Decentralized and Adaptive K-Means Clustering for Non-IID Data using HyperLogLog Counters

Amira Soliman[1], Sarunas Girdzijauskas[1], Mohamed-Rafik Bouguelia[2], Sepideh Pashami[2], and Slawomir Nowaczyk[2]

[1] RISE SICS, Sweden.
{aaeh,sarunasg}@kth.se
[2] Halmstad University, Sweden.
firstname.secondname@hh.se

Abstract. The data shared over the Internet tends to originate from ubiquitous and autonomous sources such as mobile phones, fitness trackers, and IoT devices. Centralized and federated machine learning solutions represent the predominant way of providing smart services for users. However, moving data to central location for analysis causes not only many privacy concerns, but also communication overhead. Therefore, in certain situations machine learning models need to be trained in a collaborative and decentralized manner, similar to the way the data is originally generated without requiring any central authority for data or model aggregation. This paper presents a decentralized and adaptive k-means algorithm that clusters data from multiple sources organized in peer-to-peer networks. Our algorithm allows peers to reach an approximation of the global model without sharing any raw data. Most importantly, we address the challenge of decentralized clustering with skewed non-IID data and asynchronous computations by integrating HyperLogLog counters with k-means algorithm. Furthermore, our clustering algorithm allows nodes to individually determine the number of clusters that fits their local data. Results using synthetic and real-world datasets show that our algorithm outperforms state-of-the-art decentralized k-means algorithms achieving accuracy gain that is up-to 36%.

1 Introduction

The predominant way of using machine learning (ML) involves collecting data to a centralized repository often in communication costly and privacy-invasive manner. Therefore, Federated Learning (FL) has been introduced as an alternative distributed and privacy-friendly approach. FL allows users to train models locally on their devices using their sensitive data, and communicate intermediate model updates to a central server without the need to centrally store the data [13]. Specifically, users start by contacting the central server and downloading the learning algorithm and a global model, that is common to all users. The algorithm trains its model locally on each device using user private data and computes update to the current global model. Afterwards, the new updates on the learning parameters obtained from the algorithm on the device of each user are sent to the central server for aggregation. The server integrates the new learning parameters and sends the aggregated global model back to each user. These interactions with the central server are repeated till reaching convergence.

This distributed approach for model computation diminishes the need for central storage of raw data, hence, computation becomes distributed among users and their personal data never leaves their devices.

FL can work very efficiently in many scenarios. The principal advantage of FL is the decoupling of global model training from the need for direct access to the raw data. However, FL has issues that can be related to system and data challenges. Scalability of FL is a major system challenge, especially in use-cases involving a large number of users (e.g., thousands of users) using and improving the global model at the same point. Additionally, data skewness represents one of the main data challenges for FL, since the data is fully distributed and is generated according to behaviours of participating users. Generating a single global model that accumulates all user behaviours might not produce the best model for particular categories of the users. Specifically, global averaging model enforces a bias towards the behavioural patterns provided by the majority of users, while suppressing the patterns of less significant users [20, 21].

It is important for distributed ML and FL to ensure that the training data is uniformly distributed (i.e., IID sampling that represents independent and identical random sampling) so that any resulting model represents unbiased estimate of the expected model parameters. However, with a huge number of users participating in the training of a FL model, there is no control over size and statistical properties of training data used at each device. Thus, it is unrealistic to assume that the data produced by many different users will always be IID data. Specifically, data points generated by users can be quite different as data on each node can be driven using different phenomena. Therefore, two randomly selected users are likely to compute very different updates. This leads to a statistical standpoint where assumptions need to be made for non-IID data [13].

Recently, Peer-to-Peer (P2P) systems have been used as underlying communication frameworks to provide decentralized ML algorithms. The overall system can be thought of as a connected undirected graph with n vertices each representing a node. These nodes can be allowed to communicate randomly with any other node in the network, which shapes the underlying topology to a random graph [5, 18, 22, 12]. Also, the communication among nodes can be restricted to enforce a specific underlying graph topology, for example the communication can be only allowed for friendship ties in social networks or among geographically co-located IoT devices [1, 21, 20].

