An Accurate and Efficient Large-scale Regression Method through Best Friend Clustering

Kun Li, Liang Yuan, *Member, IEEE,* Yunquan Zhang, *Senior Member, IEEE,* and Gongwei Chen

Abstract—As the data size in Machine Learning fields grows exponentially, it is inevitable to accelerate the computation by utilizing the ever-growing large number of available cores provided by high-performance computing hardware. However, existing parallel methods for clustering or regression often suffer from problems of low accuracy, slow convergence, and complex hyperparameter-tuning. Furthermore, the parallel efficiency is usually difficult to improve while striking a balance between preserving model properties and partitioning computing workloads on distributed systems. In this paper, we propose a novel and simple data structure capturing the most important information among data samples. It has several advantageous properties supporting a hierarchical clustering strategy that contains well-defined metrics for determining optimal hierarchy, balanced partition for maintaining the clustering property, and efficient parallelization for accelerating computation phases. Then we combine the clustering with regression techniques as a parallel library and utilize a hybrid structure of data and model parallelism to make predictions. Experiments illustrate that our library obtains remarkable performance on convergence, accuracy, and scalability.

Index Terms—Distributed Machine Learning, Scalable Algorithm, Large-scale Clustering, Parallel Regression

1 Introduction

Machine Learning (ML) has become one of the crucial mainstays of information technology over past decades, albeit commonly hidden, part of modern life pervasively. As technologies based on machine learning like artificial intelligence (AI) rise prominently, the data size that researchers have to study with is also growing exponentially [12], [31]. This makes training time for models range from hours to weeks, which poses intense pressures across computation, networking, and storage. Thus, accelerating model training is an important research challenge within the machine learning field.

Today, High Performance Computing (HPC) and parallel techniques catalyze the modern revolution in machine learning. More and more companies are turning to HPC as the solution for ML-based productivity and AI-enabled innovation, such as Google Cloud TPU, Amazon AWS AIHPC, and Microsoft Azure. Within the field of machine learning, clustering and regression are two fundamental and crucial techniques, which are distinguished as the representative unsupervised and supervised learning methods [14], [28].

Clustering methods divide data into different groups by data attributes. Various types of clustering are proposed for providing a range of different uses [25], [30], [37], [40], [46]. Basically, they can be distinguished as hierarchical (nested) versus partitional (unnested) methods, where Agglomerative Hierarchical Clustering (AHC) and K-Means are two fundamental methods that are widely used [43], [53]. AHC and its variants make a set of nested clusters organized as a hierarchical tree [36]. K-Means is simple and efficient on a variety of problems [32], [33]. Recently, Graph Clustering (GC) such as Minimum Spanning Tree clustering (MSTC) also stands prominently for detecting clusters with irregular boundaries by graph theory. It uses Prim's algorithm or Kruskal's algorithm to construct a minimum spanning tree (MST) first, sorts edges to remove inconsistent ones for connected components, and repeats until a threshold loops [21], [27].

Regression involves intensive computations in the model training phase inherently. A standard distributed method for regression is to map the computation on p-processor system evenly and then perform a global reduction regularly [39]. Generally, these distributed regression methods work poorly in scaling cases due to busy synchronization sweeps. Well-known approaches use Divide-and-Conquer (DC) algorithm to optimize the parallel process. The basic idea of Divide-and-Conquer Regression (DCR) is to divide the data into p similar parts, generate p similar training models, and average p models for a final solution. The memory and computation overhead is reduced by DCR, such as DCKRR on Kernel Ridge Regression (KRR) and DCSVM on Support Vector Regression (SVR) [24], [51]. However, the accuracy is not guaranteed in various cases [49].

More recent work is the parallel optimization for KRR: Balanced KRR v2 (BKRR2), which is the latest version by researchers [48]. BKRR utilizes K-Means to partition n samples to p clusters, where p is the number of processes. Each cluster contains n/p dispatched samples, and then p models are generated correspondingly. To guarantee the load-balance, no additional sample will be added to it if the clustering center contains n/p dispatched samples by K-Means. Regression is then performed on each distributed

- *Kun Li, Liang Yuan, and Yunquan Zhang are with the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences (CAS), Beijing 100190, China. E-mail: likungw@gmail.com, yuanliang@ict.ac.cn, zyq@ict.ac.cn.*
- *Gongwei Chen is with the Key Laboratory of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences (CAS), Beijing 100190, China. E-mail: gongwei.chen@vipl.ict.ac.cn.*
- *Kun Li and Gongwei Chen are also with the School of Computer Science and Technology, University of Chinese Academy of Sciences (UCAS), Beijing 100190, China.*

(Corresponding author: Liang Yuan.)

models, and results are gathered for an average. Based on the BKRR, BKRR2 is proposed to improve the accuracy by training models independently without a global reduction. Although BKRR2 proves that it could achieve higher accuracy and efficiency than the current fastest method [48], [49], various problems are still not clearly addressed.

First, the accuracy and the convergence for K-Means are not always satisfactory on large dataset. It needs at least thousands of times for convergence in the experiments of BKRR2. Since the clustering center is changing dynamically, the volume of transferred data is also massive in each iteration. BKRR2 also avoids confronting this problem directly and implements K-Means only with a process. Second, since K is a hyperparameter to assign #clusters in K-Means, its value is simply set to #processes in BKRR2 and that is not robust. The clustering results deviate from the real distribution drastically as #processes change, which leads to a poor accuracy in many cases. Even worse, the training time increases with more generated clusters assigned by increasing processes, which is essentially not adequate for large-scale training. Furthermore, BKRR2 achieves load-balance at the cost of accuracy. In the process of clustering, samples are traversed and dispatched to their nearest cluster center. However, a more relevant sample to some "full" cluster cannot be added to it for its late traversal order by the design of BKRR2.

In this paper, we design an efficient parallel regression library to address the pending problems in existing methods. First, we propose a new graph structure called Best Friend Graph to capture the most important information among the data samples. Based on the proposed structure, a novel clustering method, Best Friend Clustering (BFC), is presented. It reduces computation over AHC approaches, improves accuracy to K-Means methods, and enhances scalability than MST-based algorithms. Then we dig out the inherent properties of the graph structure and design a well-defined metric to determine the optimal aggregation.

Next, a balanced partition algorithm for data parallelism is proposed based on the idea of backtracking, which could reflect more characters on the real distribution. Instead of deleting edges by a non-increasing order serially in MST-based work [7], [21], we gather samples in the same group by swapping their pointers simply. The larger clusters are then split by backtracking and smaller clusters are merged from the optimal hierarchy to obtain total n/p samples in each process. Thus, the storage for edges is released, and the procedure could be parallelized easily.

Furthermore, Best Friend Clustering is applied to parallel regression. Although the above-proposed methods can be used separately, we combine them with multiple regression techniques as a complete parallel regression library. Each process generates single or multiple training models independently according to the clustering results. For each test sample, it selects the best one from these models in the prediction phase. Thus, model parallelism is achieved efficiently in each process. To be as general as possible, our library has no specific limits on the dimensionality of the dataset and the format of the distance measure.

At last, we present an experimental study on our Best Friend Clustering with three different clustering methods. Then the accuracy and scalability are analyzed for our regression library with state-of-the-art work. The encouraging performance demonstrates the distinct superiority of the proposed methods on distributed systems.

The main contributions in this paper include:

- A novel Best Friend Clustering method is proposed for large-scale clustering, which is accurate, fast, and parameter-free.
- Theoretical properties of the proposed clustering method are studied thoroughly, and we design a strategy to decide the optimal clustering aggregation through a well-defined metric based on analytical properties.
- We optimize the clustering on distributed systems, apply it to parallel regression, and propose a partition algorithm for load-balancing and spatial locality.
- By utilizing a hybrid structure of data and model parallelism, we combine the proposed methods with regression techniques as a parallel library. It shows a remarkable performance on convergence, accuracy, and scalability.

2 Background

2.1 Regression Techniques

Regression is a supervised machine learning technique that is utilized to investigate the relationship between one or more predictors and response variables for a best-fit curve. The obtained curve can be then employed for making predictions on new data points. Basically, there are various algorithms that are utilized to build a regression model, and in our paper, three extensively-used techniques are molded out and implemented in our library.

