

Clustering Data in Secured, Distributed Datasets

Sayantan Dey(✉), Lee A. Carraher, Anindya Moitra, and Philip A. Wilsey

Department of EECS, University of Cincinnati, Cincinnati, OH 45221, USA
deysn@mail.uc.edu, leecarraher@gmail.com, wilseypa@gmail.com

Abstract. The massive growth in data generation and collection has brought to the forefront the necessity to develop mechanized methods to analyze and extract information from them. Data clustering is one of the fundamental modes to discover new insights from data. However, high dimensional data has its own challenges where many conventional clustering algorithms fails either in accuracy or scalability. To further complicate the issue, distinct subsets of sensitive data may reside in geographically separated locations with the sensitive nature of the data preventing (or inhibiting) its access for mechanized analysis. Thus, methods to discover information from the collective whole of these secured, distributed data sets that also preserves the integrity of the data must be found. In this paper we develop and assess a distributed algorithm that can cluster geographically separated data while simultaneously preserving the strict privacy requirements of non sharing of protected high dimensional data. We implement our algorithm on the distributed map-reduce based platform Spark and demonstrate its performance by comparing it to the standard data clustering algorithms.

Keywords: Data clustering · Data mining · Privacy preserving · Spark · Distributed · Map-reduce

1 Introduction

Many traditional data mining algorithms do not scale well with increases in the sample size or high dimensionality of modern data collection activities. Furthermore, the extreme scaling sizes of the expanding collections of data aggravates an issue in clustering known as *the curse of dimensionality* (or COD) [9]. The COD hinders clustering as the traditional distance matrices that are used for similarity measures can end up rendered meaningless. Another phenomenon in high dimensional big data is that they are sometimes distributed across multiple locations. When these large datasets are geographically distributed they cannot always be easily moved to a central location. Furthermore, in the cases of secure data stored in multiple locations (such as patient medical data), the

Support for this work was provided in part by the National Science Foundation under grant ACI–1440420.

© Springer Nature Switzerland AG 2019
S. Misra et al. (Eds.): ICCSA 2019, LNCS 11624, pp. 557–572, 2019.
https://doi.org/10.1007/978-3-030-24311-1_40

ability to exchange or share the data for analysis may be impossible; even if its common analysis would be significantly beneficial to understanding and identifying important membership classes (clusters) distributed collectively throughout the data sets. This has now become a well known research area by itself and is called *Privacy Preserving Data Mining* [2].

This paper presents a scalable distributed data clustering algorithm that can process very large high-dimensional distributed datasets *without sharing the collective data stored in the secured location.* The algorithm, called RPHash, combines *random projection* with *locality sensitive hashing* (LSH) to provide a scalable solution for clustering high-dimensional distributed datasets. The RPHash algorithm can be embedded in a map-reduce style distributed computing framework and operates without sharing or exchanging the data stored in the distributed locations. Essentially, RPHash distributes common seeds for the algorithms deployed to the remote sites. The local node algorithm uses random projection and LSH to compute candidate centroids and frequency counts for the candidate centroids. The candidate centroids and frequency counts are then combined and reduced at a central site to select specific global candidate centroids as the cluster centers. These selected cluster centers are then retransmitted back to the remote locations where the final steps of the algorithm can associate each local record to its centroid. This algorithm utilizes random projection as a destructive operation and exchanges only probable approximate centroids in the projected dimensional vector space. Thus, information on the specific nature of the original data is obfuscated such that its un-retrievable [11].

This work extends earlier designs of RPHash [12,17] to deliver higher performance, more accurate distributed clustering performance. We apply the concepts of dimensionality reduction, locality sensitive hashing, discrete subspace quantizer lattices and tree based clustering to achieve this capability. Our results demonstrate that our algorithms is fast, accurate and stable. Its accuracy is comparable to other well known standard clustering algorithms such as k-means [18], Kmeans++ [8], SOTA [19], and Ward's Hierarchical agglomerative [27], while being much faster in general. The RPHash software is released under an open source license and is publicically available from git repositories at https://github.com/wilseypa/rphash-java.

This paper is organized as follows. Section 2 summarizes some of the related work done in the area of privacy preserving data mining. Section 3 presents some background information. Section 4 introduces the RPHash algorithm. Section 5 explains the datasets used, experiments performed and comparison of results to standard algorithms. Finally, Sect. 6 contains some concluding remarks.

2 Related Work

The need for distributed Privacy Preserving Data Mining is surveyed in [2]. Most work in this area is based on the application of cryptographic encryption that are basically secure sub-protocols through which the data is exchanged. Two algorithms described in [23] and [22] address distributed clustering. In [23] the

authors approximate each feature vector so as to minimize data transmission between nodes and thereby compromising on the accuracy of the result. There is a trade-off between the accuracy and percentage of approximation as seen from their results. They rely on transmitting only the approximation vectors. Jagannathan *et al.* [22] develop an algorithm called K-clustering which is based on the K-Means algorithm. Their algorithm preserves privacy as they do not reveal the intermediate candidate cluster centers.

