Designing a Parallel Feel-the-Way Clustering Algorithm on HPC Systems*

Journal Title
XX(X):1–14

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Weijian Zheng[1], Dali Wang[2], Fengguang Song[1]

Abstract

This paper introduces a new parallel clustering algorithm, named Feel-the-Way clustering algorithm, that provides better or equivalent convergence rate than the traditional clustering methods by optimizing the synchronization and communication costs. Our algorithm design centers on how to optimize three factors simultaneously: reduced synchronizations, improved convergence rate, and retained same or comparable optimization cost. To compare the optimization cost, we use the Sum of Square Error (SSE) cost as the metric, which is the sum of the square distance between each data point and its assigned clusters. Compared with the traditional MPI k-means algorithm, the new Feel-the-Way algorithm requires less communications among participating processes. As for the convergence rate, the new algorithm requires fewer number of iterations to converge. As for the optimization cost, it obtains the SSE costs that are close to the k-means algorithm. In the paper, we first design the full-step Feel-the-Way k-means clustering algorithm that can significantly reduce the number of iterations that are required by the original k-means clustering method. Next, we improve the performance of the full-step algorithm by adopting an optimized sampling-based approach, named *reassignment-history-aware sampling*. Our experimental results show that the optimized *sampling-based Feel-the-Way method* is significantly faster than the widely used k-means clustering method, and can provide comparable optimization costs. More extensive experiments with several synthetic datasets and real-world datasets (e.g., MNIST, CIFAR-10, ENRON, and PLACES-2) show that the new parallel algorithm can outperform the open source MPI k-means library by up to 110% on a high performance computing system using 4,096 CPU cores. In addition, the new algorithm can take up to 51% fewer iterations to converge than the k-means clustering algorithm.

Keywords

Parallel machine learning algorithms, communication and synchronization reducing, distributed clustering methods, high performance computing

1 Introduction

High-performance computing (HPC) and machine learning (ML) have been widely adopted by both academia and industries to address the enormous data problems at extreme scales. Research has been reported on the interactions of HPC and machine learning, however, achieving high performance and scalability for parallel and distributed machine learning algorithms is still a challenging task. This paper takes the k-means clustering method as an example to introduce a new type of parallel machine learning algorithms, named Feel-the-Way algorithms. The k-means clustering method is one of the most widely used big data analysis methods due to its effectiveness, clean and simple design Jain et al. (1999); Hartigan and Hartigan (1975). It is used to solve global optimization problems by pushing each point towards a local minimum repeatedly. Similar to many other machine learning methods (e.g., random search based methods and generalized descent methods), the clustering methods are also affected by local minima, convergence rate, seeding, parameter staleness, quality of the solution, and computation and communication costs.

One of the key factors to achieve high performance on extreme-scale HPC systems is to reduce the communication cost and the synchronization cost associated with the parallel algorithms. Researchers have conducted extensive studies to design communication-reducing and synchronization-reducing algorithms Frommer and Szyld (2000); Demmel et al. (2008); Bru et al. (1988); Song et al. (2010); Demmel and Dinh (2018). While these algorithms can reduce the communication cost and synchronization cost, they can also cause slower convergence rates, lower quality of solutions, or even divergence Ho et al. (2013); Tallada (2016); Zheng et al. (2017); Shi et al. (2016). Consequentially, the overall execution time of the modified parallel algorithm has prolonged because the increased number of iterations (i.e., slower convergence) overwhelm the benefits of improved performance per iteration.

Feeling intrigued by this problem, we strive to study whether we can design a new type of machine learning algorithms that can not only *reduce the synchronization and communication cost* but also achieve *better convergence*. Our goal is to find the best combination of these two factors:

[1]Department of Computer Science, Indiana University-Purdue University Indianapolis, Indiana
[2]Oak Ridge National Laboratory, Tennessee

Corresponding author:
Fengguang Song, Indiana University-Purdue University Indianapolis
Email: fgsong@cs.iupui.edu

The new algorithms can attain minimized synchronizations meanwhile achieving the same or faster convergence rate.

To that end, we design the new type of Feel-the-Way algorithms. The motivation of the algorithms is that each machine learning process behaves like a hiker who is trying to "cross an unknown river by touching or feeling the rocks in the riverbed, and thinking of both speed and safety." In such a metaphor, the current rock is the safe "harbor" for which a person can always come back even though all the surrounding rocks don't work out well. Also when many persons (i.e., processes in the algorithm) are crossing the same river, information sharing from time to time will lead to a better solution faster. Following this philosophy, we design the so-called Feel-the-Way algorithms.

In the Feel-the-Way algorithm that is applied to k-means clustering as an example, each process undertakes a part of the dataset, and runs L local steps before synchronizing its local solution with other processes. The novelty of the Feel-the-Way algorithm lies in that we view the first step as the "safe" point and the rest L-1 steps as opportunities to enhance the first step. The rationale is that the solution from the first step is always a "decent" candidate for the global solution. If the second, third, ..., or L-th step 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. Considering an extreme case, even 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 ML method because the first step will be the "winner" as selected by the Feel-the-Way algorithm.

Our first attempt to realize the algorithm shows promising results. Experiments in Section 7.2 demonstrate that using L=5 can reduce the average number of iterations from 61 to 10 with the MNIST dataset. However, we need to solve the following challenge: the time spent on the second to the L-th local steps must be significantly short to reduce the total execution time. Otherwise, the saved iterations may be overwhelmed by the time spent on the second to L-th local steps. Hence, we consider the second to L-th local steps as the cost we pay in order to reduce the number of iterations.

In order to reduce the time spent on the second to L-th steps, we use a number of sampling methods to avoid re-clustering all the data points multiple times. Based on our analysis, not every point is reassigned in every local step. 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 Kurban and Dalkilic (2017), and 5) a new reassignment-history-aware sampling method. These five sampling methods have different time complexities and distinct efficiencies on convergence. All the sampling methods are controlled by a sampling rate r, and each local step may use a different sampling rate. Eventually, we call the new ML algorithm "sampling-based Feel-the-Way algorithm". To differentiate from the original algorithm, we refer to the original algorithm (i.e., without sampling) as "full-step Feel-the-Way algorithm".

We have designed and implemented sequential algorithms for both the full-step Feel-the-Way algorithm and the sampling-based Feel-the-Way algorithm. 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. Note that the sampling-based algorithm only computes a sampled 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 design and 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 its centers with other threads' centers once every L steps.

We conduct a number of experiments with multiple synthetic datasets and real-world datasets of MNIST, CIFAR-10, ENRON, and PLACES-2 on HPC computers. We evaluate the new algorithm with respect to performance, convergence rate, and accuracy. The metric of performance is measured by the wall-clock execution time. Convergence rate is measured by the number of iterations the algorithms take to converge. Finally, accuracy is measured by the Sum of Squared Error (SSE) cost that measures the differences between each point and its center within each cluster.

The experimental results show that the new *reassignment-history-aware sampling method* is more effective than the other sampling methods, and the new parallel Feel-the-Way algorithm can outperform the open source MPI k-means library from Northwestern University Liao (2017) significantly. To the best of our knowledge, this paper makes the following main contributions:

1. We propose a full-step Feel-the-Way algorithm and a sampling-based Feel-the-Way algorithm (in both sequential and parallel implementations) 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* .
3. Our experiments with real-world datasets demonstrate that the type of Feel-the-Way algorithms can reduce the number of synchronizations but without slowing down the convergence rate, which has been scarcely studied.

2 Related work

In this section, we list the related work in four categories: 1) parallel clustering algorithms, 2) existing software that supports clustering, 3) different asynchronous methods, and 4) relevant sampling methods.