2.1.1 Linear regression

Linear regression is a widely-used method to find the linear relationship between the dependent variable and one or more independent variables by employing a straight line. Given a d-dimensional training sample $\mathbf{x}_i$ and the corresponding measured regressand y_i, the objective in training phase is to find a $\mathbf{w}$ such that $y_i \approx \mathbf{w}^T \cdot \mathbf{x}_i$. This can be formulated as a least squares problem in Equation 1 to find the optimal line by minimizing the sum of the residuals. Then for a test sample x^*, we can utilize the training model to predict the regressand by $y^* = \mathbf{w}^T \cdot \mathbf{x}^*$, and evaluate the accuracy with Mean Squared Error (MSE) method.

$$\mathbf{w}_{opt} = \arg\min \frac{1}{2}\sum\nolimits_i^n (y_i - \mathbf{w}^T \mathbf{x}_i)^2 \tag{1}$$

2.1.2 Kernel Ridge regression

To avoid overfitting and solve some ill-posed problems, L2 regularization with a positive parameter λ is used in Equation 2, which is called ridge regression.

$$\mathbf{w}_{opt} = \arg\min \frac{1}{2}\sum\nolimits_i^n (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \frac{1}{2}\lambda ||\mathbf{w}||^2 \tag{2}$$

Moreover, kernel method is widely utilized to map samples to a high dimensional space using a nonlinear mapping. Thus the Kernel Ridge Regression (KRR) is presented by combining ridge regression with kernel method, and it learns model in high dimensional space for a better prediction

accuracy. The solution vector $\boldsymbol{\alpha}$ can be written in closed form in Equation 3,

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda \mathbf{I}_N)^{-1}\mathbf{y} \tag{3}$$

where $\mathbf{K}$ is a n-by-n kernel matrix constructed by $\mathbf{K}_{i,j} = \phi(x_i, x_j)$, $\mathbf{y}$ is the corresponding n-by-1 regressand vector. Then $\boldsymbol{\alpha}$ is employed to predict regressands in prediction phase as Equation 4.

$$y^* = \sum_{i=1}^{N} \alpha_i k(x^*, x_i) \tag{4}$$

2.1.3 Support Vector Regression

Support Vector Machine (SVM) can also be used as a regression method called Support Vector Regression (SVR), holding all the key features such as maximal margin that characterize the algorithm. Both KRR and SVR can learn a non-linear model by using kernel tricks, while they differ in the loss functions, i.e., ridge and epsilon-insensitive loss respectively. In the case of SVR, a margin of tolerance (epsilon) is set to the SVM, and we can tune it to gain the desired accuracy of our model. The solution is given in Equation 5 and constrained to Equation 6, where $\mathbf{w}$ is the magnitude of the normal vector to the surface that is being approximated.

$$\mathbf{w}_{opt} = \arg\min \frac{1}{2}||\mathbf{w}||^2 \tag{5}$$

$$|y_i - \mathbf{w}x_i| \leq \varepsilon \tag{6}$$

2.2 Clustering Methods

Many clustering methods are proposed in the literature to recognize the cluster of different characteristics. In this subsection, AHC, K-Means, and MSTC are introduced briefly.

2.2.1 Agglomerative Hierarchical Clustering

Agglomerative Hierarchical Clustering is an important category of clustering methods, which makes a set of nested clusters organized as a hierarchical tree. The Basic algorithm consists of following key steps: start with individual samples as clusters, merge two nearest clusters successively until one cluster obtained. Algorithm 1 describes AHC formally. Generally AHC algorithm can produce a better-quality clustering result. However, the slow convergence speed, extensive computation operations, and expensive storage requirements make scaling problematic on a larger dataset.

Algorithm 1 Agglomerative Hierarchical Clustering Algorithm

1: Let each sample be a cluster.
2: **repeat**
3: Merge the nearest two clusters as a new one.
4: Update the proximity matrix.
5: **until** Only one cluster remains.

2.2.2 K-Means

K-Means is one of the most prominent partitional approaches for clustering. The basic steps are illustrated in Algorithm 2. K-Means is simple and efficient on a variety of problems. However, non-globular clusters or clusters of different densities and sizes cannot be solved by K-Means, and the outliers in data can also affect results. Furthermore, K value is a hyperparameter specified in advance, which determines the quality of clustering significantly.

Algorithm 2 K-Means Algorithm

1: Choose K initial sample as centroids.
2: **repeat**
3: Assign each sample to its nearest centroid to form K clusters.
4: Recompute K centroids.
5: **until** Centroids remains unchange.

2.2.3 MST-based Clustering

The basic idea of MSTC is as follows. First, set a threshold cluster size k. Then construct MSTs using classic MST algorithms until k clusters obtained. At last, delete the maximum edge iteratively to obtain exactly k clusters. The algorithm can obtain a comparatively better result on clusters with irregular boundaries, while it is highly sequential and computationally intensive [7].

Algorithm 3 MST-based Clustering Algorithm

1: Compute and sort the pairwise distances.
2: **repeat**
3: Run MST algorithm to form clusters.
4: **until** k clusters.

3 Method

In this section, we first propose a simple, efficient, and accurate method for large-scale clustering. Section 3.1 and Section 3.2 provide the definition of Best Friend Graph and properties of this new method. They serve as the fundamental data structure of our method. Then a strategy to decide the optimal clustering aggregation through a well-defined metric is presented in Section 3.3. Next, we use a hybrid structure of data and model parallelism for parallel regression. Section 3.4 presents the detailed description of balanced partition for data parallelism. It achieves load-balance by utilizing merge and split operations on the hierarchical clustering structure. At last, the parallel regression method achieved with model parallelism is described in Section 3.5.

3.1 Best Friend Clustering

Clustering is intuitively inspired by the most fundamental social relation, friend circle, for detecting the potential groupings in the data. Existing clustering methods normally consider relationships between all pairs of samples. This leads to a slower convergence rate and a large overhead, which are not adequate for large-scale datasets on distributed systems. In reality, many of the relationships can be ignored.

Our clustering method arises from the observation that people often make a decision with their best friends. Therefore, we simplify the friend cycle by only considering the most important best friend relationship.

A *Best Friend Graph* $G(V, E)$ is defined on a dataset of n input samples for clustering. The vertex set V consists of these n data elements. Each vertex $i \in V$ is associated with a directed *best friend edge* (i, j) where the destination j is its best friend, i.e. i's nearest neighbor. Note that if there exist multiple nearest neighbors with the same distance to i, we simply choose the vertex with the smallest lexicographic order as the only best friend of i. Therefore, a Best Friend Graph $G(V, E)$ with $|V| = n$ vertices contains n directed best friend edges. Figure 1 illustrates a tractable example of the Best Friend Graph. We take 12 representative cities from Global Cities Index [45] as a study case. The geographical distances among them are calculated and 12 directed edges identifying the best friend relationships among them.

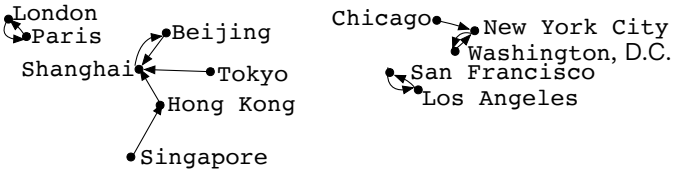

Fig. 1. Best Friend Graph with a case study of global cities.

Based on the Best Friend Graph, a cluster is defined as a group of connected vertices. Evidently, the city samples are clustered into four groups by related friends in the first step. The clusters distinguished by Best Friend Clustering are consistent with the geographic taxonomies. For example, Los Angeles and San Francisco are recognized as the Western US. Chicago, Washington DC, and New York City are then classified as the Eastern US.

Let $T_k(V_k, E_k)$ be a connected subgraph k in G and $v_k = |V_k|$. Specifically, each new cluster is represented by a centroid $\overline{x}_k$. The centroid $\overline{x}_k$ representing T_k is updated by averaging all vertices x_i that have been assigned to this cluster in Equation 7. To find it arithmetically, one computes the arithmetic mean of the x_i' coordinates separately for each dimension.

$$\overline{x}_k = \frac{1}{v_k} \sum_{x_i \in V_k} x_i \tag{7}$$

We can then build a new Best Friend Graph of all generated cluster centers in the previous step. In our example, the four clusters are further combined into two groups as illustrated in Figure 2. We recursively apply this approach to the new clusters until there is only one cluster containing all the samples. The whole process converges quickly in logarithmic time as a cluster contains at least two samples in each hierarchy. Furthermore, the computation and communication time are reduced progressively with hierarchies.

3.2 Properties

In this subsection, we analyze several crucial properties of Best Friend Graph, which are the basis to decide the optimal aggregation.