3 Background

The RPHash algorithms uses *random projection* and *locality sensitive hashing* to improve performance and obfuscate secured data to facilitate privacy preserving data mining. The projection step relies on the bounded error results from the Johnson-Lindenstrauss Lemma (JL-Lemma) [34] and the DB-Friendly projection studies [1] (both projection methods are implemented in the system, but only one is used for any given run). Locality Sensitivity Hashing (LSH) is a family of hashing methods that tend to hash higher-dimensional data to buckets with the goal of mostly placing similar items into the same bucket. For RPHash, the (LSH) function is employed, as a probabilistic representation of vector locality to improve the prohibitive limits of the subspace embedding dimensionality requirements of the JL-Lemma. Stated more formally, an LSH function is any hash function with the property that hashed records with more similar components are more likely to be hashed to the same bucket than records with fewer similarities. That is:

Definition 1 (Locality Sensitive Hash Function [13]). A hash function $h \in \mathbb{H} = \{h : S \to U\}$ is $(r_1, r_2, p_1, p_2) -$ sensitive if for any $u, v \in S$

$$\text{if } d(u,v) \leq r_1 \text{ then } Pr_{\mathbb{H}}[h(u) = h(v)] \geq p_1$$
$$\text{if } d(u,v) > r_2 \text{ then } Pr_{\mathbb{H}}[h(u) = h(v)] \leq p_2$$

Informally, this states that exists a family of hash functions (LSH functions) $\mathbb{H}$. When a vector is hashed by a hash function selected uniformly at random from $\mathbb{H}$, it will hash vectors u and v into the same bucket with a high probability p_1 if their distance is bounded by r_1 and with low probability p_2 if the distance exceeds r_2. While considerable work has gone into finding better and near optimal [6] functions for optimizing the signal to noise ratio for LSH functions, the best solutions often require multiple passes over the data to build data aware functions.

Map-Reduce [15] is a paradigm where an algorithm is written so that it can be efficiently deployed for distributed execution. The distributed computing machines may be locally connected or geographically distributed to form a cluster. The map-reduce framework helps enable the processing of very large datasets in reasonable time. This is achieved by decomposing an algorithm into a map function and a reduce function. The input data is converted into tuples of key-value pairs and segmented into parts. The map function operates on these

segmented files of tuples on compute nodes where each segment is assigned. The result of key-value pairs are stored and sorted by the keys. The reducers task takes in all the results from the mappers in parallel sorting them and producing a new combined key, value pair grouped by the same key. The reduce function then applies a user defined function to process this to produce the final results.

Hadoop [36] and Spark [37] are two frameworks or engines for implementing the distributed processing of MapReduce. Spark which was released later than Hadoop also has its streaming API which allows for processing data streams. Spark has more functionalities than Hadoop and is a hundred times faster than Hadoop when computation is in memory and ten times when done on disk.

4 The RPHash Algorithms

RPHash [11] is a data clustering algorithm. It can be used for dense region and microcluster identification. This algorithm requires two map-reduce phases for its distributed version implementation. RPHash-TWRP is a version of RPHash that varies its LSH function with the usage and construction of a tree to identify clusters while requiring only one map and reduce phase.

The degenerative cases for LSH k-nearest neighbor search is used for identifying candidate cluster centers in RPHash. In the (RPHash) algorithm, both approximate and randomized techniques are employed to provide a stochastic element to this clustering algorithm. To combat the curse of dimensionality, RPHash performs multi-probe, random projection of high dimensional vectors to the unique partitions of the Leech Lattice (Λ_{24}) [5] or hypersphere surface [30]. Then clustering region assignments are performed by decoding vector points into partitions of the Lattice.

The sequential implementation of the RPHash algorithm relies on the efficient Leech lattice decoder of Vardy, Sun, Be'ery, and Amrani [3,4,29,33] as a discrete space quantizer. The 24 dimensional subspace partitioned by the Leech Lattice is small enough to exhibit the spherical clustering benefit of random projection. Low distortion random embeddings are also feasible for very large dataset objects while avoiding the occultation problem [31]. Projected clustering of representative cluster centroids will not in general be correlated with other projections of data into projected cluster centroids. To recover data from the projection step, we must map projected vectors back to their original un-projected data space counterparts. The original data space vectors is then used to compute centroids corresponding to the clusters in the projected space.

The Distributed version of the RPHash is shown in Algorithms 1 and 2. We assume n nodes are there. x_k is the data vector of a partial dataset X. P denotes the set of projection matrices. H is a LSH function. C is the set of set of bucket collision counts. m and d original and projected dimensions. This approach is inspired from the works of [32] and [10].

