud Computing and Big Data (CCBD)*. IEEE, 2016, pp. 99–104.

[21] Y. You, A. Buluç, and J. Demmel, "Scaling deep learning on gpu and knights landing clusters," in *Proceedings of the 2017 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'17)*. ACM, 2017, p. 9.

[22] D. Sculley, "Web-scale k-means clustering," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 1177–1178.

[23] C. A. Hoare, "Algorithm 65: find," *Communications of the ACM*, vol. 4, no. 7, pp. 321–322, 1961.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[26] B. Zhou, A. Khosla, A. Lapedriza, A. Torralba, and A. Oliva, "Places: An image database for deep scene understanding," *arXiv preprint arXiv:1610.02055*, 2016.

[27] B. Klimt and Y. Yang, "The enron corpus: A new dataset for email classification research," in *European Conference on Machine Learning*. Springer, 2004, pp. 217–226.