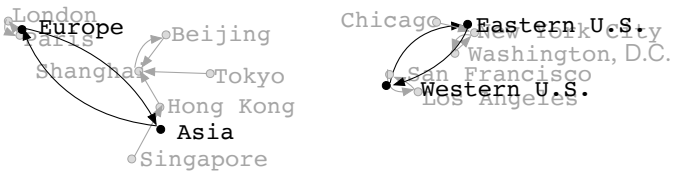

Fig. 2. Best Friend Clustering with a case study of global cities.

Lemma 3.1. There exists at least one directed cycle in a Best Friend Graph.

Proof. If we replace all directed edges with undirected edges, a Best Friend Graph becomes an undirected graph with n vertices and n edges. By induction, we can easily prove that it involves at least one undirected cycle. Transforming the edges on the cycle back to directed edges, we obtain a directed cycle. Otherwise, there must be one vertex i associated with two best friend edges $(i, *)$ on the cycle, a contradiction to the best friend graph definition.

Lemma 3.2. Each weakly connected component of a Best Friend Graph contains one and only one directed cycle.

Proof. Each weakly connected component $G'(V', E')$ of a Best Friend Graph $G(V, E)$ is still a Best Friend Graph according to the definition. With Lemma 3.1, G' contains at least one cycle. Thus we obtain $|E'| \geq |V'|$. We must have $|E'| = |V'|$, i.e. only one cycle in G', otherwise will get $|E| > |V| = n$, contradicting the Best Friend Graph definition.

Lemma 3.3. The edge weights on a directed path of a Best Friend Graph are non-increasing.

This is obvious from the definition. For example, for the directed path $i \to j \to k$, we have $\omega(i, j) \geq \omega(j, k)$, otherwise j's best friend should be i rather than k.

Lemma 3.4. The length of a directed cycle in a Best Friend Graph is two and the weights of edges on a cycle are identical.

Proof. Assume there is only one nearest neighbor to each vertex, i.e. the weight ω of a best friend edge (i, j) is strictly smaller than the distance between i and other vertex $\omega(i, j) < \omega(i, k), k \neq i, j$. If there exists a directed cycle involving more than two vertices, e.g. $i \to j \to k \to i$, we have $\omega(i, j) > \omega(j, k) > \omega(k, i) = \omega(i, k)$. Thus the best friend edge of i should be (i, k), a contradiction.

Otherwise, we have $\omega(i, j) \geq \omega(j, k) \geq \omega(k, i) = \omega(i, k)$. Since (i, j) is a best friend edge, we obtain $\omega(i, j) = \omega(i, k)$ and $\omega(i, j) = \omega(j, k) = \omega(k, i)$. According to the smallest lexicographic order rule in Best Friend Graph definition, we get an inconsistent lexicographic order $j \prec k \prec i \prec j$.

Corollary 3.1. Starting from any vertex and traversing along the Best Friend Graph will enter the cycle.

Definition 3.1. A best friend forest $F(V, E)$ of a Best Friend Graph $G(V, E')$ is defined by replacing all directed edges with undirected ones and removing one edge on each cycle.

Theorem 3.1. A connected component, a tree $T(V_T, E_T)$ in a best friend forest is a minimum spanning tree of the corresponding complete graph $G(V_T, E)$.

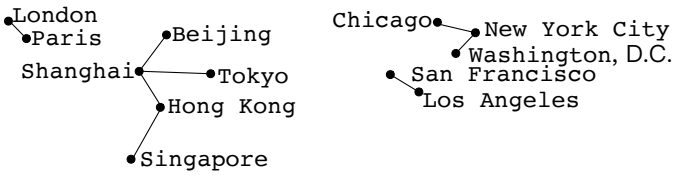

Fig. 3. Best friend forest with a case study of global cities.

Proof. Let $T^*(V_T, E_{T^*})$ be the MST of the complete graph $G(V_T, E)$. For an edge (i, j) in $E_{T^*} - E_T$, we obtain an undirected path in $T(V_T, E_T)$. According to Corollary 3.1, the corresponding directed path connecting i and j in the Best Friend Graph is one of the following three cases: $i \to \cdots \to j$, $i \leftarrow \cdots \leftarrow j$ and $i \to \cdots \to k \leftarrow \cdots \leftarrow j$. Since adding (i, j) it to $T(V_T, E_T)$ leads to a cycle, there must be an edge (u, v) in the path that does not belong to $T^*(V_T, E_{T^*})$. In any of the three path cases, we have $\omega(i, j) \geq \omega(u, v)$ according to Lemma 3.3. We must have $\omega(i, j) = \omega(u, v)$, otherwise we will get spanning tree with a smaller weight by replacing (i, j) with (u, v). This contradicts the assumption that $T^*(V_T, E_{T^*})$ is a MST. Repeat the process for all edges in (i, j) in $E_{T^*} - E_T$, we will obtain T with a same weight to T^*. Thus $T(V_T, E_T)$ is also a MST.

Figure 3 shows the corresponding best friend forest of the example in Figure 1. With Theorem 3.1, each connected component in Figure 3 is a minimum spanning tree, and the graph is a minimum spanning forest $F(V, E)$.

3.3 Optimal Aggregation

The target of clustering is expected to make high intracluster compactness and intercluster dispersion [4]. Since a set of clustering hierarchies are built, we turn to choose an optimal clustering hierarchy as the input to our regression model by using a rational metric. As discussed in Theorem 3.1, Best Friend Clustering specifies a minimum spanning forest for each hierarchy intrinsically, and it connects all scattered clusters as a whole network. Thus, the clustering validity is analyzed based on the MST network, where metrics are quantified by the properties of MST.

Definition 3.2. Let a $T_i = (V_i, E_i)$ denote a minimum spanning tree in a best friend forest F. The intracluster compactness c_i for T_i is defined as:

$$c_i = \frac{1}{e_i} \sum_{(j,k) \in E_i} \omega(j, k), \tag{8}$$

where $e_i = |E_i|$.

Algorithm 4 shows the procedure of the best friend forest construction and the distance calculation of each connected component. The first function finds all best friend edges and records them in a global array. The second function traverses all connected components. It finds an unvisited node as a root of a MST and feeds it the third function which utilizes the Depth-First Search (DFS) to traverse all the nodes in the MST and accumulate the number and their weights in two global arrays $c[]$ and $e[]$.

Definition 3.3. Let a best friend forest F_k be the kth hierarchy produced by Best Friend Clustering. Assume that there are m clusters T_1, T_2, ..., T_m in F_k, and cluster T_i contains v_i samples. Then the intercluster dispersion d_i for T_i is defined as:

$$d_i = \min\{d(\overline{x}_i, \overline{x}_j) | 1 \leq j \leq m, j \neq i\}, \tag{9}$$

where $\overline{x}_i$ and $\overline{x}_j$ are the new cluster centers and $d(\overline{x}_i, \overline{x}_j)$ is the Euclidean distance between cluster T_i and T_j.

To determine the optimal clustering hierarchy, a metric measured by hierarchical clustering index (HCI) is defined in Definition 3.4. It combines the intracluster compactness c_i and the intercluster dispersion d_i.

Definition 3.4. Let MSTs T_1, T_2, ..., T_m denote clusters in a best friend forest F_k for the kth dendrogram hierarchy dendrogram produced by Best Friend Clustering. Then the $HCI(k)$ is defined as a linear combination of the intracluster compactness and intercluster dispersion:

$$HCI(k) = \frac{1}{m} \sum\nolimits_{i=1}^{m} \left(\frac{d_i - c_i}{d_i + c_i}\right), \tag{10}$$

and the optimal clustering hierarchy is:

$$k_{opt} = \arg\max\{HCI(k)\}. \tag{11}$$

Figure 4 illustrates the computing process of HCIs for the global cities example. With scaled distance denoted on each edge, hierarchical MST results are depicted in Figure 4 for the first two clustering hierarchies. From the Definition 3.2 for intracluster compactness, we have $c_1 = 1$, $c_2 = (3 + 5 + 4 + 7)/4 = 4.75$, $c_3 = 2$, $c_4 = (2 + 1)/2 = 1.5$ in S_1, and $c_1 = 30$, $c_2 = 16$ in S_2. Since similar vertices are merged iteratively, an important observation is that the