In this paper, we present a P2P k-means clustering algorithm. The general k-means algorithm takes input as an integer k and a set of m data points with d dimensions in $\mathbb{R}^d$. The goal is to cluster these data points by finding k centers that minimize the sum of the squared distances between each point and the closest center to which it can be assigned to form a cluster [15, 14]. Finding an exact solution to the k-means problem is known to be NP-hard, therefore existing algorithms adopt incremental optimization strategies [14, 4]. Our proposed algorithm extends the general k-means algorithm and allows nodes having distributed data to cooperate in P2P fashion to reach a clustering consensus using their solitary local data and leverage models from others peers.

Our P2P k-means algorithm executes in iterations, such that in each iteration nodes compute an approximation of the new centroids in a decentralized manner by collaboratively exchanging their local estimations and applying weighted averaging. Updating the centroids using weighted averaging function takes into account the number of data points that a node used to calculate its centroids. The more data points used in calculating a centroid, the higher the weight associated with this centroid while applying the averaging function. Differently form existing FL and general P2P k-means algorithms, our proposed algorithm deals with skewed data distributions as well as asynchronous updates of the cluster centroids. We allow nodes to have different pace in executing the exchange iterations, such that some nodes can be more actively engaged than others.

The active nodes can make the system biased toward the properties of their local data. Additionally, these nodes execute exchanges more often which makes their data to be over-represented when applying the weighted averaging function [8]. Naiive weighted averaging function fails to keep track of unique data points represented by a centroid, consequently it keeps accumulating the number of data points owned by active nodes each time they are engaged in an exchanging round. Therefore, our clustering algorithm employs HyperLogLog counters to correctly approximate the total number of data points used in calculating the centroids [6]. HyperLogLog is a probabilistic data structure, which provides a reasonable approximation of cardinality estimation. We integrate HyperLogLog counters so that nodes can keep track of distinct data points used so far in model training, hence allow our clustering algorithm to correctly approximate the number of data points in the network. Therefore, our clustering algorithm can operate under asynchronous computations and prevent model aggregation from being biased towards peers interacting with high frequency.

Decentralized data generation makes imbalanced data and missing classes imperative. The data is expected to be highly skewed due to the heterogeneous nature of participating nodes. A lot of work has been done for solving class imbalance and missing classes using data resampling, however most of these methods require access to the whole data, which is not applicable in decentralized systems [17]. To address these challenges, our clustering algorithm incorporates two different techniques to allow nodes decide the proper number of clusters that fit their local data. Particularly, when two nodes try to merge their local models represented by their centroids, our merging function applies k-means on centroids to group every pair of centroids that are close to each other. Then, our first approach to adaptively fix the number of cluster applies Bradley, Fayyad and Reina (BFR) algorithm to further merge the closest clusters together [19]. We provide another merging function using MinHash algorithm [3].

Our contributions can be described as follows: 1) We provide a decentralized P2P k-means algorithm that can successfully handle ***skewed and non-IID data distribution*** among the participating nodes. 2) We provide a computational environment participating nodes to ***asynchronously compute clustering consensus*** in P2P networks. 3) We provide a novel ***adaptive k-means clustering*** algorithm that allow nodes to individually determine the number of

clusters that fits their local data. 4) We provide experimental evaluation of the proposed decentralized and adaptive k-means algorithm using multiple synthetic as well as real-world dataset. The results show that our algorithm outperforms state-of-the-art decentralized M-Means algorithms achieving accuracy gain that is up to 36%.

The paper is organized as follows: in Section 2 and Section 3, we present an overview of existing centralized, distributed as well as P2P k-means algorithms. Section 4 introduces our proposed methods for P2P Adaptive k-means clustering algorithm. Section 5 shows the experimental evaluation of our proposed algorithm compared to the state-of-the-art P2P k-means algorithms. Finally, Section 6 concludes our paper.

2 Background

Clustering is a technique that is used to partition elements in a dataset such that similar elements are assigned to same cluster while elements with different properties are assigned to different clusters. One of the earliest clustering techniques in the literature is the k-means clustering method [15, 14]. Given a set $\mathbf{X} = \{\boldsymbol{x_1}...\boldsymbol{x_n}\}$ of m samples in $\mathbb{R}^d$, the k-means problem is to find the minimum variance clustering of the dataset into k clusters with centroids C, such that the following potential function is minimized,

$$\phi = \frac{1}{m} \sum_{x \in X} \min_{c \in C} \|x - c\|^2 . \tag{1}$$

Identifying these centroids implicitly defines the cluster to which each sample is assigned. Each data point is mapped to the cluster with the nearest mean, serving as a representation of the cluster. As defined, finding an exact solution to the k-means problem even for $k = 2$ is NP-hard [4].