Clustering algorithms: The k-means clustering method is one of the most widely used data mining algorithms. One type of clustering work tries to avoid computing the distance between points and the new centers. For instance, triangle

equality based methods, such as Moore (2000), Elkan (2003) and Hamerly (2010), can eliminate the distance computation by applying geometry knowledge. Elkan's algorithm Elkan (2003) uses the extra information of inter-center distances to decide whether a new point-to-center distance computation is necessary. Another type of work focuses on using tree-based (e.g., kd-tree, anchor hierarchy) data structures to achieve faster indexing and computations Alsabti et al. (1997); Kanungo et al. (2002); Pelleg and Moore (1999). To obtain high performance on large datasets, researchers have also done extensive research on parallel clustering algorithms, which include MapReduce based algorithms Zhao et al. (2009); Jin et al. (2016), and MPI based algorithms Kerdprasop and Kerdprasop (2010); Kantabutra and Couch (2000); Dhillon and Modha (2002). In our work, we use the hybrid MPI/Pthread programming model to implement the new parallel sampling-based Feel-the-Way algorithm.

Existing clustering software: There are a number of Python libraries for k-means and other machine learning algorithms such as Scikit-learn Pedregosa et al. (2011), Orange Demšar et al. (2013), Mlpy Albanese et al. (2012) and Pylearn2 Goodfellow et al. (2013). These libraries mainly target programmer productivity instead of high performance, especially on distributed memory systems. There are also machine learning frameworks such as Mahout Apache Mahout (2017), Spark MLlib MLlib (2017), Google TensorFlow Abadi et al. (2016), GraphLab Low et al. (2012) and Amazon AML Amazon Machine Learning (2017). They provide high productivity by using Python or Java on Cloud platforms. Unfortunately, their performances are still suboptimal on HPC systems. 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 Gittens et al. (2016). Differently, we focus on high performance and design a new parallel clustering algorithm for HPC systems using MPI and Pthread.

Asynchronous methods to reduce communications: Communication reduction has drawn a lot of attention from researchers when designing parallel machine learning algorithms. The *lazy synchronization* method has been applied to many machine learning algorithms. The basic idea is to allow all tasks to execute independently and have a lesser need for synchronization. In the work of Di Fatta et al. (2013), each computer independently computes its local points' assignment and centers. Hsieh et al. (2014) and You et al. (2015) accelerate SVM by asynchronously training models on each partition of the data. Several general purpose machine learning frameworks also support lazy synchronization Ho et al. (2013); Li et al. (2014); Low et al. (2012); Tallada (2016)

However, the lazy synchronization method often leads to slower convergence and lower accuracy due to using stale parameters. A few studies have investigated the issue Ho et al. (2013); Li et al. (2014); Zheng et al. (2017). Xing et al. studied machine learning applications that use the stochastic gradient descent (SGD) method, and adaptively change the SGD step size to compensate for errors caused by stale synchronizations Ho et al. (2013). They have implemented asynchronous data communication using parameter servers. Furthermore, Zheng et al. (2017) employ simulated annealing to accelerate k-means clustering and offer the same quality as the synchronous clustering method.

We share the same philosophy of minimizing synchronizations with the existing work, but besides reducing communication, we also target how to design new approaches to preventing convergence rates from getting slower.

Related sampling methods: A number of sampling based methods have been developed to cluster large scale datasets faster. A simple approach is to find the clustering centers based on a small set of data points. The work of Sculley (2010) randomly selects a small number of data points from each batch in every iteration. Similar approaches have been introduced by Havens et al. (2012); Har-Peled and Mazumdar (2004); Chitta et al. (2011). Chen and Cai (2011) and Wang et al. (2011) also apply the sampling ideas to the spectral and pairwise clustering algorithms. Kurban and Dalkilic (2017) design the *heap* sampling method, and apply it to k-means clustering and achieve higher performance. Another approach prunes away irrelevant and redundant dimensions from data points to improve performance Elankavi et al. (2017).

In this paper, we design and test five different sampling methods and incorporate them (as a submodule) into our new parallel Feel-the-Way algorithm. The Feel-the-Way algorithm is a hybrid of sampling algorithm and the original synchronous clustering algorithm. It is able to obtain almost the same optimization cost as the original k-means. In addition, we compare the *heap* method with our new *reassignment-history based* sampling method, and show that the reassignment-history-aware method can result in a better convergence rate than the heap method.

3 The 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 communication.

The algorithm consists of three functions: 1) The local optimization function that optimizes each data block; 2) the merge function that derives the global clustering centers from different blocks; and 3) the main function that controls when to stop the algorithm. The rest of the section will introduce the three functions in details.

3.1 Local optimization function

Underlying data structure: We use a simple *blocked* data layout to store the input dataset. Each data point is a vector with m attributes. Assuming we have n data points, a dataset can be viewed as an $n \times m$ matrix. In our algorithm, we divide the matrix into blocks of rows, where each *block* represents a set of b consecutive data points. This blocked data structure is used to implement our Feel-the-Way algorithms.

The local optimization function is responsible for reducing each individual block's optimization cost. The basic idea is as follows: we run one step of the synchronized k-means algorithm (i.e., the original k-means), then we run L-1 asynchronous steps upon each block. The additional L-1

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 centers ← g_centers; end if
10:     ▷ stage 2: Use all local points to improve centers
11:     for each point i in points do
12:         ▷ stage 2.1 find the closest center for point i
13:         (dist, new_center) ← find_nearest_center(points[i], centers);
14:         membership[i] = new_center;
15:         ▷ stage 2.2 update each block's partial sum and size
16:         blk_local_sum[new_center] += points[i];
17:         blk_local_size[new_center]+=1;
18:         blk_cost_new += dist;
19:     end for
20:     ▷ stage 2.3: Recalculate new centers and new cost
21:     for each center c do
22:         centers[c] = blk_local_sum[c] / blk_local_size[c];
23:     end for
24:     if (l = 0) then 1st_step_cost ← blk_cost_new; end if
25:     l++;
26: end while
27: ▷ stage 3: Returns if L steps are finished
28: return {blk_local_sum, blk_local_size, 1st_step_cost}
```

steps on each block are designed to reduce each block's SSE (Sum of Squared Error) cost monotonically.

In Algorithm 1, we present the pseudocode of the local optimization function, which goes through three stages. *Stage 1*: Set the current block's clustering seed as the newly merged global centers. *Stage 2*: Use the current block's data points to improve the block's clustering cost. In stage 2.1, find the closest center for each data point; In stage 2.2, compute each block's local centers' sizes and sums of coordinates; In stage 2.3, calculate new local centers. *Stage 3*: The local optimization function returns updated clustering results after a number of L local steps have been applied to the current block. Note that we need to keep the first step's cost, which is just equal to the SSE cost from the previous global iteration.

3.2 The merge function

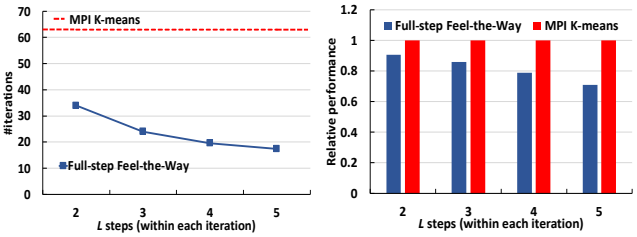
After calling the local optimization function, each data block has its own set of centers. Therefore, N data blocks will have N set of centers. Given each data block's *local_sum*, *local_size* and a new cost *local_cost*, the merge function will add them together to get the *global_sum* and *global_size*,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

3.3 The main function

Algorithm 2 shows the main function of the full-step Feel-the-Way algorithm. It calls the previous local optimization and the 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 (i.e., lines 10-18). After all the data blocks have been processed once, the new global centers can be computed (i.e., lines 20-22). Here, we use the

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_new; g_size_new = 0; g_sum_new = 0;
9:      pre_g_cost_old = pre_g_cost_new; pre_g_cost_new = 0;
10:     for each block i ← 0 to n_b - 1 do
11:         /* Run local optimization on each block */
12:         (local_sum, local_size, pre_local_cost) ←
13:         local_optimization_block(i-th block, g_centers,
14:         g_size_old, membership, L);
15:         /*Merge each block's coordinate sum into global sum*/
16:         (g_sum_new, g_size_new, pre_g_cost_new) ←
17:         merge(local_sum, local_size, pre_local_cost);
18:     end for
19:     /* Compute new global centers */
20:     for each center c do
21:         g_centers[c] = g_sum_new[c] / g_size_new[c];
22:     end for
23: until pre_g_cost_old - pre_g_cost_new ≤ threshold
```



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

Figure 1. Comparison between the full-step Feel-the-Way algorithm and the MPI k-means algorithm.