Algorithm 4 Best Friend Forest Construction

Require: $visited[] = 0, c[] = 0, e[] = 0, mst\_num = 0$

```
function InitializeBestFriendForest
    for i = 0 → n − 1 do
        j = findBestFriend(i)
        addEdge(i, j, ω(i, j))
    end for
end function
function TraverseMST
    for i = 0 → n − 1 do
        if !visited[i] then
            Search(i)
            c[mst_num]/ = e[mst_num]
            mst_num = mst_num + 1
        end if
    end for
end function
function Search(i)
    visited[i] = 1.
    e[mst_num]+ = 1
    while getNextNeighbor(i, &k) do
        if (!visited[k]) then
            c[mst_num]+ = getEdgeWeight(i, k)
            Search(k)
        end if
    end while
end function
```

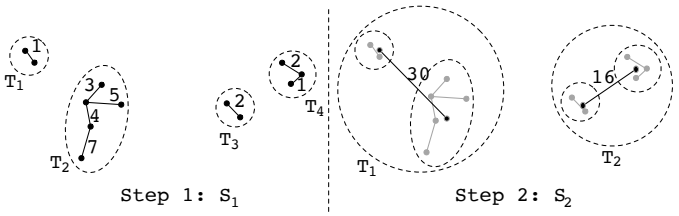

Fig. 4. Hierarchical MST results by Best Friend Clustering.

d_i for cluster i in S_k is exactly the best friend weight for vertex i in S_{k+1}. For instance, $d_1 = d_2 = 30$, $d_3 = d_4 = 16$ are obtained in S_1 by the information acquired in S_2. Thus, we have $HCI(1) = \frac{1}{4}\sum_{i=1}^{4}(\frac{d_i - c_i}{d_i + c_i}) = 0.82$ by Equation 10. Similarly, we obtain the value of $HCI(2) = 0.51$ with the new $c_1 = 30$, $c_2 = 16$, and $d_1 = d_2 = 70$ in S_2. The specific values of d1 and d2 are measured by the distance between two new merged clusters, and it is acquired from the new centroids calculated in S_3. Here we give the values $d_1 = d_2 = 70$ in S_2 straightforward for brevity. Apparently, the clustering result of S_1 is evaluated better than S_2 due to $HCI(1) > HCI(2)$. It is worth noting that HCI is not an introduced parameter but a metric to measure the clustering quality. It can be combined with BFC simply for determining an optimal clustering hierarchy k_{opt} with a maximum HCI automatically. Therefore, the whole clustering process is still parameter-free.

3.4 Balanced Partitions

Based on the profiling results, we observe that the partitions by clustering are typically irregular and imbalanced. This makes computing nodes load-imbalanced, and thus we need to devise a new partition algorithm to achieve data parallelism. In our design, a balanced partition on p computing nodes means that the number of samples on each node is close to $n_p = n/p$. Based on the data organization of clustering results, we propose a balanced partition algorithm by utilizing a backtracking mechanism, which is composed of MERGE and SPLIT operations.

MERGE is performed on the piecemeal clusters with small sizes. Figure 5 (a) shows a case with 4 processes on the clustering result S_1. Since we have $n_p = 12/3 = 4$, the sizes of cluster C_3 and C_4 are too small compared to n_p. Therefore, we sort the S_1 by cluster size, and merge the small clusters into the same node for a total size close to n_p on it. It is worth noting that the models are still trained independently on each node, which means the objective of MERGE is to make up the n_p instead of mixing models on different clusters. In Figure 5 (a), cluster C_3 and C_4 are merged into process 3 as group G_3 and G_4. The total sizes are 3, 5, and 4 respectively on process 1 to 3, which achieves a balanced partition for S_1.

SPLIT is utilized to separate the large clusters of which the size is much larger than n_p. Since the sample pointers in same clusters are moved together in each iteration, we achieve SPLIT operation by backtracking mechanism based on the clustering array structure. A case with 5 processes on S_2 is illustrated in Figure 5 (b). In this case, we have $n_p = 12/5 = 2.4$ while the original cluster C_3 are larger than n_p apparently. Thus we perform backtracking on the S_2 by the tracks of pointers. The backtracked $S_2^{'}$ contains 2 new split clusters, where the size of $C_3^{'}$ is still larger than n_p. Then we perform another SPLIT to separate $C_3^{'}$ into G_3 and G_4 respectively. Therefore, the group G_1 to G_5 are dispatched to nodes evenly, and they are trained as independent models. By utilizing the backtracking mechanism, a principle is followed that closer samples are always guaranteed to gather together after the SPLIT operation, and spatial locality is also exploited simultaneously.

Algorithm 5 Parallel Regression

Require: n samples for training, k samples for testing, best-clustered results A

1: $p \leftarrow$ rank of a process
2: Initialize best Mean Squared Error $MSE^* \leftarrow \infty$
3: Balanced Partitions on n samples in A for A^p
4: **function** REGRESSION
5: TRAINING(A^p)
6: TESTING(M^p)
7: Reduce: MSE=$(\sum_{p=1}^{P} e^p)/k$
8: **if** $(p = 0)\&\&(MSE < MSE^*)$ **then**
9: $MSE^* = MSE$
10: **end if**
11: **end function**
12: **function** TRAINING(A^p)
13: $C^p \leftarrow$ Cluster Number in A^p on Rank p
14: **for** $i = 0 \rightarrow C^p$ **do**
15: Training Model M_i^p for Cluster i on Rank p
16: **end for**
17: **end function**
18: **function** TESTING(M^p)
19: Initialize local MSE $e^p \leftarrow 0$
20: **for** $i = 0 \rightarrow k - 1$ **do**
21: **if** FindNearCLuster$(i) = C^*$ **then**
22: Select best M^* for prediction
23: Update e^p
24: **end if**
25: **end for**
26: **end function**

3.5 Independent Prediction

Parallel regression methods normally construct p independently models, where p is the number of processors (hardware parallelism) [48], [51]. They often take p into consideration from the beginning as an input parameter.

Two major disadvantages exist with this approach. First, p is essentially irrelevant to the input data and may mismatch the intrinsic structure of data samples. Our method employs the Best Friend Graph hierarchically and efficiently constructs a series of cluster hierarchies that does not depend on any predefined value. Second, existing methods may require a data reorganization to improve the load-balance. However, this procedure may lose the relationship information among the data that is moved from one cluster to others and hurts the compactness of the final models.

The parallel regression is summarized in Algorithm 5 formally. First, we perform a training phase in line 5. In our work, each process contains at least one cluster after balanced partition, which means corresponding models are generated by training separately form line 12 to line 17. Then model

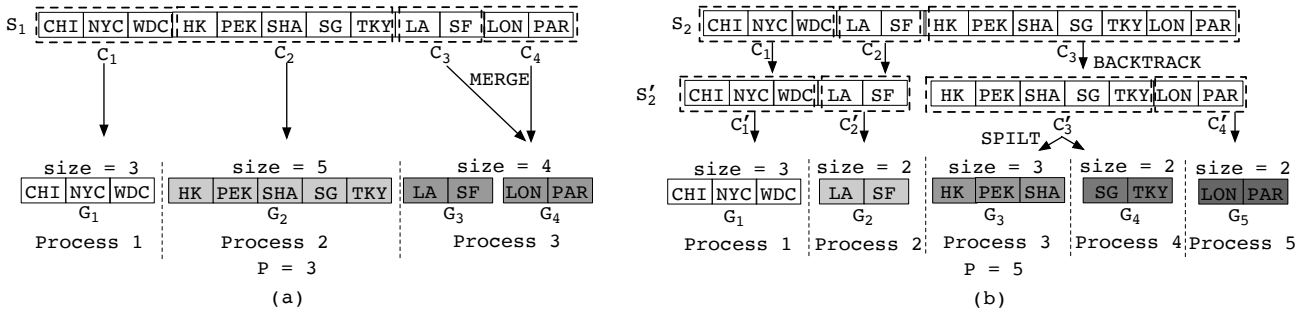

Fig. 5. Balanced partitions to achieve load balancing on each process. Figure 5 (a) describes the MERGE for small tasks, while Figure 5 (b) shows the SPLIT for large tasks based upon backtracking mechanism. Abbreviated forms are used for city names.

parallelism is utilized for making predictions independently on each process from line 18 to line 26. For a given test sample x^*, we only use the corresponding model M^* to perform a prediction if its closest cluster center is C^* on process p. As illustrated in line 23, instead of conducting communication regularly, the errors are accumulated on each node first. Reduce operation is only required at last in line 7 to make statistical analyses. Since an intact message is cheaper than scattered messages in MPI communication, the latency overhead is further reduced by this optimization.