The k-means algorithm starts by randomly choosing k points in the vector representation space of input data, these points serve as the initial centroids of the clusters. Afterwards, all samples are each assigned to the centroid they are closest to. Then, for each cluster a new centroid is computed by averaging the feature vectors of all samples that are assigned to it. The process of assigning samples and recomputing centroids is repeated until the process converges.

3 Related Work

Several distributed k-means algorithms have been proposed to cluster datasets that are distributed over different locations. These algorithms assume that there is a central coordinator that communicates with all other nodes in the distributed system. The clustering goal is to partition the distributed dataset, into k clusters consistent with the global clustering that can be obtained using the centralized algorithm. Some of these algorithms perform the process of computing the centroids of the clusters in a distributed manner using averaging techniques. The main idea is to generate centroids of local data at each computing node, then transmit them to the central coordinator which computes the average [7, 23]. Other algorithms generate summaries of local data at each node and send them

to the central coordinator to perform the clustering algorithm using the collected summaries [10, 11].

Decentralized clustering on distributed data using P2P has been studied recently. Some solutions introduce distributed k-means algorithms that construct a global set of artificial points to act as a proxy for the entire dataset [1, 16]. There are some solutions that consider P2P random networks and work in static settings, however they are aimed at computing basic average of centroids. Fellus et al. [5] propose a decentralized k-means algorithm which executes in communication rounds, and in each round nodes compute an approximation of the new centroids in a distributed manner.

It is clear that both distributed and decentralized k-means can be efficiently solved using collaborative averaging as well as summarizing techniques. However, the calculation of local approximation only succeed when the data is not skewed. As aforementioned, our main focus is to provide a decentralized P2P k-means method to handle non-IID data as well as asynchronous computations.

4 Decentralized *K*-Means

In P2P k-means, we consider a set of n nodes $V = \{v_i, 1 \leq i \leq n\}$ which can communicate randomly with each other. On each node v_i there is a local set of data points $P_i \subseteq \mathbb{R}^d$, and the global dataset is $P = \bigcup_{i=1}^{n} P_i$. The goal is to find a set of k centers which optimize cost function defined in (1) in a decentralized manner while preserving theoretical guarantees for approximating clustering cost without exchanging the local data among nodes.

4.1 General P2P *K*-Means Algorithms and Their Limitations

In the beginning, we want to emphasize on the limitations of general methods in case of asynchronous scenarios with non-IID data distribution. General P2P k-means methods apply the steps of k-means while using the local data points available at each node after all nodes agree on the set of initial centroids. Then, each node performs the exchange procedure by selecting a random peer form the network to which it sends the computed centroids. We introduce two examples to further explain the consequences of update procedure. First, we consider a P2P network with number of nodes n=3, such that nodes n_1, n_2, and n_3, have data points with sizes equal to 30, 60, 90 data points, respectively. Also, we assume that the number of clusters equals to 3.

Example 1. We assume that every node has data points from the three clusters, where each cluster is represented by one third of the number of points at each node (i.e. n_1 has 10 data points in each cluster, n_2 has 20, etc.). Node n_1 is the only active node to perform exchange iterations. Thus, n_2 and n_3 are going to apply the update operation as described in Algorithm (1). The update function applies simple averaging and treats the centroids generated by n_1 equally with the centroids of other peers, though for example n_3 uses more data points in computing its centroids. Accordingly, if the data owned by n_1 is not representative of the data owned by other peers, n_1 is causing deviation for the general clustering model, though it owns only less than 17% of total data points in the system.

Algorithm 1: Update for k-means at node n_i with centroids from n_j

Local centroids $c_k^{(i)}$: $c_1^{(i)}$, $c_2^{(i)}$, ..., $c_k^{(i)}$

Procedure `Update(`$c_k^{(j)}$`)`

  for $k \leftarrow 1$ **to** K **do**

1 $c_k^{(i)} \leftarrow \frac{1}{2}\left(c_k^{(i)} + c_k^{(j)}\right)$

2 KM-Clustering()

Before illustrating our second example, we want to briefly describe how weighted averaging can be executed in instead of non-weighting averaging. The centroid weight is going to be proportional to the number of data points used to calculate it. In this case, the update function takes an extra input that tells the number of data points belong to each cluster, i.e., nodes exchange the number of data points used to compute each centroid. For further exchange rounds, nodes have to update their counters to keep track of the number of data points used so far in generating the current centroids. Thus, nodes need to continuously accumulate the number of data points used to compute the centroids after every update operation.