In contrast to the original RPHash method that only updates LSH buckets, the RPHash-TWRP method combines bucket updation with a counter that increments the counts of all sub-hashes as well. This bares a slight resemblance

Algorithm 1. 2-Pass RPHash

Data: n: number of compute nodes used

MapPhase 1
forall the *nodes* n **do**
  forall the $x_k \in X$ **do**
    forall the $p_i \in \mathbb{P}$ **do**
      $\tilde{x_k} \leftarrow \sqrt{\frac{m}{d}} p_i^\intercal x_k$
      $t = \mathbb{H}(\tilde{x_k})$
      $L[k][i] = t$
      C.add(t)

ReducePhase 1
forall the $C_i \in n$ **do**
  get C_i from n nodes
$C = merge(C_i)$

MapPhase 2
forall the *nodes* n **do**
  forall the $x_k \in X$ **do**
    forall the $c_i \in C.top(K)$ **do**
      if $L[k] \cap M[i][0] \neq 0$ **then**
        $\Delta = M[k] - x_k$
        $M[k] = M[k] + \Delta / count$
        $L[k]$.add($M[i][0]$)
  Result: M_i

ReducePhase 2
forall the $M_i \in n$ **do**
  get M_i from n nodes
$M = merge(M_i)$
Result: M

to Liu *et al.* [24] while adapting their algorithm to work in distributed setting and without using any supervised learning. RPHash-TWRP can use a variety of metrics for the tree splitting condition. Unlike Liu *et al.* RPHash-TWRP does not concern itself with the more complicated to compute C4.5 entropy method, or the possibility of splitting clusters with random hyperplanes. RPHash-TWRP, avoids the latter restriction by virtue of its application to high dimensional data sets and the probability of splitting a cluster going to zero as the dimensionality grows. The theorem below is a consequence of the curse of dimensionality.

Theorem 1 (Hyperrectangle Splitting). *The probability of splitting a hyperrectangular region into two equal mass clusters where subsequent dimensional cuts contain the smaller of the two induced regions region approaches 0 exponentially in d.*

Algorithm 2. Adaptive LSH

$i = 1$
$ct, ct_prev = C(\mathbb{H}^{i+1}(x)), C(\mathbb{H}^{i}(x))$
while $i < n$ ***and*** $2ct > ct_prev$ **do**
  $ct_prev, i = ct, i + 1$
  $ct = C(\mathbb{H}^{i}(x))$
Result: $\mathbb{H}^{i}(x)$

$$\lim_{d\to\infty} \frac{Vol(R) - Vol_{removed}(R)}{Vol(R)} = 0,\ R\ Rectangle\ \in \mathbb{R}^d. \tag{1}$$

Proof.

Let X s.t. $x_j = [0...c...0] \in \mathbb{R}^d$ is orthogonal, $c \in [0, 1)$,
$\sum_{i}^{n} x_i = \mathbb{P}$, is a plane in $\mathbb{R}^d$,
R is a unit hyper-rectangle in $\mathbb{R}^d$ with $Vol(R) = 1$.

Let: $S_1(p)$ be the volume of the projection of R on x_p

If we restrict $S_1(p) + \tilde{S_1}(p) = 1$, $S_1(p) \leq \tilde{S_1}(p)$ for all p we obtain a dimension-wise construction for the volume of the smaller region of a hyhper-rectangle split by a hyperplane.

Next consider the volume of such a hyper-rectangular region $V(R_{s(X)})$

$$V(R_{s(X)}) = \prod_{p}^{n} S_1(p) \text{ where } S_1(p) \in U[0, \tfrac{1}{2}).$$

$V(R_{s(X)}) \leq 2^{-n}$ for all n

$\lim_{n\to\infty} 2^{-n} = 0$

$\Rightarrow V(R_{S(X)}) + V(\tilde{R}_{S(X)}) = V(s)$

$V(\tilde{R}_{S(X)}) = V(s)$ as $d \to \infty$ ■

Algorithms 3 and 4 together constitutes the RPHash-TWRP algorithm, where x denotes a single vector belonging to the dataset X, n is the number of compute nodes, m and d are the projected and original number of feature dimensions, $p \in P$ is the projection matrix, h is the hashed vector obtained by hashing with the hash function H, and C is the set of centroids and hash ids. In the off-line Algorithm 4 we output a overestimate of our centroids denoted by L. We can then use any algorithms such as k-means or Hierarchical Agglomerative to reduce the centroids to the desired number. The algorithm is linear in the input vector size X. For each vector RPHash-TWRP must compute the projection, and update the counter (Algorithm 3). This algorithm introduces two new operations the + to mean population weighted addition, and $\gg$ for the bit shift operation.

Algorithm 3. Tree Generation

MapPhase
forall the *nodes* n **do**
  forall the $x \in X$ **do**
    $\tilde{x} = \sqrt{\frac{m}{d}} p^{\intercal} x$ `// Projection`
    $h := \mathbb{H}(\tilde{x})$ `// LSH Hashing`
    while $h > 0$ **do**
      $h = h \gg 1$
      $x' = C[h] + x$
      C.add(h, x')
ReducePhase
forall the $C_i \in n$ **do**
  get C_i from n nodes
$C = merge(C_i)$
Result: C

Projection using the db friendly approach of Achlioptas [1] can be performed in $dm/3$ operations where d is the original dimensionality, m is the projection sub-dimension. The off-line step consists of exploring and updating the count records (as shown in Algorithm 4). In general it follows a depth first search traversal for candidate clusters with a worse case complexity of exploring all non-leaf nodes, $\theta(2^{m-1})$.