4 Implementation

4.1 Parallelization

To reduce the computation, memory, and communication overheads, we design an efficient parallel implementation of our method. The parallelization of the first function of Algorithm 4 is straightforward. The data samples are evenly distributed to all processors and the calculation is parallelized accordingly. The distribution of the example is shown in Figure 6 where each process is dispatched with 3 samples for computing their nearest neighbors respectively.

The traversal of the best friend forest and the calculation of compactness, i.e. the second function of Algorithm 4 seems to be an inherently serial task. However, provided with Lemmas in the previous section, we are able to identify individual components (trees) with the edge information in the best friend forest. With Lemma 3.2, we know that each cycle identifies a tree, and with Lemma 3.4 a cycle is easy to find by searching its two equal edges.

Algorithm 6 provides the parallel version of the second function TRAVERSEMST in Algorithm 4. All the best friend edges are gathered to all processors in line 1. Every processor then finds all pairs of equal edges by sorting all $|V|$ edges or using a hash method from line 4 to line 12. The number of trees in the best friend forest is the number of pairs of equal edges. Each processor traverses a set of trees, and the traversal of each tree starts with either of the two nodes on its cycle in line 14. Overall, it only transfers short messages that only contain two values: the pairs of nearest neighbors and the corresponding shortest distances in our implementation.

4.2 Computation

The process of finding the nearest neighbor dominates the overheads of the computation. For a *Best Friend Graph* $G(V, E)$ where the vertex set V consists of n data elements, there is no need to compute a full $n \times n$ distance adjacent matrix on a large-scale dataset in practice. For example, the best friend can be obtained easily via fast approximate nearest neighbor methods with O(log(n)) time complexity (such as FLANN [35], k-d tree [19]).

Algorithm 6 Parallellization of TRAVERSEMST in Algorithm 4

1: Gather the whole Best Friend Forest
2: $hash.initialize()$
3: $mstNum = 0$
4: **for** $i = 0 \rightarrow n - 1$ **do**
5: $\quad j = getBestFriend(i)$
6: $\quad$ **if** $hash.exist((i, j))$ **then**
7: $\quad\quad startNode[mstNum] = i$
8: $\quad\quad mstNum+ = 1$
9: $\quad$ **else**
10: $\quad\quad hash.add((i, j))$
11: $\quad$ **end if**
12: **end for**
13: **parfor** $i = 0 \rightarrow mstNum - 1$ **do**
14: $\quad$ SEARCH($startNode[i]$)
15: **end parfor**

Instead of using these ready-made fast library, we adopt Euclidean distance for pairwise distance calculation in a more general implementation. Since the number of data samples decrease at least half at each recursion, we average them and use the mean vectors for computing the best friend. This also simplifies the computation and optimizes the time complexity from O(n^2) to O($n$log(n)), especially with the best case O(n) by our constant-round convergence. To further leverage the ability of vector processing units in modern CPUs, we group vl data samples in a vector register and perform vl calculations in a SIMD style, where vl is the maximum number of $double$ elements a register can hold. This improves the computation efficiency significantly.

4.3 Data Organization

Provided with the load-balancing scheme described in Section 3.4, we design a simple data organization method to guarantee an efficient implementation.

After building the Best Friend Forest in each clustering hierarchy, sample pointers divided to the same cluster are swapped to stay together. Each sample actually represents a cluster which is clustered by points in the last round. The relative position of these points in a low-hierarchy cluster are fixed, i.e. they are swapped as a whole big sample point.

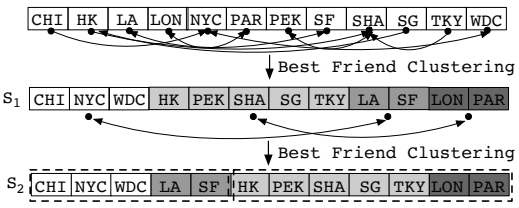

Fig. 6. Data organization in Best Friend Clustering. Clusters are distinguished by colors and boxes in two steps respectively.

This data organization is critical to balanced partition since it guarantees that the closer samples are in the array, the more similarities match. Figure 6 depicts the process for pointer swaps in each clustering hierarchy. Although London, Paris, and Hong Kong are clustered into the same group in the S_2 hierarchy, London and Paris contain more similarities as they are far away from Hong Kong in the clustering array. Thus, the load-balancing scheme is implemented simply based on a newly-designed data structure. The data organization updates as the BFC steps forward, which only requires negligibly simple pointer swaps without additional operations.

5 Experiments

5.1 Setup

Platforms. We develop the library in C++ and our experiments are performed on a high-performance cluster. Each machine of the cluster is composed of two Intel Xeon Platinum 9242 processors with 2.30 GHz clock speed (turbo boost frequency of up to 3.80 GHz), which owns 96 physical cores organized into two sockets. The processor contains a 71.5 MB smart cache. AVX512 instruction set extension is supported and it's able to conduct operations for 8 double-precision floating-point data in a SIMD manner.

Benchmarks. The experiments are conducted in two parts. First, the proposed Best Friend Clustering is evaluated with three classic 2D datasets. Results of K-Means [49], AHC [38] and MSTC [29] methods are also presented as different benchmarks for comparison. Then we turn to the evaluation on convergence, accuracy, and scalability for our regression library with five real large-scale datasets. Since DCKRR [52] and BKRR2 [48] are two closely related papers, they are employed as two benchmarks in this paper. Moreover, we alternate the clustering methods in BKRR2 with AHC [38] and MSTC [29] as AHCKRR and MSTKRR respectively. Thus the experimental configurations for parallel regression cover the whole spectrum of representative clustering methods: BFCKRR (Best Friend Clustering), BKRR2 (K-Means), AHCKRR (Agglomerative Hierarchical Clustering), and MSTKRR (Minimum Spanning Tree Clustering).

Datasets. Three classic shape datasets are employed to demonstrate the quality of clustering in Table 5.1, which can be obtained in the Clustering Basic Benchmark (CBB) [18]. They represent well-understood clustering problems and are widely-used benchmarks for checking the applicability of clustering algorithms [18], [20], [41], [50]. Then Datasets Million Song Data (MSD) and Cadata are used as two of our evaluated datasets for regression since they are both used in the paper of DCKRR and BKRR2 [48]. To further justify the scaling efficiency of our approach, we use another three real datasets, which contain the data on higher dimensions and interdisciplinary research. The details of these five datasets are sorted by #Train and summarized in Table 2. All these datasets are available in the UCI Machine Learning Repository [15]. The distance measure is also adopted fairly by using Euclidean distance.

TABLE 1
Description of Classic Shape Datasets.

Dataset	#Samples	#Labels	#Dimensions
Zahn's Compound [50]	399	6	2
Aggregation [20]	788	7	2
R15 [41]	600	15	2

TABLE 2
Description of Large Real Datasets.

Dataset	#Train	#Test	#Dimensions	Field
cadata	18,432	2,208	8	Housing
Proteins	40,730	5,000	9	Biomedicine
APS Failure	60,000	16,000	171	Vehicle
MSD	463,715	51,630	90	Music
Gas Sensor	4,095,000	900,900	20	Chemistry

5.2 Visualization

For better clarity, the quality of Best friend Clustering is visualized intuitively in Figure 7 by utilizing the classic shape datasets in Table 5.1. To evaluate the classification performance quantitatively, the adjusted mutual information (AMI) [42] is used in Figure 7 to measure the similarity between partitions on ground truth data and the cluster assignment obtained by the methods. Due to that the performance of K-means is greatly influenced by the K value, we set K with real #labels straightway. Nevertheless, K-means yields poor results especially on Zahn's Compound [50] and it cannot separate the arbitrary shapes smoothly. AHC and MST clustering outperform K-means in most cases. Compared with three methods mentioned above, we can observe that BFC maintains the merges quite well and clusters these data better than the considered benchmarks.

5.3 Convergence

Since the connected samples generated by our clustering method are grouped at least half recursively, BFC could reach a fast convergence in O(log$N$) rounds. The details for HCIs and #clusters are given in Table 3 and Table 4 respectively, where the best convergent hierarchies are highlighted. The average sizes for each cluster are 56, 4, 15,000, 453, and 2,379 respectively on five datasets. Moreover, the number of iterations still falls in low single digits on large-scale datasets like Gas Sensor, which illustrates that the clustering is less sensitive to the increasing training size and appropriate for scaling cases.