SSE cost as the program's stop condition in order to achieve a minimal SSE cost.

4 Sampling-Based Feel-the-Way Algorithm

In this section, we first analyze the performance of the full-step Feel-the-Way algorithm. Next, we extend it to a new sampling-based Feel-the-Way algorithm, which is motivated by the unscalable performance of the full-step algorithm, In the end, we design and compare five different types of sampling approaches.

4.1 Motivation for a new sampling-based Feel-the-Way algorithm

We use a synthetic dataset SYN (described in Table 5) to evaluate the performance of the full-step Feel-the-Way algorithm and the MPI k-means algorithm. Table 1 shows their corresponding SSE costs, which are within 1% difference. Also, Figure 1.a shows the number of iterations and Figure 1.b shows their relative performance. For all the MPI k-means experiments, we use the open source *Parallel K-means Data Clustering Library*, which was developed by Northwestern University Liao (2017). In this paper, we refer to the open source clustering library as *MPI k-means*.

Figure 1.a reveals that the full-step Feel-the-Way algorithm can significantly reduce the number of iterations.

Table 1. Costs of the MPI k-means and Full-step Feel-the-Way experiments, whose performance are shown in Figure 1.

Dataset	L = 2	L = 3	L = 4	L = 5
SYN	6.3433e9	6.3443e9	6.3451e9	6.3464e9
	k-means cost: 6.2645e9			

For instance, MPI k-means has 63 iterations while the full-step Feel-the-Way has 17 iterations when L=5.

In Figure 1.b, however, we find that the full-step Feel-the-Way is not faster than MPI k-means, even though Feel-the-Way has a less number of iterations than k-means. The reason is that the time spent between the second and *L-th* local step is expensive, which eventually becomes larger than the benefit of the saved iterations. Later in Lemma 1, we 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.

Puzzled by the problem, we hope to create a new algorithm, which ideally spends little or no time on the local steps (from the second to the L-th steps), at the same time reducing the number of iterations. Our in-depth analysis of point reassignments between different clusters finds that, not all data points have been reassigned to a distinct cluster. Moreover, fewer and fewer data points are reassigned from the first iteration to the last iteration. As shown in the following Figure 2, we measure the number of reassigned data points at each iteration on MNIST dataset using MPI k-means. We can observe that the number of reassigned points decrease from 60,000 to 0.

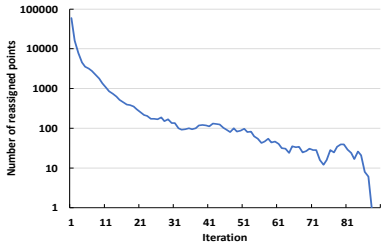


Figure 2. Number of reassigned data points with the MNIST dataset using the k-means at each iteration. K-means needs 89 iterations.

By intuition, if we could just pick up those reassigned points (and skip those unchanged points), the Feel-the-Way algorithm can certainly run much faster. To that end, we introduce different sampling methods to the full-step Feel-the-Way algorithm's local steps to intelligently select a small portion of points to compute clustering, rather than considering all points (as done by the full-step Feel-the-Way algorithm).

4.2 Design of the algorithm

The main function of the sampling-based Feel-the-Way algorithm is almost the same as the full-step Algorithm 2 except we introduce a new local optimization function.

The new local optimization function is presented in Algorithm 3. In line 9, the 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 14, each sampled point is reassigned to its closest center, and those consequentially affected cluster's coordinate sum and number of points are updated accordingly. Note that this algorithm considers all the points when it is the first local step, as stated in line 10.

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 rest of $L-1$ local steps to improve the cost of the first local step. In addition, the sampling algorithm utilizes the first local step to compute the global SSE cost for all blocks of points, and determines which points should be sampled based upon the reassignment statistics of the first local step. Furthermore, the blk_local_sum and blk_ local_size do not need to be reset to zero in every new local step. Instead, they will be quickly updated based on the previous local step.

4.3 Development of five sampling methods

We develop five different sampling methods. These sampling methods utilize two types of information: 1) the distance of a data point to each center: near or far; and 2) the history of previous steps that reassign certain points to different centers.

The five sampling methods are described as follows:

1. *Random sampling*: This is the simplest one. It just 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* ratio (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 an assumption that if a data point is equally close to all k centers, it is more likely that this point will change its nearest center.

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:  l = 0 /* l is the current step */
5:  while l ≤ L − 1 do
6:      /*same as lines 6-9 of Algorithm 1 */
7:      ▷ stage 2: Use sampled local points to improve centers
8:      for each point i in points do
9:          if (sampled_points > max_sample_points) then break; 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(points[i], centers);
14:             membership[i] = new_center;
15:             ▷ stage 2.2 update each block's partial sum and size
16:             blk_local_sum[new_center] += points[i];
17:             blk_local_size[new_center]+=1;
18:             if (l != 0 ) then
19:                 blk_local_sum[old_center] -= points[i];
20:                 blk_local_size[old_center]-=1;
21:             end if
22:         end if
23:     end for
24:     /*same as lines 20-25 of Algorithm 1 */
25: end while
26: ▷ stage 3: Returns if L steps are finished
27: return {blk_local_sum, blk_local_size, 1st_step_cost}
```

3. *Coefficient of Variation (CV)*: CV is defined as $\frac{\text{standard deviation}}{mean}$. Each data point has k distances to k centers, respectively. This method computes the CV of those k distances. A small CV value reveals that the distances of the point to all k centers are almost the same. Hence, those points that have smaller CVs will be selected because we anticipate that a point with the same distances to all centers is likely to be reassigned soon (i.e., unsettled yet).
4. *Heap sampling*: Heap-based sampling was introduced by Kurban and Dalkilic (2017). 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, it always selects the points that are at the high levels of the heap. The heap-based method assumes that the points that are far from their belonging center will switch to a new center soon.
5. *Reassignment-history-aware sampling*: This is a new sampling method proposed by us. Reassignment-history-aware sampling keeps track of which data points have been reassigned in the past. If a point was assigned to a new center in the previous step, we consider it as a good sampling candidate based on the assumption that it is a "hot" point and will change to a new center again in the current step. In fact, we have observed this behavior in our clustering experiments, where "hot" points are indeed unsettled.
   If there are m points that have been reassigned to new centers in the previous step, we will randomly pick s sample points from the set of m points. Usually m is larger than s. s can be either a constant or decided by a sampling ratio. When m becomes too small (e.g., less than 20), we will choose all m points as the sampling points.

Time complexity of the sampling methods:
Suppose each sampling method needs to select s sample points from each data block, given k centers and each data block with b points, we list the time complexities of all the sampling methods in Table 2 For the Random and Reassignment-history-aware methods, their time complexities are equal to $s \times O(1)$. For the Max-min and CV sampling methods, we invoke the QuickSelect algorithm Hoare (1961) s times to find the best points. Since QuickSelect has an average time complexity of $O(b)$. The overall time complexity is $O(sb)$ for the Max-min and CV methods. As to the Heap method, it takes $O(logb) \times s$ to maintain a heap data structure.

Table 2. Time complexity of different sampling methods.

Random	Max-min	Heap	CV	Reassignment-history-aware
$O(s)$	$O(sb)$	$O(slogb)$	$O(sb)$	$O(s)$

5 Parallel Implementation

We have implemented the Feel-the-Way algorithm on distributed-memory multicore HPC systems using the hybrid MPI/Pthread programming model. Algorithm 4 essentially parallelizes Algorithm 2 by distributing data blocks across MPI processes and then Pthreads. As shown in Algorithm 4, the parallel program executes the following three steps:

1. *Processes read and distribute data*: As shown in lines 5-13, the root process will first distribute the dataset by reading a file and sending data points to the other processes. Assume there are m data points and P processes, each process will take m/P data points. Then, each thread will obtain its data points according to the block size B. As shown in line 20, given T threads per process, each thread will get $m/B/P/T$ blocks.
2. *Each thread executes the local_optimization_block() kernel*: As shown in line 22, each thread will optimize its own blocks of data points sequentially by calling Algorithm 3.
3. *Merge the results of each thread*: Finally, all the results from different threads will be merged. In this step, the algorithm uses a multilevel merging method to compute the global centers. All the blocks within a thread will be merged in the first level to obtain the thread-local results. Next, in the second level, all results from different threads within the same process will be merged to obtain the process-local results. At the last level, different processes will merge their results to obtain the global centers and cost. The global cost will be used to decide whether the algorithm should terminate or not.

6 Theoretical Analysis

The previous sections 3 and 4 have presented how the full-step and sampling based Feel-the-Way algorithms work. In this section, we will analyze the algorithms in order to answer two questions: 1) How can we compute the new SSE

Algorithm 4 Parallel sampling-based Feel-the-Way clustering algorithm