Algorithm 4. Off-line Tree Search

forall the $H \in sort(C.ids)$ **do**
  if $2C[H] < C[H \gg 1]$ **then**
    $C[H \gg 1] = 0$
$L = []$
forall the $h \in sort(C.counts)$ **do**
  $L \leftarrow$ medoid$(C[H])$
return L

4.1 Distributed RPHash-TWRP on Spark

The RPHash-TWRP algorithm was implemented on Apache Spark as shown in Fig. 1. It is important to note that the Spark framework was used to suit our need of a framework on which the distributed version can be run. Generally the dataset is maintained on a shared drive and then Spark distributes it to the nodes for use. In our case the data is composed of multiple secured datasets located at the compute site. Thus to make use of the Spark infrastructure, a null RDD (Resilient Distributed Dataset) is created and distributed to satisfy the requirement of Spark. The input to the driver node is the JAVA JAR (Java

ARchive) file containing the code to be run, the file locations, and the number of cluster centers (K) to find. The code generates a seed which it distributes to the worker nodes along with the K parameter and the null RDD created. The JAR is kept at a shared location. The input JAR file had the code that would perform local computations. The driver node or master node distributed the JAR file along with the seed, location of the local files, and the number of clusters. Then the worker nodes run the Map-Phase shown in Algorithm 3 to create the partial tree. These partial trees are then sent back to the driver node. In the reduce phase the driver node merges the received partial trees to form the complete tree. After merging, the driver node executes Algorithm 4 to return the centroids. Thus, for this RPHash-TWRP algorithm only a single Map-Reduce step is required to locate the cluster centroids.

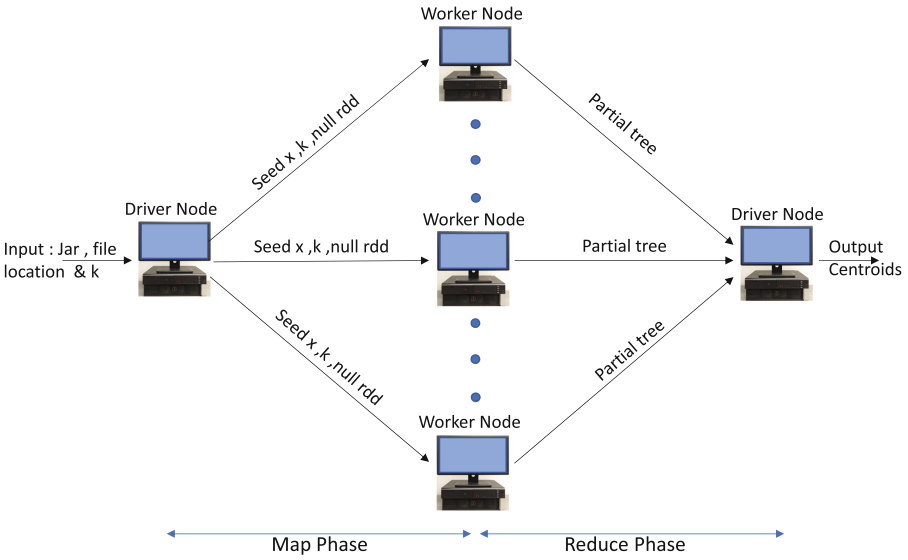

Fig. 1. RPHash-TWRP Map-Reduce implementation architecture on Apache Spark

4.2 Data Security

The important aspect of data security was kept in mind while implementing the algorithms. The algorithms do not transmit any of the original data vectors to remote sites, thereby nullifying any attempts of the attacker who might steal information from the communication channel. It can also be observed from [11] that the probability of getting back the original vector from the projected vector is insignificant. The transferred centroids are just approximations of several data vectors and are not accurate sensitive information that are in the actual data files. This is beneficial for different organizations that do not want to share their own data but get a collective result of the collective data. For example, this

algorithm would preserve the HIPPA [20] privacy rules as no patient information would need to be shared and yet provide, for example, the ability to locate common/frequent behaviors/responses to specific trials across multiple hospitals.

5 Experimental Assessment

The experimental assessment of the RPHash-TWRP algorithm consists of both synthetic (with and without noise) and real world test data. Synthetic test data of 10,000 vectors from dimensions 100 to 7,000 were generated. Each data set has 10 Gaussian clusters with labels recorded for all points. We also generated data sets with varying amount of noise. We chose the dimensionality of 1000 and then injected noise varying from 5 to 40% in increments of 5. To generate them we first generated a dataset with no noise and then we replaced the specified percentage of vectors with randomly generated vectors but preserving the original label. Four datasets of 1,000 dimensions having 300,000, 600,000, 900,000 and 1,200,000 vectors with 5% noise and 20 clusters each were also generated.

We also used five real world datasets to test the performance of RPHash-TWRP. These data sets are all available at the UCI machine learning repository.