5.4 Accuracy

The proposed balanced partition strategy has little impact on the accuracy of the model since it reserves the samples with most similarities in the same cluster. Nonetheless, to assure the correctness of our implementation, we perform the

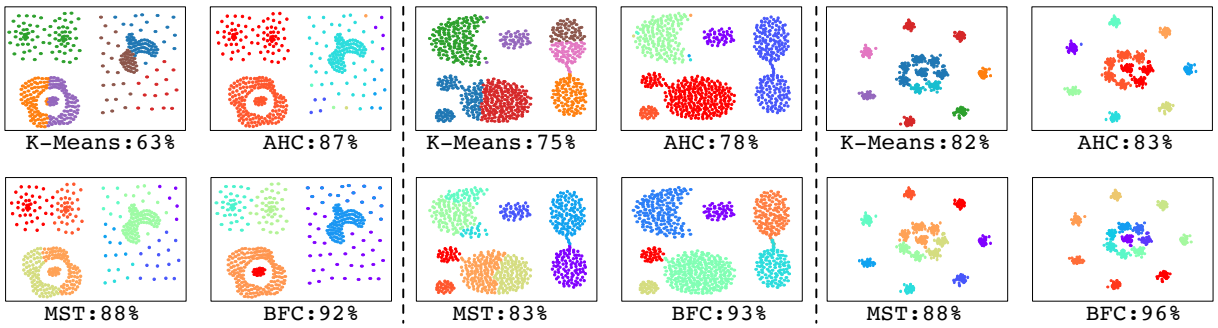

Fig. 7. Visualization for considered methods on Zahn's Compound [50] (left), Aggregation [20] (Center), and R15 [41] (right) with AMI score annotated.

TABLE 3
Hierarchical HCIs in Best Friend Clustering

STEPS	1	2	3	4	5	6	7
CADATA	0.31	0.51	0.26	0.45	0.24	0	-
PROTEINS	0.65	0.44	0.28	0.14	0.13	0.11	0
APS FAILURE	0.30	0.56	0.27	0.35	0.58	0.50	0
MSD	0.09	0.25	0.19	0.02	0	-	-
GAS SENSOR	0.22	0.38	0.62	0.24	0.17	0.13	0

TABLE 4
#clusters for each hierarchy in Best Friend Clustering

STEPS	0	1	2	3	4	5	6	7
CADATA	18,432	4,737	326	64	8	2	1	-
PROTEINS	40,730	11,273	724	191	32	6	2	1
APS FAILURE	60,000	15,230	712	124	11	4	2	1
MSD	463,715	54,307	1,023	60	7	1	-	-
GAS SENSOR	4,095,000	231,541	13,115	1,721	202	10	2	1

configurations scaling from 96 to 12,288 cores and compare with the reported accuracy (measured by MSE). To give a fair comparison, the best parameters were finely tuned from the same parameter set to achieve the lowest MSE in different methods. As shown in Figure 8, BFCKRR achieves the lowest MSEs when KRR techniques are employed. Moreover, BFCLR and BFCSVR could also make a superior prediction in most cases. Among all considered baselines, DCKRR and BKRR2 produce a poor quality, which adopts no samples clustering and K-means clustering respectively. As the core increases, high accuracy is obtained steadily by our library while the MSEs of other baselines ripples drastically. This illustrates that our balanced partition algorithm adds great support to accuracy in scaling cases.

5.5 Scalability

Figure 8 also illustrates the scalability for different methods on five datasets. We observe that our library consistently achieves high performance on them compared to baselines. With a larger dataset like Gas Sensor, gaps between them are further widened and it is even more than 13.6x faster than BKRR2. Moreover, the time for our library decreases regularly as we double the number of cores, while both DCKRR and BKRR2 exhibit a bad scaling performance by a growing curve. This illustrates that the K-means-based or DC-based methods are poor in large-scale regression, where performances are jeopardized cumulatively by the increasing K value and expensive DC operations. With similar parallel implementation achieved, MSTKRR outperforms BKRR2 in most cases. However, AHCKRR suffers from the lowest performance as agglomerative style algorithms come with a quadratic time complexity inherently.

To dissect the procedures of our library and see how it varies as the size of datasets grows, we provide a quantitative look into the cases of two typical datasets, i.e., the smallest and largest ones. Figure 9 compares the logarithmic time for clustering I/O, clustering, regression I/O, regression, and communication in our library. Here the I/O means all cost of the non-computational parts except communication. Upon inspection, it becomes distinct that the increasing cores also aggravate the communication and I/O cost. Despite the scaling pressure brought by communication and I/O, our library still obtains a sustained scaling performance. Table 5 shows the analytical proportion and speedup of two datasets. Interestingly, although the average proportion on different datasets contains little difference, our library can obtain a higher speedup on Gas Sensor. Based on conjoint analyses on Figure 9, we can observe that the key contribution lies in the better scaling efficiency of parallelizable procedures (clustering and regression parts) on large-scale datasets.

5.6 Discussion

In this subsection, we provide a quick recap on previous experiments to tease out the contributions from different aspects of our proposed method.

We first investigate the classification performance on the classic shape datasets compared to three typical baselines in Section 5.2. Apparently, our method could achieve a better classification intuitively. Then the convergence experiments in Section 5.3 demonstrate that our method could achieve a constant convergence stably even confronted with millions of sample elements. Accuracy experiments in Section 5.4 conduct cases scaling to 12,288 cores on various benchmarks. Since our balanced partition algorithm captures most similarities among samples, a competitive result is obtained on distributed systems. At last, the scalability experiments demonstrate that our regression library leveraging BFC outperforms the referenced benchmarks across a broad variety of configurations in scaling cases.

6 Related Work

Our paper shows closely concerns for two main lines of research. The first thread is to study the efficient clustering methods with better-quality on distributed systems. Bahmani *et al.* extended the k-means problems by a MapReduce

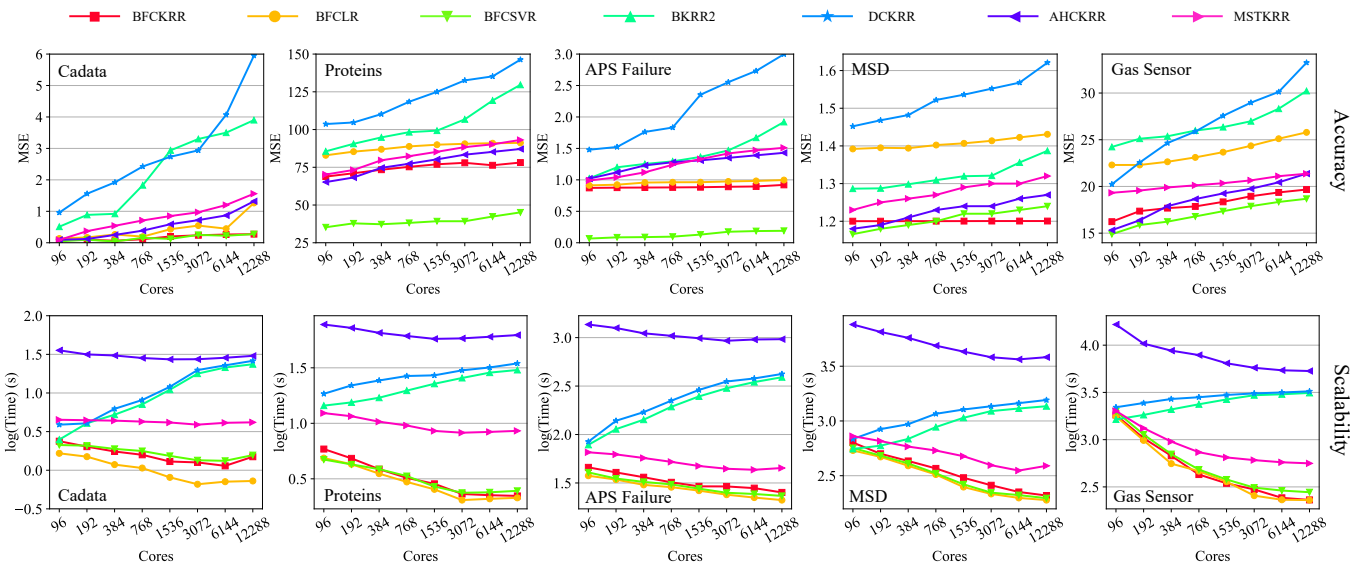

Fig. 8. Accuracy and scalability collected with the same configurations for different methods on real datasets. Our parallel library includes KRR, LR, and SVR techniques. They are applied with BFC method and abbreviated to BFCKRR, BFCLR, and BFCSVR respectively. Benchmarks for parallel regression applied with representative clustering methods are DCKRR (Divide-and-Conquer without clustering) [52], BKRR2 (K-Means) [48], AHCKRR (Agglomerative Hierarchical Clustering) [38] and MSTKRR(Minimum Spanning Tree Clustering) [29].