```
1:  parallel_feelway_clustering(points, m, B, k, threshold, P, T)
2:  /* B: block size */
3:  /* P: #MPI processes; T: #threads/process */
4:  ▷ Read and distribute data points
5:  if pid == 0 then
6:      n_b = m / B / P; /*#blocks per process*/
7:      Read a file and send a subset of data points to each process
8:      /* select first k data points as initial cluster centers */
9:      for each center c ← 0 to k-1 do
10:         g_centers[c] = points[c]
11:     end for
12:     Broadcast g_centers to the other processes
13: else
14:     Receive a subset of points from P0, stored in points_proc
15: end if
16: g_cost_new = MAXIMUM
17: repeat
18:     ▷ Each process has T threads
19:     for each thread tid ← 0 to T do
20:         Distribute points_proc to each thread, stored in points_thrd
21:         Call local_optimization_block() kernel on tid's own points_thrd.
22:     end for
23:     g_cost_old = g_cost_new
24:     Merge all processes' results and compute g_cost_new
25: until g_cost_old - g_cost_new ≤ threshold
```

cost correctly by only considering a set of sampling points? 2) In what conditions will the Feel-the-Way algorithm be faster than the conventional k-means method?

6.1 Fast SSE cost computation

For each local step l that requires sampling points (i.e., $l \in \{1 \dots L-1\}$), we need to compute the new SSE cost at the end of the l-th step. Note that we cannot recompute every point's distance to its center due to 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 *unsampled* points.

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

Also suppose $cost(S)$ is the SSE cost of the sampled point. It can be easily computed or updated whenever a sampled point is reassigned to the closest center.

The SSE cost of unsampled points (i.e., $cost(U)$) is more complex and can be computed as follows. The basic idea is that we utilize the old SSE cost of unsampled data points $cost(U)_{old}$ to compute the new cost of unsampled data points. More 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 points U, we keep their coordinates sums and the number of points attached to each center from the previous step, denoted as sum and $size$.

Provided $\sigma_i = C'_i - C_i$, where C_i is the i-th center, then the corresponding SSE cost computation can be expressed by the following formula:

$$\begin{aligned} cost(V) &= cost(U) + cost(S) \\ &= cost(U)_{old} - 2\sum_{i=1}^{k}(sum_i - size_i \cdot C_i) \cdot \sigma_i \\ &\quad + \sum_{i=1}^{k}(size_i(\sigma_i \cdot \sigma_i)) + cost(S) \end{aligned} \tag{1}$$

6.2 Speedup analysis for the full-step and sampling-based Feel-the-Way algorithms

Lemma 1. *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,

$$\begin{aligned} T_{k-means} \geq T_{full-step} &\implies t \times m \geq t \times n \times L \\ &\implies m \geq n \times L \end{aligned} \tag{2}$$

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

Theorem 1. *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

$$\begin{aligned} T_{k-means} \geq T_{sample} &\implies t \times m \geq (t + (L-1) \times r \times t) \times n \\ &\implies m \geq n \times (1 + (l-1) \times r) \end{aligned} \tag{3}$$

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

Theorem 1 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 sampling ratio r, the sampling-based algorithm will be faster than k-means as long as the Feel-the-Way algorithm has fewer iterations. Experimental results in Section 7.2 demonstrate the speedup of the sampling-based Feel-the-Way algorithm, in which r=1%.

7 Experimental Results

In this section, we will show three sets of experimental results with both synthetic and real-world datasets to evaluate the performance of our algorithms: 1) Evaluation of the effectiveness of different sampling methods; 2) Performance comparison between an open source MPI k-means library, the full-step, and the sampling-based Feel-the-Way algorithms in terms of wallclock time and number of iterations; and 3) Scalability experiments with our parallel implementation on thousands of CPU cores. In all the experiments, we use double-precision floating point numbers and 32-bits integers to perform computations.

The open source MPI k-means library: The MPI k-means library was developed at Northwestern University Liao (2017). Since the MPI library only computes single-precision floating numbers and does not terminate the algorithm based on the SSE cost, we modify the source code slightly to make a fair comparison with our algorithms. The following three parts have been revised. Also, readers can find the revised source code at `https://github.com/zwj3652011/MPI_kmeans.git`

1. *Single precision to double precision*: Single-precision floating numbers are used by the original MPI k-means library. However, our software uses double-precision floating point numbers. Thus, we convert the single-precision numbers in the MPI library to double precision numbers.
2. *Stop condition is now decided by the SSE cost*: In the original MPI k-means library, the algorithm will stop when there are very few data points whose assignment is still changing. Our algorithm will stop when the SSE cost cannot be significantly reduced. We change the stop condition of the MPI k-means library to use the SSE cost. Thus, by setting the same stop condition threshold, our algorithm and the MPI k-means library can obtain comparable SSE costs.

3. *A new input function to read dataset files*: For certain data formats that are not supported by the MPI k-means library (such as the *document* dataset used in our experiments), we have developed a new input file reader to support the formats.

HPC 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 3 shows detailed information about the system.

Different datasets: Our experiments use four real-world and two synthetic datasets. Table 4 describes the four real-world datasets: *MNIST*, *CIFAR-10*, *ENRON*, and *PLACES-2*. MNIST is a well-known dataset for hand-writing digit recognition LeCun et al. (1998). CIFAR-10 is used for object recognition with 60,000 color images Krizhevsky and Hinton (2009). PLACES-2 is generated from *Places 2* Zhou et al. (2016), which is a large collection of images from different scenarios. ENRON is a set of documents and used in the email classification research area Klimt and Yang (2004).