1. The *Human Activity Recognition Using Smartphones (HAR)* dataset taken from [7] consists of 10,299 vectors containing 561 features comprising of 6 clusters.
2. The *Smartphone Dataset for Human Activity Recognition (HAR) in Ambient Assisted Living (HARAAL)* dataset [14] has 5,744 vectors and 561 features. It has 6 clusters.
3. The *gene expression cancer RNA-Seq Data Set (RNASEQ)* dataset is taken from [16]. It has 801 vectors and 20,531 features and 4 clusters.
4. The *Smartphone-Based Recognition of Human Activities and Postural Transitions Data Set (HAPT)* dataset taken from [28] has 10,929 vectors and 561 features. This data has 12 clusters.
5. The *Gas Sensor Array Drift (GSAD)* dataset [35] has 13910 vectors and 128 features. It is composed of 6 clusters.

5.1 Algorithms Used for Comparison

We compared the performance our RPHash-TWRP against seven standard well known algorithms.

- *K-Means*: This algorithm of Hartigan and Wong [18] is implemented with the function k-means in R [25].
- Four methods of *Agglomerative Hierarchical clustering*, namely: Single Linkage, Complete Linkage, Average Linkage and Ward's algorithm of minimum variance method [27]. We have used the function `hclust` in R for implementing these algorithms.

- *Self-organizing Tree Algorithm* (SOTA): This algorithm is based on neural network. It is implemented using the 'sota' (Package 'clValid') in R [25].
- The parallel Spark implementation of *K-Means++* available in Spark *MLlib* for the distributed datasets.

These algorithms are selected due of their importance, popularity and availability in R statistical computing framework. These algorithms use FORTRAN, C and C++ subroutines from R to make them run faster. The implementation language of RPHash is Java.

5.2 Hardware Platforms

We captured the run times (in secs) for the scalability study on the synthetic and the real world datasets. They were run on a 16 core Intel(R) Xeon(R) E5-2670 @ 2.6 GHz with 64 GB RAM. For the Spark compute platform, we created the cluster with three machines. Two identical machines composed of an Intel Core(TM)i7-4770 CPU with 4 cores @ 3.40 GHz and having 32 GB memory. The third machine composed of an Intel Core(TM)W3550 CPU with 4 cores @ 3.07 GHz with 16 GB memory.

5.3 Experiment Methodology

Each of the configurations of RPHash-TWRP is tested on all the synthetically generated labeled data sets. The combination of parameters that produces the most consistent and best clustering accuracy on the data set is chosen as the optimal configuration for them for that synthetic dataset. We then chose the most common configuration for the noise test, where we ran RPHash-TWRP on the synthetically generated noise datasets. Finally, We also tested the algorithms on the real world datasets.

This configuration having projected dimension of 16, and offline cluster of Agglomerative clustering for RPHash-TWRP was implemented for the distributed version on spark platform and tested on the same datasets. The datasets were split into approximately three equal parts and kept in the three machines. These partial datasets were local to each machines. Spark was setup in the standalone mode.

5.4 Experimental Results

The clustering accuracy of RPHash is evaluated using 2 external clustering validation measures: Adjusted Rand Index (ARI) and Cluster Purity. We also use the internal measure, WCSS for evaluation. ARI [21] measures the extent to which points from the same ground-truth partition appear in the same cluster, and the extent to which points from different ground-truth partitions are grouped in different clusters. Cluster purity [26] measures how many data points were correctly assigned to its original cluster. WCSS (also called WCSSE or SSE) is

the within cluster sum of squared error. A lower value of WCSS indicates better clustering performance. It is basically the objective function that k-means algorithm tries to minimize in order to find suitable clusters.

Because of the stochastic nature of RPHash-TWRP, we run every configuration of the algorithm 6 times on all data sets and compute the mean and standard deviation of the measures. Similarly, K-means is also run 6 times and means and standard deviation are recorded. The other algorithms are run once as they are deterministic.

The measured ARI, PURITY and WCSS for the synthetic datasets are plotted in Fig. 2. We observed, that, RPHash-TWRP has the value of 1 for ARI and PURITY for all these datasets, indicating perfect clustering. RPHash-TWRP's WCSS also matches the baseline WCSS (*i.e.,* the actual WCSS for a dataset) for these synthetic data.