TABLE 5
Analytical Proportion and speedup for different procedures by our library on Cadata and Gas Sensor datasets.

Dataset	Cadata							Gas Sensor						
Component	C.I/O(%)[1]	C.(%)	R.I/O(%)	R.(%)	M.(%)	S.(C.+R.)[2]	S.	C.I/O(%)	C.(%)	R.I/O(%)	R.(%)	M.(%)	S.(C.+R.)	S.
96	2.80	63.98	0.34	28.28	0.84	1.00	1.00	1.88	64.84	0.35	31.51	0.89	1.00	1.00
192	3.58	54.96	0.54	25.75	1.49	1.34	1.18	3.64	62.33	0.67	29.38	3.04	1.85	1.76
384	4.54	46.88	1.54	25.15	4.00	1.74	1.36	5.97	53.34	1.22	27.59	8.67	3.21	2.70
768	5.94	40.41	2.21	22.73	10.73	2.19	1.50	11.00	46.31	2.35	22.80	15.53	5.95	4.27
1536	7.94	31.62	3.39	21.60	25.45	3.18	1.83	14.00	37.21	3.09	21.45	21.79	8.68	5.28
3072	8.48	25.35	4.33	18.22	34.86	3.99	1.88	16.50	26.83	3.78	16.60	29.45	13.61	6.14
6144	10.09	27.21	5.18	14.04	43.01	4.66	2.09	20.27	18.84	4.52	15.54	37.01	21.09	7.53
12288	8.29	20.73	4.48	12.70	39.45	4.38	1.59	21.75	17.46	4.92	14.86	39.12	23.60	7.92
Mean	6.46	38.89	2.75	21.06	19.98	2.81	1.55	11.88	40.90	2.61	22.47	19.44	9.87	4.57

[1] For better clarity, procedures for clustering, regression, and communication are abbreviated with C., R., and M. respectively.
[2] The speedup is also abbreviated to S..

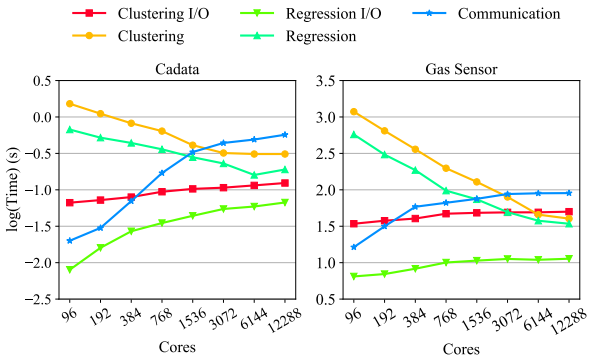

Fig. 9. Time dissection on Cadata & Gas Sensor datasets.

algorithm on distributed nodes [5]. Ene *et al.* optimized the greedy algorithm on MapReduce to solve the k-center problems and the local search strategy to address the k-median problems [16]. Subsequently, several papers studied similar problems with the MapReduce model [2], [6], [8]. Then Bouguettaya *et al.* built a hierarchy based on a group of centroids generated by K-Means to improve the efficiency of AHC [10], while it only was implemented for a single node.As for Graph Clustering, massive efforts were also put into studying efficient algorithms [1], [3], [13], [17]. Grygorash *et al.* presented the Hierarchical euclidean-distance-based MST clustering algorithm (HEMST). Given the number of clusters as an input, HEMST accelerate the convergence progress by merging multiple edges, while the edges larger than the threshold are required to be sorted and removed in order [21]. Jin *et al.* split the clustering problem into various overlapped subproblems by a Prim algorithm, solved each subproblem, and then merged them into an overall solution [26]. However, the MSTs are needed to store at the Map side and then shuffle to the Reducers. Bateni *et al.* [7] extended MST-based method called affinity clustering with two classic MST algorithms.It still relied heavily on MapReduce and requires moving all the edges to one machine serially for MSTs after the edges deletion. Moreover, affinity clustering did not reveal the exact running times and number of machines used in their experiments [7]. Wang *et al.* utilized a divide-and-conquer scheme to construct approximate MSTs, while the process to detect the long

edges of the MST is also highly sequential at an early stage for clustering [44], [54]. As a result, an efficient clustering algorithm competent for parallel computing on large-scale data is in need crucially to improve the accuracy of K-means, efficiency of AHC, and scalability of GC methods.

The second focus of this paper is closely related to distributed regression, which suffers from serious scalability problems in both computation time and memory usage [11]. Mini-batch Gradient Descent (MBGD) was used for training on batched data [23], while it was proposed for serial implementation. MapReduce-based methods [22], [47] made computation distributed locally which holds parts of the data. However, the overall cost in terms of computation and network is high because of the busy synchronization sweeps. The Divide-and-Conquer algorithm was then adopted on distributed systems for SVM and KRR [24], [51], [52]. Parallel SVM (PSVM) is presented recently to decrease memory and time consumption [9], [34]. Zhang *et al.* proved that regression with kernel method is more accurate than non-kernel methods, and DCKRR designed by them can outperform all the previous approximate methods [51]. You *et al.* [49] presented K-Means kernel Ridge Regression (KKRR) for efficient regression on clustered data. Recent work BKRR2 [48] was optimized on KKRR by averaging the loads on each node and had better accuracy than DCKRR and KKRR, which was considered as a state-of-the-art approach for parallel kernel regression. Thus, in this paper, the focus is paid to the comparison with two closely related papers DCKRR and BKRR2, which are also elaborated in the Introduction section.

7 Conclusion

In this paper, we propose a Best Friend Clustering method, which is more accurate, fast, and meanwhile parameter-free. Then we devise a strategy to determine the optimal aggregation through a well-defined metric on distributed systems. Moreover, a balanced partition inspired by backtracking is devised for load-balance in parallel implementation. At last, we integrate proposed methods with regression techniques as a parallel library, which shows superior performance on convergence, accuracy, and scalability.

Acknowledgments

This work is supported by National Natural Science Foundation of China under Grant No. 61972376, No. 62072431, No. 62032023; the Science Foundation of Beijing No. L182053.

References

[1] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 459–467. SIAM, 2012.

[2] Prajesh P Anchalia, Anjan K Koundinya, and NK Srinath. Mapreduce design of k-means clustering algorithm. In *2013 International Conference on Information Science and Applications (ICISA)*, 2013.

[3] Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 574–583, 2014.

[4] Tetsuo Asano, Binay Bhattacharya, Mark Keil, and Frances Yao. Clustering algorithms based on minimum and maximum spanning trees. In *Proceedings of the fourth annual symposium on Computational geometry*, pages 252–257, 1988.

[5] Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. Scalable k-means++. *Proc. VLDB Endow.*, 5(7):622–633, March 2012.

[6] Maria-Florina F Balcan, Steven Ehrlich, and Yingyu Liang. Distributed k-means and k-median clustering on general topologies. *Advances in Neural Information Processing Systems*, 2013.

[7] Mohammad Hossein Bateni, Soheil Behnezhad, Mahsa Derakhshan, Mohammad Taghi Hajiaghayi, Raimondas Kiveris, Silvio Lattanzi, and Vahab Mirrokni. Affinity clustering: Hierarchical clustering at scale. In *NIPS*, pages 6867–6877, 2017.

[8] MohammadHossein Bateni, Aditya Bhaskara, Silvio Lattanzi, and Vahab S Mirrokni. Distributed balanced clustering via mapping coresets. In *NIPS*, pages 2591–2599, 2014.

[9] Ron Bekkerman, Mikhail Bilenko, and John Langford. *Scaling up machine learning: Parallel and distributed approaches*. Cambridge University Press, 2011.

[10] Athman Bouguettaya, Qi Yu, Xumin Liu, Xiangmin Zhou, and Andy Song. Efficient agglomerative hierarchical clustering. *Expert Systems with Applications*, 42(5):2785–2797, 2015.

[11] CL Philip Chen and Chun-Yang Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information sciences*, 275:314–347, 2014.

[12] Gongwei Chen, Xinhang Song, Haitao Zeng, and Shuqiang Jiang. Scene recognition with prototype-agnostic scene layout. *IEEE Transactions on Image Processing*, 29:5877–5888, 2020.