In addition, our experiments use two synthetic datasets. Table 5 shows the two synthetic datasets, which are named *SYN* and *SYN-Large*. The synthetic datasets are generated using different Gaussian distributions, for which the interval between each center coordinate value is set to 100 and the radius of center coordinate value is set to 50. The steps of generating synthetic dataset is as follows: 1) Generate a random number using a uniform distribution and decide the membership for this data point. Each data point has $1/k$ possibility of assigning to one of the centers. 2) Based on the Gaussian distribution, we will generate a random number r for each data points at every coordinate. 3) Then, the value of radius, interval and R will be used to decide the corresponding coordinates value. The value will be ranged from $interval - radius$ to $interval + radius$. 4) Finally, repeat the previous three steps until we have generated enough data points.

In all the experiments that execute the Feel-the-Way algorithms, we use five different clustering seeds and report the average of their measured performance in our performance figures.

Table 3. The BigRed II Supercomputer.

Nodes	1,020
Memory per node	64 GB
Processors per node	2
Cores per processor	16
Max cores per job	4,096
Processor	AMD Opteron 6380 2.5GHz
Interconnect	Cray Gemini
MPI	Cray-MPICH 7.2.5

Table 4. Description of the real-world datasets.

	MNIST	CIFAR-10	ENRON	PLACES-2
#clusters (K)	10	10	10	10
#Data points	60,000	60,000	38,400	1,024,000
#Coordinates	784	3,072	28,102	2,048
Dataset size	110MB	626MB	24.7MB	3.25GB

Table 5. Description of the synthetic datasets.

	SYN	SYN-Large
#clusters (K)	10	20
#Data points	6,400	409,600
#Coordinates	1,024	2,048
Dataset size	50.48MB	7.06GB

7.1 Evaluation of Different Sampling Methods

Since the effectiveness of a sampling approach determines the performance of the sampling-based Feel-the-Way algorithm, 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 following subsection, we first use the *Reassign-hist* method as an example to illustrate how a sampling ratio can affect the convergence rate and execution time. Next, we use a fixed small ratio of 1% to compare five sampling methods.

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

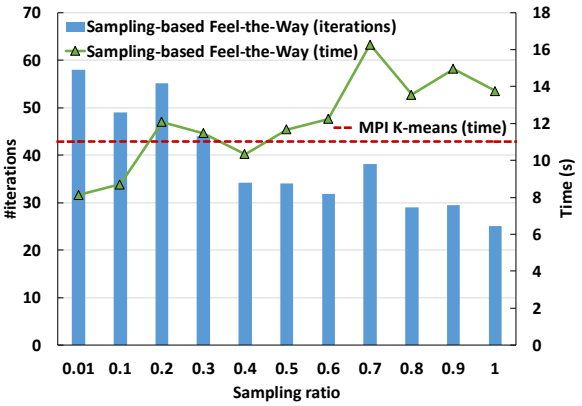
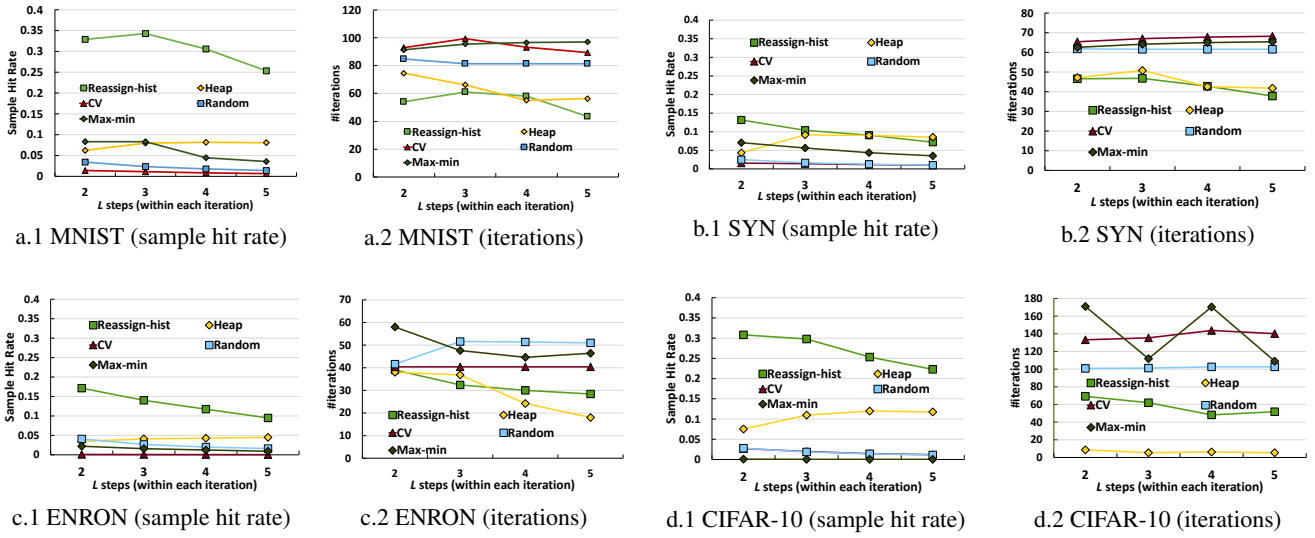


Figure 3. Effect of the sampling ratio with the MNIST dataset using the *Reassign-hist* sampling method. MPI k-means needs 89 iterations.

Table 6. SSE costs corresponding to the sampling-ratio experiments that are shown in Figure 3.

dataset: MNIST			
k-means	1.5302×10^{11}		
Sampling-based method (Reassign-hist)	0.01 (Sample ratio)	0.1 (Sample ratio)	
	1.5358×10^{11}	1.5362×10^{11}	
	0.2	0.3	0.4
	1.5340×10^{11}	1.5338×10^{11}	1.5337×10^{11}
	0.5	0.6	0.7
	1.5336×10^{11}	1.5335×10^{11}	1.5330×10^{11}
	0.8	0.9	1.0
	1.5329×10^{11}	1.5329×10^{11}	1.5328×10^{11}

Figure 3 shows that both the number of iterations and the execution time change as the sampling ratio increases from 1% to 100% for the MNIST dataset. Here we use the *Reassign-hist* sampling method 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 58 iterations. The number of iterations then drops from 58, 49 to 29 as the ratio rises from 1%, 10% to 80%. Eventually, when



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)

Figure 4. Comparing five sampling methods with four different datasets: a) MNIST, b) SYN, c) ENRON, d) CIFAR-10.

Table 7. Costs of different sampling methods for MNIST.

Sampling methods	L = 2	L = 3	L = 4	L = 5
Reassign	1.5334e11	1.5337e11	1.5358e11	1.5376e11
Heap	1.5328e11	1.533e11	1.5333e11	1.5336e11
CV	1.5327e11	1.5327e11	1.5327e11	1.5327e11
Random	1.5327e11	1.5327e11	1.5327e11	1.5327e11
Max-min	1.5327e11	1.5327e11	1.5327e11	1.5327e11
k-means	1.5302e11			

Table 8. Costs of different sampling methods for SYN.

Sampling methods	L = 2	L = 3	L = 4	L = 5
Reassign	6.3821e9	6.3918e9	6.4059e9	6.4215e9
Heap	6.4070e9	6.3971e9	6.4389e9	6.4569e9
CV	6.3470e9	6.3469e9	6.3469e9	6.3469e9
Random	6.3449e9	6.3449e9	6.3449e9	6.3449e9
Max-min	6.3446e9	6.3447e9	6.3447e9	6.3447e9
k-means	6.2645e9			

Table 9. Costs of different sampling methods for ENRON.

Sampling methods	L = 2	L = 3	L = 4	L = 5
Reassign	2.3041e7	2.3039e7	2.3041e7	2.3040e7
Heap	2.3156e7	2.3184e7	2.3432e7	2.3470e7
CV	2.2872e7	2.2872e7	2.2872e7	2.2872e7
Random	2.2974e7	2.2972e7	2.2972e7	2.2972e7
Max-min	2.2773e7	2.2880e7	2.2963e7	2.2958e7
k-means	2.2882e7			

Table 10. Costs of different sampling methods for CIFAR10.

Sampling methods	L = 2	L = 3	L = 4	L = 5
Reassign	4.740e11	4.741e11	4.744e11	4.745e11
Heap	4.852e11	5.018e11	5.093e11	5.181e11
CV	4.739e11	4.739e11	4.739e11	4.740e11
Random	4.739e11	4.739e11	4.739e11	4.739e11
Max-min	4.738e11	4.739e11	4.738e11	4.739e11
k-means	4.7355e11			

the sampling ratio is equal to 100%, we see a significant reduction in the number of iterations. This experiment tells us that given a specific sampling method, reducing the sampling ratio may not necessarily increase the number of iterations significantly (e.g., 20%, 1% have nearly the same number of iterations).

Regarding execution time (see the y-axis on the right), it increases gradually as the sampling ratio increases. For instance, the execution time jumps from 8 seconds to 16 seconds when the ratio rises from 1% to 70%. This is because a larger sampling ratio implies a larger time complexity for each local step l ($l = 2, 3, 4$). Theorem 1 proves the relationship between time and the number of iterations given a specific L and sampling ratio.

To provide a complete picture, Table 6 also provides the SSE costs associated with an arrange of sampling ratios used by Figure 3. We can see that the SSE costs of using different sampling ratios are almost the same when the *Reassign-hist* sampling method is used.

Reason to use a small sampling ratio: We choose to use a small sampling ratio of 1% for the following reasons. 1) It will lead to a small overhead (around 1%) to compute the local steps $l \in \{2 \dots L\}$. 2) In combination with the first full step $l = 1$, the sampling-based Feel-the-Way algorithm can achieve a less number of iterations than k-means (e.g., 58 iterations on average versus MPI k-means' 89 iterations, as shown in Figure 3).

Comparing the five sampling methods: We evaluate which sampling method provides the best performance based on two metrics: accuracy and the number of iterations. 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. 2) The number of iterations. This metric measures how many iterations are needed by the Feel-the-Way clustering algorithm to converge.