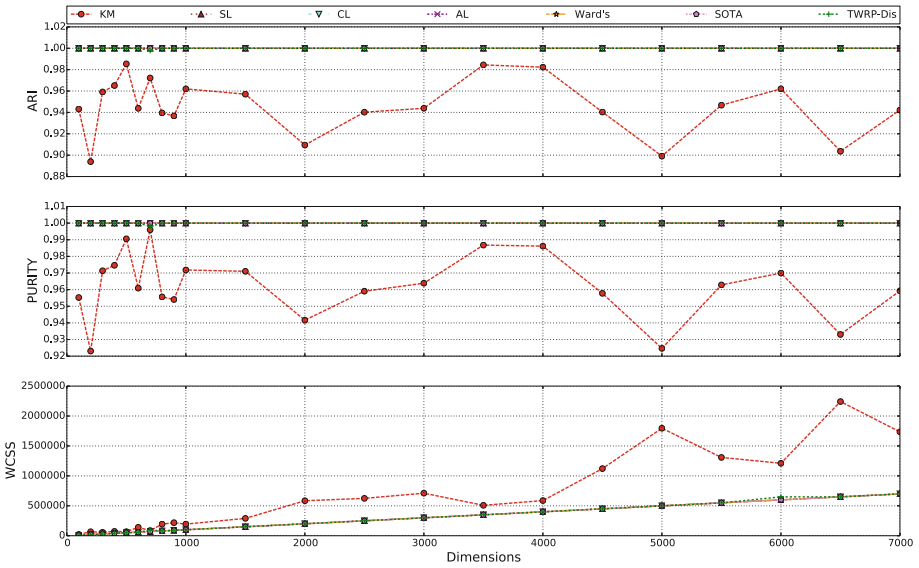

Fig. 2. ARI, Purity and WCSS Plotted Against Increasing dimension. The legend abbreviations for this and all plots are: KM: k-means (Hartigan and Wong), SL: Single Linkage Agglomerative, CL: Complete Linkage Agglomerative, AL: Average Linkage Agglomerative, Wards's: Ward's algorithm, and TWRP-Dis: RPHash-TWRP distributed

For the noise injected data, ARI, and WCSS are plotted in Fig. 3. The WCSS of RPHash-TWRP is lower than single, average and complete linkage and SOTA algorithms. RPHash-TWRP also performs comparably to the other clustering algorithms in terms of ARI. As the noise grows to 40% (when the signal itself is poor) all the algorithms tend to perform poorly.

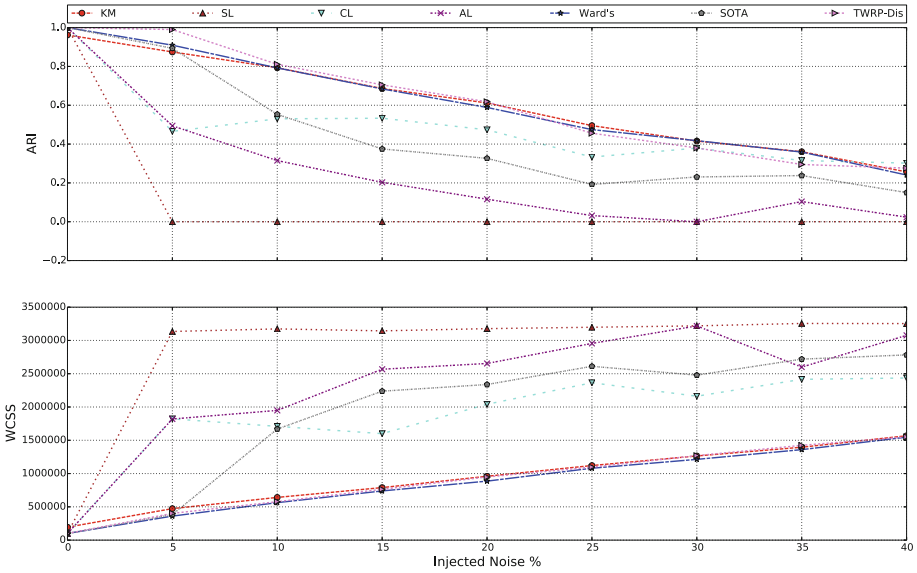

Fig. 3. ARI and WCSS plotted against noise

The results for the real world HAR dataset are summarized in Fig. 4. The results show that RPHash-TWRP, K-means, and Wards algorithm have similar performances. In contrast, single link performs poorly.

The scalability results are shown in Table 1 for the synthetic data sets from dimensions 100 to 7,000. RPHash-TWRP shows very little growth as the dimension increases. The same cannot be said of the other algorithms as we know that their time complexity depends on the data dimensionality. We also tested the run time of our algorithms as we increased the number of vectors keeping the dimension fixed at 1,000 as shown in Fig. 5. We see a linear growth in the run time of RPHash-TWRP and it is more than 10 times faster than the Sparks default K-Means++. Interestingly, in all cases, RPHash-TWRP has higher external measure accuracy than distributed K-Means++.

Table 1. Scalability with respect to dimension.(Run time in secs)

Dim.	100	500	1K	2K	3K	4K	5K	6K	7K
Time	7.54	7.87	8.51	10.73	13.39	13.78	15.14	17.24	19.8