Example 2. We assume that n_3 has a missing class, so the data points belong in two clusters not three. However, k-means algorithm splits the points into 3 clusters according to the input $k = 3$. Consider n_3 to be the active node in the first exchange round, so it exchanges its centroids with the number of their associated data points to n_1 and n_2. Then, n_1 and n_2 perform weighted averaging and update their centroids and increase their counters with data points from n_3. In the second exchange round, n_1 and n_2 are engaged together in exchange iteration. Now, when n_1 updates its centroids again using centroids of n_2, number of data points used in weighted averaging is going to reflect what n_3 owns twice, as centroids of n_2 and n_1 both count data points of n_3. Accordingly, data points owned by n_3 are going to be overrepresented, adding to this the fact that its centroids are not correct in representing the three clusters expected in the global model.

4.2 P2P *K*-Means with HyperLogLog Counters

HyperLogLog (HLL) counters are extremely useful for big data as they dramatically decrease the amount of memory needed to approximate the exact cardinality estimation compared to other data structures [6]. HLL counters hash the input data into a bit sequence, while making sure that the hashing function distributes bits as evenly and uniformly as possible in the hashing space. Then, HLL counters encodes the generated hash representation of the input in their bit sequence. Regardless of how many times a particular value appears in the input, it is going to be hashed to the same value, hence encoded only once in the HLL bit sequence. The cardinality is estimated by calculating the maximum number of leading zeros in the binary representation of the generated bit sequence. If the maximum number of leading zeros observed is n, an estimate for the number of distinct elements in the input set is expected to be 2^n.

Algorithm 2: Generate HyperLogLog function for k-means at node n_i

Procedure `GenerateHLL(` $(hll)_k^{(i)}$`)`
  foreach $x \in X_i$ **do**
1 $c \leftarrow clusterID\,(x)$
2 $hll_c^{(i)}.append\,(x)$
3 **for** $k \leftarrow 1$ **to** K **do**
4 **if** $k \neq c$ **then** $hll_k^{(i)}.remove\,(x)$

Interestingly, HLL counters have the property that they can be merged by combining their bit sequences [9], such that generated representation contains the elements encoded in the two HLL counters. In our clustering algorithms we integrate HLL counters to address the limitations of general P2P k-means methods as described in the previous two examples. We start by executing a regular k-means at each node to generate the centroids using the local data. Afterwards, we create a HLL counter per cluster at each node as described in Algorithm (2). The bit sequence of each HLL encodes the hash representation of the data points belonging to that cluster.

Algorithm 3: HyperLogLog update for k-means at node n_i

Procedure `Update(`$c_k^{(j)}$, $(hll)_k^{(j)}$`)`
1 $centroids \leftarrow c_k^{(i)} \cup c_k^{(j)}$; $hll_{all} \leftarrow (hll)_k^{(i)} \cup (hll)_k^{(j)}$
2 $cent\_merge \leftarrow KMeans\,(centroids, k)$ `// identify centroid pairs`
3 $i \leftarrow 0$; $a \leftarrow 0$; $b \leftarrow 0$
4 **for** $m \in cent\_merge$ **do**
  `/* m is a pair indicating centroids to be merged */`
5 $card_x \leftarrow cardinality\,(hll_{all}[m[0]])$; $card_y \leftarrow cardinality\,(hll_{all}[m[1]])$
6 $hll_{un} \leftarrow merge\,(hll_{all}[m[0]], hll_{all}[m[1]])$; $card_{un} \leftarrow cardinality\,(hll_{un})$
7 **if** $card_{un} < (card_x + card_y)$ **then**
8 **if** $card_x > card_y$ **then** $a \leftarrow \frac{card_x}{card_{un}}$; $b \leftarrow 1 - a$
  else $b \leftarrow \frac{card_y}{card_{un}}$; $a \leftarrow 1 - b$
  else $a \leftarrow \frac{card_x}{card_{un}}$; $b \leftarrow \frac{card_y}{card_{un}}$
9 $c_i^{(i)} \leftarrow a \times centroids[m[0]] + b \times centroids[m[1]]$
10 $hll_i^{(i)} \leftarrow hll\_un$; $i \leftarrow i + 1$
11 KM-Clustering()