Figure 4 shows the experimental results with four datasets: a) MNIST, b) SYN, c) ENRON, and d) CIFAR-10. For each dataset, we measure the *Sampling Hit Rate* and *Number of Iterations* for five sampling methods, respectively. Note that *Sampling Hit Rate* and *Number of Iterations* are displayed

separately in two subfigures such as a.1 and a.2 for MNIST, b.1 and b.2 for SYN, and so on.

As shown in Figure 4 a.1, the Reassign-hist method has the highest sampling hit rate (around 33% when $L = 2$ and 3). Then its hit rate drops to 30% and 25% when $L = 4$ and 5. The second best sampling methods are the Heap and Max-min methods, which have a hit rate of 8%. The last two methods (Random and CV) achieve the lowest sampling hit rates that are always less than 5%.

With respect to the convergence rate, Figure 4 a.2 shows that the *Reassign-hist* method converges the fastest, taking around 50 iterations. By contrast, the Random sampling method takes 80 iterations, and the CV, Max-min sampling methods take around 90–100 iterations. The Heap sampling method converges slow when $L = 2$, then 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. Also, the SSE costs corresponding to Figure 4 a.1 and a.2's experiments are shown in Table 7, whose costs are close to each other.

Figure 4 b.1 shows the sampling hit rates of different methods for the SYN dataset. When $L > 2$, the Heap method is as good as the Reassign-hist method. The CV and Random methods have the lowest sampling hit rate. Correspondingly, Figure 4 b.2 shows that the Reassign-hist method and Heap method have the smallest number of iterations. Costs of all experiments are almost the same, as presented in Table 8.

Figure 4 c.1 and c.2 show the experimental results for the ENRON dataset. The Reassign-hist method has a significantly higher hit rate than the other sampling methods. Please note that when $L = 5$, the Heap method takes fewer iterations to converge than the Reassign-hist, but its SSE cost is 2% larger than the Reassign-hist method as shown in Table 9.

Figure 4 d.1 and d.2 show the experimental results for the CIFAR-10 dataset. Again, The Reassign-hist 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 Reassign-hist method by up to 9% as shown in Table 10. This is the reason that the Heap method takes fewer iterations to reach a less optimal solution.

In general, the Reassign-hist 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 will use the Reassign-hist method to show the performance of the Feel-the-Way algorithm.

7.2 Sequential performance of the full-step and sampling-based Feel-the-Way

In the second set of experiments, we compare the performance of MPI k-means, full-step Feel-the-Way, and sampling-based Feel-the-Way clustering algorithms. We use three real-world datasets (i.e., MNIST, CIFAR-10 and ENRON) and one synthetic dataset to compare the algorithms.

Figure 5 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 MPI k-means. The average costs of all experiments are presented in Table 11.

Table 11. SSE costs for the experiments that are shown in Figure 5. The first sub-row is used for the sampling-based algorithm. The second sub-row is for the full-step algorithm. The third sub-row is for the MPI k-means library.

Dataset	L = 2	L = 3	L = 4	L = 5
a) MNIST	1.5334e11	1.5337e11	1.5358e11	1.5376e11
	1.5327e11	1.5328e11	1.5328e11	1.5329e11
	1.5302e11			
b) SYN	6.3443e9	6.3443e9	6.3451e9	6.3464e9
	6.3821e9	6.3918e9	6.4059e9	6.4215e9
	6.2645e9			
c) ENRON	2.2944e7	2.2950e7	2.2950e7	2.2952e7
	2.2897e7	2.2828e7	2.2793e7	2.2858e7
	2.2882e7			
d) CIFAR10	4.7309e11	4.7348e11	4.7352e11	4.7353e11
	4.7296e11	4.7300e11	4.7301e11	4.7304e11
	4.7331e11			

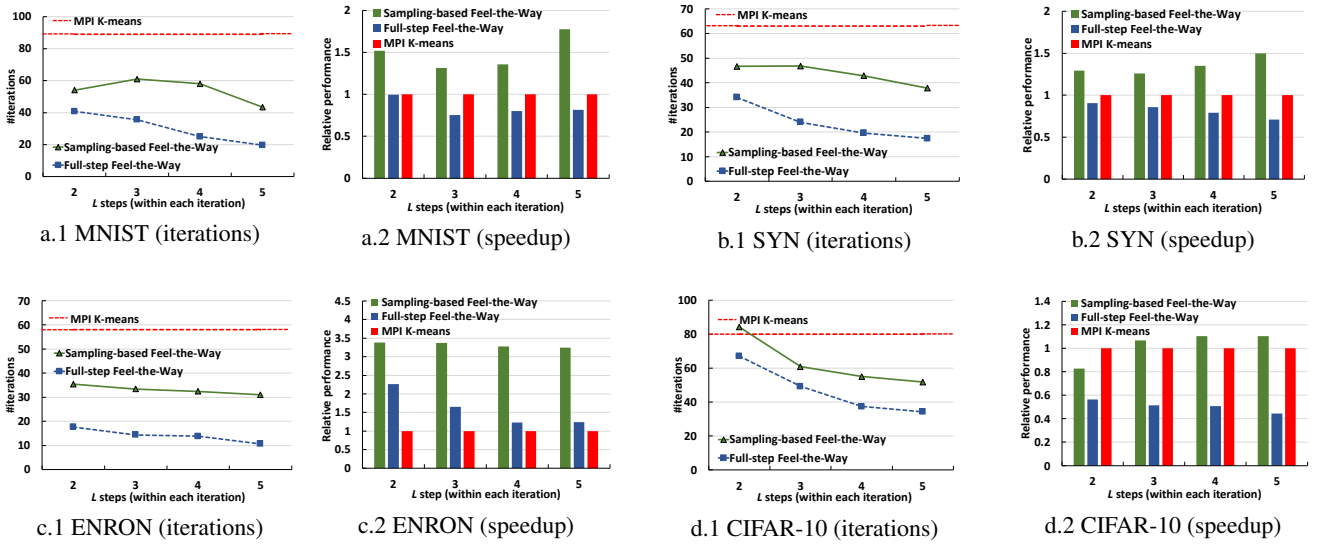
Figure 5 a.1 and a.2 show the performance with MNIST. In Subfigure a.1, the full-step Feel-the-Way algorithm has the least number of iterations because it is able to utilize its L number of local steps to improve the cost. The sampling-based Feel-the-Way algorithm has the second least number of iterations due to using a small portion (i.e., 1%) of sampled points. For instance, when L is equal to 5, sampling-based and full-step Feel-the-Way take an average number of 43.4 and 19.6 iterations to converge. By contrast, the MPI k-means library takes 89 iterations to converge using its default seeds.

In Figure 5 a.2, we show the speedup of the Feel-the-Way algorithms relative to the MPI k-means library. *Speedup* is computed by the division of two algorithms' execution time. The higher the number, the better the performance is. Thanks to the small sampling overhead of the sampling-based algorithm, the sampling based Feel-the-Way is faster than the full-step Feel-the-Way by up to 2.2 times although it requires more iterations to converge (as shown in subfigure a.1). Also, it is faster than the MPI k-means library by up to 1.8 times.

Figure 5 b shows the performance comparison with the SYN dataset. In the figure, the sampling-based Feel-the-Way algorithm has the second least number of iterations, and the best speedup, which outperforms the MPI k-means library by 1.5 times.

Similarly, Figure 5 c shows the performance with the ENRON dataset. The full-step Feel-the-Way algorithm is faster than the MPI k-means by up to 2.26 times, and the sampling-based Feel-the-Way algorithm is faster than the MPI k-means by 3.28 times.

In Figure 5 d that corresponds to the experiments with the CIFAR-10 dataset, the sampling-based Feel-the-Way algorithm attains speedups between 1.06 and 1.1 over the MPI k-means library.



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)

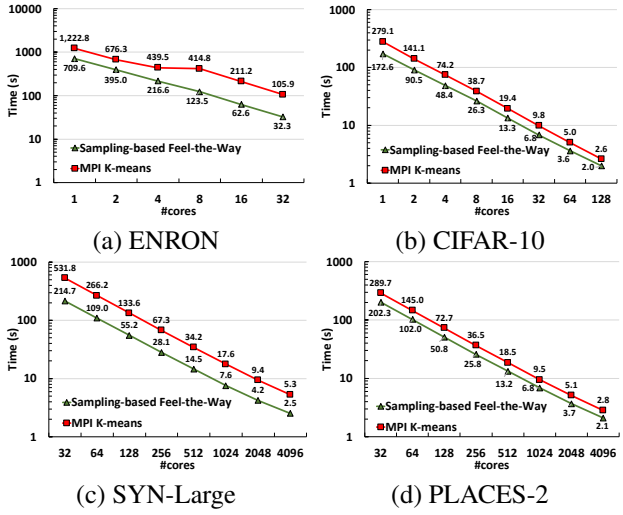
Figure 5. Effect of L steps on 3 algorithms (Sampling-based, Full-step, MPI k-means) with 4 datasets

Table 12. Parameter setting for the scalability experiments shown in Figure 6.

dataset	block size	sampling ratio	L steps
ENRON	1,200	0.01	4
CIFAR-10	468	0.01	3
SYN-Large	100	0.01	2
PLACES-2	250	0.01	2

Table 13. SSE costs for the strong scalability experiments shown in Figure 6.

dataset	k-means	Sampling-based Feel-the-Way
ENRON	2.2882×10^7	2.2951×10^7
CIFAR-10	4.7331×10^{11}	4.7355×10^{11}
SYN-Large	7.5308×10^{11}	7.6992×10^{11}
PLACES-2	5.4261×10^{12}	5.4226×10^{12}



(a) ENRON (b) CIFAR-10

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