Dataset	Measures	ARI	Purity	WCSS	Time
	k-means	0.4610	0.6002	182 169	66.33
	SL	0.0000	0.1890	556 519	493.95
HAR	CL	0.3270	0.3770	222 044	494.47
	AL	0.3321	0.3588	236 143	494.21
	Ward's	0.4909	0.6597	191 441	494.64
	SOTA	0.3143	0.3966	210 490	23.63
	RPHash-TWRP-Dis	0.4788	0.6407	189 817	7.69
	k-means	0.3988	0.6498	2 498 381	182.90
	SL	0.0003	0.1821	6 023 519	601.98
HAPT	CL	0.0488	0.2505	4 584 352	602.42
	AL	0.0055	0.2046	5 491 388	602.04
	Ward's	0.4033	0.6624	2 617 769	602.68
	SOTA	0.3026	0.3848	2 990 195	31.13
	RPHash-TWRP-Dis	0.3698	0.6119	2 780 331	8.06
	k-means	0.2461	0.4240	1 618 089	23.55
	SL	0.0000	0.1964	3 166 056	148.43
HARAAL	CL	0.0003	0.2002	3 043 579	148.53
	AL	0.0001	0.1972	3 097 976	148.45
	Ward's	0.2764	0.3929	1 653 179	148.58
	SOTA	0.2370	0.3785	1 814 593	12.30
	RPHash-TWRP-Dis	0.2953	0.3887	1 913 092	7.41
	k-means	0.1539	0.4427	27 714 236 160 297	9.05
	SL	0.0000	0.2165	192 076 751 899 323	42.61
Gas-	CL	0.0380	0.3474	47 050 045 285 192	42.57
Sensor	AL	0.0037	0.2865	75 622 509 139 114	42.45
	Ward's	0.2007	0.4378	31 162 058 051 998	42.87
	SOTA	0.0281	0.3435	46 727 103 818 336	11.93
	RPHash-TWRP-Dis	0.1759	0.4426	55 266 567 437 976	6.11
	k-means	0.6438	0.8402	12 834 131	180.32
	SL	0.0007	0.3783	16 007 266	97.10
RNASEQ	CL	−0.0124	0.3758	15 692 260	97.10
	AL	0.0007	0.3783	16 007 266	97.08
	Ward's	0.5955	0.8202	12 916 461	97.09
	SOTA	0.3205	0.6317	13 632 923	87.72
	RPHash-TWRP-Dis	0.4770	0.7416	21 363 363	10.7

Fig. 4. Performance for real data sets

Data Size	Algorithm	ARI	Purity	Time
300K	k-means++parallel spark	0.8134	0.8247	655.0
	RPHash-TWRP	0.997	0.999	58.5
600K	k-means++parallel spark	0.8533	0.8750	1265
	RPHash-TWRP	0.997	0.998	111.0
900K	k-means++parallel spark	0.8809	0.9136	1905
	RPHash-TWRP	0.998	0.999	162.0
1.2M	k-means++parallel spark	0.8496	0.8728	2808
	RPHash-TWRP	0.998	0.999	212.0

Fig. 5. Scalability and measures with respect to size of dataset (seconds)

6 Conclusions

In this work we introduced the distributed version of the Tree-Walk addition to the RPHash algorithm (RPHash-TWRP) for clustering large datasets with log-linear processing complexity. We implemented RPHash-TWRP on the distributed platform Spark using Map-Reduce framework and evaluated it using real world datasets, synthetic data, and synthetic data with variable noise percentage. The results show that RPHash-TWRP performs comparable to various other common standard clustering methods. In tests with synthetic data with noise, we find that our method outperforms other standard implementation of K-Means++ from the spark library MLlib in both run time and accuracy. RPHash-TWRP exhibited more stability when compared to k-Means.

The complexity measures of our method shows that it scales well both in time and space complexity, with very little loss in clustering performance accuracy. The overall scalability is predictable and does not hide large constants. RPHash-TWRP is designed to process very large high dimensional distributed clustering problems while preserving privacy. The results shows that RPHash-TWRP achieves this goal by means of its running time, scalability and accuracy.

References

1. Achlioptas, D.: Database-friendly random projections. In: Proceedings of the 20th Symposium on Principles of Database Systems, pp. 274–281 (2001)
2. Aggarwal, C.C., Yu, P.S. (eds.): Privacy-Preserving Data Mining: Models and Algorithms. Springer, Boston (2008). https://doi.org/10.1007/978-0-387-70992-5
3. Amrani, O., Be'ery, Y.: Efficient bounded-distance decoding of the hexacode and associated decoders for the Leech lattice and the Golay code. IEEE Trans. Commun. **44**(5), 534–537 (1996)
4. Amrani, O., Be'ery, Y., Vardy, A., Sun, F.W., van Tilborg, H.C.A.: The Leech lattice and the Golay code: bounded-distance decoding and multilevel constructions. IEEE Trans. Inf. Theory **40**(4), 1030–1043 (1994)
5. Andoni, A.: Nearest neighbor search: the old, the new, and the impossible. Ph.D. thesis, Massachusetts Institute of Technology, September (2009)
6. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In: 47th Annual IEEE Symposium on Foundations of Computer Science. FOCS 2006, pp. 459–468 (2006)
7. Anguita, D., Ghio, A., Oneto, L., Parra, X., Reyes-Ortiz, J.L.: UCI machine learning repository (2012). https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones
8. Arthur, D., Vassilvitskii, S.: K-means++: the advantages of careful seeding. In: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, pp. 1027–1035. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2007), http://dl.acm.org/citation.cfm?id=1283383.1283494
9. Bellman, R.E. (ed.): Adaptive Control Processes: A Guided Tour. Princeton University Press, Princeton (1961)
10. Cafaro, M., Tempesta, P.: Finding frequent items in parallel. Concurr. Comput.: Pract. Exper. **23**(15), 1774–1788 (2011). https://doi.org/10.1002/cpe.1761