[13] Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, Mohammad-Taghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1326–1344. SIAM, 2016.

[14] Norman R Draper and Harry Smith. *Applied regression analysis*, volume 326. John Wiley & Sons, 1998.

[15] Dheeru Dua and Casey Graff. Uci machine learning repository.

[16] Alina Ene, Sungjin Im, and Benjamin Moseley. Fast clustering using mapreduce. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011.

[17] Hossein Esfandiari, Mohammadtaghi Hajiaghayi, Vahid Liaghat, Morteza Monemizadeh, and Krzysztof Onak. Streaming algorithms for estimating the matching size in planar graphs and beyond. *ACM Transactions on Algorithms (TALG)*, 14(4):1–23, 2018.

[18] Pasi Fränti and Sami Sieranoja. K-means properties on six clustering benchmark datasets. *Applied Intelligence*, 48(12):4743–4759, 2018.

[19] Jerome H Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 1977.

[20] Aristides Gionis, Heikki Mannila, and Panayiotis Tsaparas. Clustering aggregation. *Acm transactions on knowledge discovery from data (tkdd)*, 1(1):4–es, 2007.

[21] Oleksandr Grygorash, Yan Zhou, and Zach Jorgensen. Minimum spanning tree based clustering algorithms. In *2006 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06)*, pages 73–81. IEEE, 2006.

[22] Qing He, Tianfeng Shang, Fuzhen Zhuang, and Zhongzhi Shi. Parallel extreme learning machine for regression based on mapreduce. *Neurocomputing*, 102:52–58, 2013.

[23] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8), 2012.

[24] Cho-Jui Hsieh, Si Si, and Inderjit Dhillon. A divide-and-conquer solver for kernel support vector machines. In *International conference on machine learning*, pages 566–574, 2014.

[25] Jie Hu, Yi Pan, Tianrui Li, and Yan Yang. Tw-co-mfc: Two-level weighted collaborative fuzzy clustering based on maximum entropy for multi-view data. *Tsinghua Science and Technology*, 26(2):185–198, 2020.

[26] Chen Jin, Md Mostofa Ali Patwary, Ankit Agrawal, William Hendrix, Wei-keng Liao, and Alok Choudhary. Disc: A distributed single-linkage hierarchical clustering algorithm using mapreduce. *work*, 23:27.

[27] Jon Kleinberg and Eva Tardos. *Algorithm design*. Pearson Education India, 2006.

[28] Max Kuhn. Caret: classification and regression training. *ascl*, pages ascl–1505, 2015.

[29] Jia Li, Xiaochun Wang, and Xiali Wang. A scaled-mst-based clustering algorithm and application on image segmentation. *Journal of Intelligent Information Systems*, pages 1–25, 2019.

[30] Jianjiang Li, Huihui Jiao, Jie Wang, Zhiguo Liu, and Jie Wu. Online real-time trajectory analysis based on adaptive time interval clustering algorithm. *Big Data Mining and Analytics*, 3(2):131–142, 2020.

[31] Kun Li, Honghui Shang, Yunquan Zhang, Shigang Li, Baodong Wu, Dong Wang, Libo Zhang, Fang Li, Dexun Chen, and Zhiqiang Wei. Openkmc: a kmc design for hundred-billion-atom simulation using millions of cores on sunway taihulight. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16, 2019.

[32] Kun Li, Liang Yuan, Yunquan Zhang, and Yue Yue. Reducing redundancy in data organization and arithmetic calculation for stencil computations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, 2021.

[33] Aristidis Likas, Nikos Vlassis, and Jakob J Verbeek. The global k-means clustering algorithm. *Pattern recognition*, 36(2):451–461, 2003.

[34] Pabitra Mitra, CA Murthy, and Sankar K Pal. A probabilistic active support vector learning algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(3):413–418, 2004.

[35] Marius Muja and David G Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2(331-340):2, 2009.

[36] Fionn Murtagh and Pedro Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.

[37] Guo Pu, Lijuan Wang, Jun Shen, and Fang Dong. A hybrid unsupervised clustering-based anomaly detection method. *Tsinghua Science and Technology*, 26(2):146–153, 2020.

[38] T. Sun, C. Shu, F. Li, H. Yu, L. Ma, and Y. Fang. An efficient hierarchical clustering method for large datasets with map-reduce. In *2009 International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 494–499, 2009.

[39] Zhanquan Sun and Geoffrey Fox. Study on parallel svm based on mapreduce. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2012.

[40] Yu Tian, Ruiqing Zheng, Zhenlan Liang, Suning Li, Fang-Xiang Wu, and Min Li. A data-driven clustering recommendation method for single-cell rna-sequencing data. *Tsinghua Science and Technology*, 26(5):772–789, 2021.

[41] Cor J. Veenman, Marcel J. T. Reinders, and Eric Backer. A maximum variance cluster algorithm. *IEEE Transactions on pattern analysis and machine intelligence*, 24(9):1273–1280, 2002.

[42] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: is a correction for chance necessary? In *Proceedings of the 26th annual international conference on machine learning*, pages 1073–1080, 2009.

[43] Ning Wang, Gege Guo, Baonan Wang, and Chao Wang. Traffic clustering algorithm of urban data brain based on a hybrid-augmented architecture of quantum annealing and brain-inspired cognitive computing. *Tsinghua Science and Technology*, 25(6):813–825, 2020.

[44] Xiaochun Wang, Xiali Wang, and D Mitchell Wilkes. A divide-and-conquer approach for minimum spanning tree-based clustering. *IEEE Transactions on Knowledge and Data Engineering*, 21(7):945–958, 2009.

[45] Wikipedia. Global city. Website, 2020. https://en.wikipedia.org/wiki/Global_city#Global_Cities_Index.

[46] Zhonghao Xue and Hongzhi Wang. Effective density-based clustering algorithms for incomplete data. *Big Data Mining and Analytics*, 4(3):183–194, 2021.

[47] Hailong Yang, Zhongzhi Luan, Wenjun Li, and Depei Qian. Mapreduce workload modeling with statistical approach. *Journal of grid computing*, 10(2):279–310, 2012.

[48] Yang You. Fast and accurate machine learning on distributed systems and supercomputers. 2020.

[49] Yang You, James Demmel, Cho-Jui Hsieh, and Richard Vuduc. Accurate, fast and scalable kernel ridge regression on parallel and distributed systems. In *Proceedings of the 2018 International Conference on Supercomputing*, pages 307–317, 2018.

[50] Charles T Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on computers*, 100(1):68–86, 1971.

[51] Yuchen Zhang, John Duchi, and Martin Wainwright. Divide and conquer kernel ridge regression. In *Conference on learning theory*, pages 592–617, 2013.

[52] Yuchen Zhang, John Duchi, and Martin Wainwright. Divide and conquer kernel ridge regression: A distributed algorithm with minimax optimal rates. *The Journal of Machine Learning Research*, 16(1):3299–3340, 2015.

[53] Xuan Zhao, Zhongdao Wang, Lei Gao, Yali Li, and Shengjin Wang. Incremental face clustering with optimal summary learning via graph convolutional network. *Tsinghua Science and Technology*, 26(4):536–547, 2021.

[54] Caiming Zhong, Mikko Malinen, Duoqian Miao, and Pasi Fränti. A fast minimum spanning tree algorithm based on k-means. *Information Sciences*, 295:1–17, 2015.

Kun Li received the B.E. degree in computer science and technology from Shandong University in 2016. He is currently pursuing the Ph.D. degree in computer science with the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences. His research focuses on parallel and distributed systems, high performance computing and machine learning.

Liang Yuan (Member, IEEE) received the PhD degree from the Institute of Software, Chinese Academy of Sciences, Beijing, China, in 2013. He is currently an associate professor with the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences. His research interests include large-scale parallel computing and heterogeneous computing.

Yunquan Zhang (Senior Member, IEEE) received the Ph.D. degree in computer software and theory from the Institute of Software, Chinese Academy of Sciences, Beijing, China, in 2000. He is a Full Professor of computer science with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. His research interests include high-performance parallel computing, with particular emphasis on large scale parallel computation and programming models, high-performance parallel numerical algorithms, and performance modeling and evaluation for parallel programs.

Gongwei Chen received the B.E. degree from the School of Information Engineering, University of Science and Technology Beijing, Beijing, China, in 2016. He is currently pursuing the Ph.D. degree in computer science with the Key Laboratory of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. His research interests include computer vision, machine learning, and image processing.