Figure 6. Strong scalability experiments. The y-axis is shown in the logarithmic scale.

7.3 *Performance of the parallel sampling based Feel-the-Way algorithm*

To evaluate the performance of our parallel Feel-the-Way implementation, we perform large scale experiments on Big Red II using thousands of CPUs. In the parallel experiments, we take four datasets as input: ENRON, CIFAR-10, SYN-Large, and PLACES-2. We also use different block sizes and L steps, and set sample ratios to 1% for different datasets, as reported in Table 12.

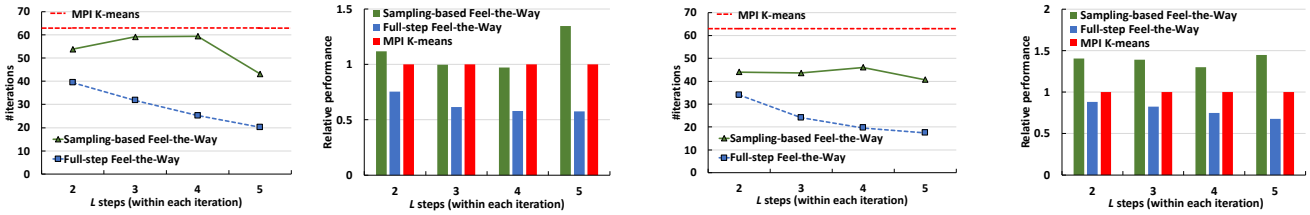
Figure 6 shows the results of the strong scalability experiments, where we increase the number of CPU cores given a fixed input size. In addition to displaying execution time, we also use Table 13 to show the SSE costs that are achieved by our Feel-the-Way algorithm and the MPI k-means library. From Table 13, we can see that the cost difference between MPI k-means and our algorithm is always small.

Figure 6.a displays the execution time of the MPI k-means library and our sampling-based Feel-the-Way algorithm with the ENRON dataset. As the number of CPU cores increases from 1 to 32, the parallel Feel-the-Way reduces the execution time from 709.6 to 32.3 seconds while the MPI k-means reduces from 1,222.8 to 105.9 seconds. We also notice that as the number of CPU cores increases from four to eight, the MPI k-means library does not scale very well. Our performance analysis found that the time spent on each iteration is almost identical when using four cores and eight cores. Also, this inefficiency only happens to the ENRON dataset from four to eight CPU cores when using the MPI k-means library.

Figure 6.b displays the execution time when taking the CIFAR-10 dataset. On 128 CPU cores, our parallel implementation is faster than the MPI k-means by 32%.

In Figure 6.c, we use 4,096 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 214.7 to 2.5 seconds using 4,096 cores, which is faster than the MPI k-means by 110%.

As for the PLACES-2 dataset, Figure 6.d shows that the parallel sampling-based Feel-the-Way implementation is able to reduce the execution time from 202.3 to 2.1 seconds when using from 32 to 4,096 CPU cores. Again, the parallel



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

Figure 7. Effect of L steps on three algorithms (Sampling-based, Full-step, MPI k-means), which are tested with two datasets (MNIST, SYN) using single-precision floating-point numbers.

Table 14. SSE costs for the single-precision floating-point number experiments as shown in Figure 7. The first sub-row is used for the sampling-based Feel-the-Way algorithm. The second sub-row is for the full-step Feel-the-Way algorithm. The third sub-row is for the MPI k-means library in single precision.

Dataset	L = 2	L = 3	L = 4	L = 5
a) MNIST	1.5334e11	1.5337e11	1.5358e11	1.5376e11
	1.5327e11	1.5328e11	1.5328e11	1.5329e11
	1.5302e11			
b) SYN	6.3821e9	6.3897e9	6.3996e9	6.406e9
	6.3432e9	6.3441e9	6.3448e9	6.3457e9
	6.2645e9			

Feel-the-Way program outperforms the MPI k-means library by 33% on 4,096 cores.

7.4 *Performance comparison using single-precision floating-point numbers*

We also use the MNIST and SYN dataset to evaluate the performance of our algorithm and implementation using single-precision floating-point numbers. The results of the full-step and sampling-based Feel-the-Way algorithms are averaged from five different sets of seeds.

Similar to the double-precision result shown in Figure 5 a.1, the full-step and sampling-based Feel-the-Way in single-precision also take fewer number of iterations to converge than the MPI k-means. In addition, the sampling-based Feel-the-Way algorithm is 34% faster than MPI k-means when the number of local steps $L = 5$, which is shown in Figure 7 a.1.

As to the dataset SYN in Figure 7 b.1 and b.2, the sampling-based Feel-the-Way achieves the speedups between 1.29 and 1.44. To show the solution quality for the above set of experiments, we also summarize their SSE costs in Table 14. We can see that the Feel-the-Way algorithms and k-means algorithm have comparable SSE costs. Also, note that although the single-precision version of MPI k-means converges faster than the double-precision version, its accuracy may become worse. For example, the single-precision output on the MNIST dataset is worse than the double-precision with a higher SSE cost (1.530205e11 vs double-precision's 1.530179e11).

8 Conclusion

In this paper, we presented a Feel-the-Way method to reduce the synchronization cost of large-scale machine learning algorithms without sacrificing the accuracy and convergence rate. We applied this method to design and implement a fast clustering algorithm for large-scale HPC systems. We first designed a full-step Feel-the-Way algorithm that, compared to the traditional k-means algorithm, takes significantly less number of iterations by applying local optimization to each data block, meanwhile providing almost the same SSE cost as the MPI k-means library. Next, we introduced different sampling methods to optimize the execution time of the full-step algorithm and designed a new algorithm called sampling-based Feel-the-Way. By sampling certain "useful" data points, the new sampling-based Feel-the-Way algorithm has a much better performance than the full-step algorithm. Five sampling methods have been designed and tested, among which the reassignment-history-aware sampling method achieves the best convergence rate.

Our future work will focus on the application of the parallel sampling-based Feel-the-Way method to design and implement other advanced machine learning algorithms (such as deep reinforcement learning and deep generative models) on extreme-scale HPC systems.

Acknowledgements

This material is based upon research supported by the Purdue Research Foundation, by the NSF Grant# 1835817, and by the U.S. Department of Energy (DOE), Office of Science, Advanced Scientific Computing Research Program.

References

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

Albanese D, Visintainer R, Merler S, Riccadonna S, Jurman G and Furlanello C (2012) mlpy: Machine learning python. *arXiv preprint arXiv:1202.6548* .

Alsabti K, Ranka S and Singh V (1997) An efficient k-means clustering algorithm .

Bru R, Elsner L and Neumann M (1988) Models of parallel chaotic iteration methods. *Linear Algebra and its Applications* 103: 175–192.