11. Carraher, L.A., Wilsey, P.A., Moitra, A., Dey, S.: Multi-probe random projection clustering to secure very large distributed datasets. In: 2nd International Workshop on Privacy and Security of Big Data, October (2015)
12. Carraher, L.A., Wilsey, P.A., Moitra, A., Dey, S.: Random projection clustering on streaming data. In: IEEE 16th International Conference on Data Mining Workshops (ICDMW), December, pp. 708–715 (2016)
13. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of the twentieth annual symposium on Computational geometry, SCG 2004, pp. 253–262. ACM, New York (2004). https://doi.org/10.1145/997817.997857
14. Davis, K.A., Owusu, E.B.: UCI machine learning repository (2016). https://archive.ics.uci.edu/ml/datasets/Smartphone+Dataset+for+Human+Activity+Recognition+%28HAR%29+in+Ambient+Assisted+Living+%28AAL%29
15. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
16. Fiorini, S.: UCI machine learning repository (2016). https://archive.ics.uci.edu/ml/datasets/gene+expression+cancer+RNA-Seq
17. Franklin, J., Wenke, S., Quasem, S., Carraher, L.A., Wilsey, P.A.: streamingR-PHash: random projection clustering of high-dimensional data in a mapreduce framework. In: IEEE Cluster 2016, September (2016). (poster)
18. Hartigan, J.A., Wong, M.A.: A k-means clustering algorithm. JSTOR: Appl. Stat. **28**(1), 100–108 (1979)
19. Herrero, J., Valencia, A., Dopazo, J.: A hierarchical unsupervised growing neural network for clustering gene expression patterns (2001)
20. Health insurance portability and accountability act (2004). http://www.hhs.gov/ocr/hipaa/
21. Hubert, L., Arabie, P.: Comparing partitions. J. Classif. **2**(1), 193–218 (1985)
22. Jagannathan, G., Pillaipakkamnatt, K., Wright, R.N.: A new privacy-preserving distributed k-clustering algorithm. In: Proceedings of the 2006 SIAM International Conference on Data Mining, pp. 494–498 (2006). https://doi.org/10.1137/1.9781611972764.47
23. Kriegel, H.-P., Kunath, P., Pfeifle, M., Renz, M.: Approximated clustering of distributed high-dimensional data. In: Ho, T.B., Cheung, D., Liu, H. (eds.) PAKDD 2005. LNCS (LNAI), vol. 3518, pp. 432–441. Springer, Heidelberg (2005). https://doi.org/10.1007/11430919_51
24. Liu, B., Xia, Y., Yu, P.S.: Clustering through decision tree construction. In: Proceedings of the Ninth International Conference on Information and Knowledge Management, CIKM 2000, pp. 20–29. ACM, New York (2000). https://doi.org/10.1145/354756.354775
25. Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M., Hornik, K.: cluster: Cluster Analysis Basics and Extensions (2013). (r package version 1.14.4)
26. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, Cambridge (2008)
27. Murtagh, F., Legendre, P.: Ward's hierarchical agglomerative clustering method: which algorithms implement Ward's criterion? J. Classif. **31**(3), 274–295 (2014). https://doi.org/10.1007/s00357-014-9161-z
28. Reyes-Ortiz, J.L., Oneto, L., Sam, A., Parra, X., Anguita, D.: UCI machine learning repository (2015). https://archive.ics.uci.edu/ml/datasets/Smartphone-Based+Recognition+of+Human+Activities+and+Postural+Transitions
29. Sun, F.W., van Tilborg, H.C.A.: The Leech lattice, the octacode, and decoding algorithms. IEEE Trans. Inf. Theory **41**(4), 1097–1106 (1995)

30. Terasawa, K., Tanaka, Y.: Spherical LSH for approximate nearest neighbor search on unit hypersphere. In: WADS, pp. 27–38 (2007)
31. Urruty, T., Djeraba, C., Simovici, D.A.: Clustering by random projections. In: Perner, P. (ed.) ICDM 2007. LNCS (LNAI), vol. 4597, pp. 107–119. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73435-2_9
32. Vaidya, J., Clifton, C.: Privacy preserving association rule mining in vertically partitioned data. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2002, pp. 639–644. ACM, New York (2002). https://doi.org/10.1145/775047.775142
33. Vardy, A.: Even more efficient bounded-distance decoding of the hexacode, the Golay code, and the Leech lattice. IEEE Trans. Inf. Theory **41**(5), 1495–1499 (1995)
34. Vempala, S.S.: The Random Projection Method. DIMACS Series. American Mathematical Society, Providence (2004)
35. Vergara, A., Fonollosa, J., Rodriguez-Lujan, I., Huerta, R.: UCI machine learning repository (2013). https://archive.ics.uci.edu/ml/datasets/Gas+Sensor+Array+Drift+Dataset+at+Different+Concentrations
36. White, T.: Hadoop: The Definitive Guide. O'Reilly Media Inc., Sebastopol (2009)
37. Zaharia, M., et al.: Apache spark: a unified engine for big data processing. Commun. ACM **59**(11), 56–65 (2016). https://doi.org/10.1145/2934664