Chen X and Cai D (2011) Large scale spectral clustering with landmark-based representation. In: *AAAI*, volume 5. p. 14.

Chitta R, Jin R, Havens TC and Jain AK (2011) Approximate kernel k-means: Solution to large scale kernel clustering. In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 895–903.

Demmel J and Dinh G (2018) Communication-optimal convolutional neural nets. *arXiv preprint arXiv:1802.06905* .

Demmel JW, Grigori L, Hoemmen MF and Langou J (2008) Communication-optimal parallel and sequential QR and LU factorizations. LAPACK Working Note 204, UTK.

Demšar J, Curk T, Erjavec A, Gorup Č, Hočevar T, Milutinovič M, Možina M, Polajnar M, Toplak M, Starič A et al. (2013) Orange: data mining toolbox in python. *The Journal of Machine Learning Research* 14(1): 2349–2353.

Dhillon IS and Modha DS (2002) A data-clustering algorithm on distributed memory multiprocessors. In: *Large-Scale Parallel Data Mining*. Springer, pp. 245–260.

Di Fatta G, Blasa F, Cafiero S and Fortino G (2013) Fault tolerant decentralised k-means clustering for asynchronous large-scale networks. *Journal of Parallel and Distributed Computing* 73(3): 317–329.

Elankavi R, Kalaiprasath R and Udayakumar DR (2017) A fast clustering algorithm for high-dimensional data. *International Journal Of Civil Engineering And Technology (IJCIET)* 8(5): 1220–1227.

Elkan C (2003) Using the triangle inequality to accelerate k-means. In: *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. pp. 147–153.

Frommer A and Szyld DB (2000) On asynchronous iterations. *Journal of computational and applied mathematics* 123(1): 201–216.

Gittens A, Devarakonda A, Racah E, Ringenburg M, Gerhardt L, Kottalam J, Liu J, Maschhoff K, Canon S, Chhugani J et al. (2016) 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, pp. 204–213.

Goodfellow IJ, Warde-Farley D, Lamblin P, Dumoulin V, Mirza M, Pascanu R, Bergstra J, Bastien F and Bengio Y (2013) Pylearn2: a machine learning research library. *arXiv preprint arXiv:1308.4214* .

Hamerly G (2010) Making k-means even faster. In: *Proceedings of the 2010 SIAM international conference on data mining*. SIAM, pp. 130–140.

Har-Peled S and Mazumdar S (2004) On coresets for k-means and k-median clustering. In: *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*. ACM, pp. 291–300.

Hartigan JA and Hartigan J (1975) *Clustering algorithms*, volume 209. Wiley New York.

Havens TC, Bezdek JC, Leckie C, Hall LO and Palaniswami M (2012) Fuzzy c-means algorithms for very large data. *IEEE Transactions on Fuzzy Systems* 20(6): 1130–1146.

Ho Q, Cipar J, Cui H, Lee S, Kim JK, Gibbons PB, Gibson GA, Ganger G and Xing EP (2013) More effective distributed ML via a stale synchronous parallel parameter server. In: *Advances in neural information processing systems*. pp. 1223–1231.

Hoare CA (1961) Algorithm 65: find. *Communications of the ACM* 4(7): 321–322.

Hsieh CJ, Si S and Dhillon I (2014) A divide-and-conquer solver for kernel support vector machines. In: *International Conference on Machine Learning*. pp. 566–574.

Jain AK, Murty MN and Flynn PJ (1999) Data clustering: a review. *ACM computing surveys (CSUR)* 31(3): 264–323.

Jin S, Cui Y and Yu C (2016) A new parallelization method for k-means. *arXiv preprint arXiv:1608.06347* .

Kantabutra S and Couch AL (2000) Parallel k-means clustering algorithm on nows. *NECTEC Technical journal* 1(6): 243–247.

Kanungo T, Mount DM, Netanyahu NS, Piatko CD, Silverman R and Wu AY (2002) An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis & Machine Intelligence* (7): 881–892.

Kerdprasop K and Kerdprasop N (2010) Parallelization of k-means clustering on multi-core processors. In: *Proceedings of the 10th WSEAS international conference on Applied computer science, ACS*, volume 10. pp. 472–477.

Klimt B and Yang Y (2004) The enron corpus: A new dataset for email classification research. In: *European Conference on Machine Learning*. Springer, pp. 217–226.

Krizhevsky A and Hinton G (2009) Learning multiple layers of features from tiny images .

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

LeCun Y, Bottou L, Bengio Y and Haffner P (1998) Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11): 2278–2324.

Li M, Andersen DG, Park JW, Smola AJ, Ahmed A, Josifovski V, Long J, Shekita EJ and Su BY (2014) Scaling distributed machine learning with the parameter server. In: *OSDI*, volume 14. pp. 583–598.

Liao W (2017) Parallel k-means data clustering for large data sets. http://www.ece.northwestern.edu/~wkliao/Kmeans/index.html.

Low Y, Bickson D, Gonzalez J, Guestrin C, Kyrola A and Hellerstein JM (2012) Distributed GraphLab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment* 5(8): 716–727.

MLlib (2017) http://spark.apache.org/mllib/.

Moore AW (2000) The anchors hierarchy: Using the triangle inequality to survive high dimensional data. In: *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., pp. 397–405.

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V et al. (2011) Scikit-learn: Machine learning in python. *Journal of machine learning research* 12(Oct): 2825–2830.

Pelleg D and Moore A (1999) Accelerating exact k-means algorithms with geometric reasoning. In: *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 277–281.

Sculley D (2010) Web-scale k-means clustering. In: *Proceedings of the 19th international conference on World wide web*. ACM, pp. 1177–1178.

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

Song F, Ltaief H, Hadri B and Dongarra J (2010) 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)*. pp. 1–11.

Tallada MG (2016) Coarse grain parallelization of deep neural networks. In: *ACM SIGPLAN Notices*, volume 51. ACM, p. 1.

Amazon Machine Learning (2017) https://aws.amazon.com/aml/details/.

Apache Mahout (2017) https://mahout.apache.org/.

Wang L, Leckie C, Kotagiri R and Bezdek J (2011) Approximate pairwise clustering for large data sets via sampling plus extension. *Pattern Recognition* 44(2): 222–235.

You Y, Demmel J, Czechowski K, Song L and Vuduc R (2015) Ca-svm: Communication-avoiding support vector machines on distributed systems. In: *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*. IEEE, pp. 847–859.

Zhao W, Ma H and He Q (2009) Parallel k-means clustering based on mapreduce. In: *IEEE International Conference on Cloud Computing*. Springer, pp. 674–679.

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

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