When nodes get engaged in an exchange round, they communicate their computed centroids as well as HLL counters representing data seen so for in computing the centroids. We use HLL counters to estimate the number of unique data points for each centroid. The update function executes first a regular k-means to find out the centroids to be merged, as shown in Algorithm (3), lines 1:2. We consider applying k-means instead of directly performing weighted averaging procedure as a first step to handle data skewness, such that two local centroids might be closer to each other and better being merged than combing them with worse options computed by other peers. We perform weighted averaging procedure using the estimated cardinalities as shown in lines 5:9. Lastly, nodes

update their previous HLL counters by merging them with the received HLL counters, as described in line 10.

4.3 Adaptive Number of Clusters at Each Node

In our algorithm we provide two functions to adaptively detect the number of clusters at each node. Our first approach applies merging function using MinHash algorithm [3]. MinHash is widely used to estimate how similar two sets of points are. In our clustering algorithm, we create bit sequences similar to ones of HLL counters that encode data points using the MinHash algorithm. Having such bit sequences per cluster, nodes can evaluate how their data points are similar to data points used in other peers to calculate the centroids before performing the merge function. If there is an overlap in the MinHash bit sequences, then the centroids can be merged, otherwise no overlap indicates the clusters are not similar. Interestingly, this indicates that one of the nodes might not have the correct clustering results due to missing classes, and by adopting the centroids from the other node without changing them it can fix the cluster memberships of its local data.

Our second approach is implemeted using BFR algorithm [19] to further merge the closest clusters together. We use BFR algorithm to computes the sum and sum of squares of each cluster in order to compute the standard deviation of points belonging to this cluster. The criterion for further merges can be determined by the gain in terms of cluster variance (i.e. lower value) after combining the data points in one cluster. The variance of merging the two clusters can still be computed using the sum and sum of squares of individual clusters.

5 Evaluation

We proceed with evaluating the performance of proposed clustering algorithm by comparing it with the state-of-the-art P2P k-means algorithms. We have implemented the competitor algorithms according to implementation provided by the original authors using C++. For each method we have used their default settings for the parameters as introduced by each algorithm. To evaluate the clustering accuracy, we have used the F1 score measure that is computed as harmonic mean of precision and recall. Precision reflects mixing of different ground-truth clusters into the extracted ones. Moreover, recall reflects the goodness of grouping nodes that belong to the same ground-truth cluster.

5.1 *K*-Means Clustering Algorithms

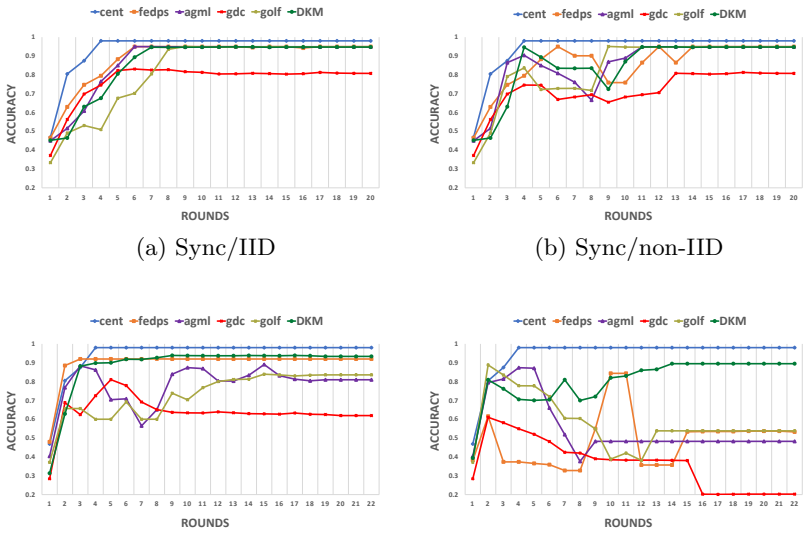
We use the name **DKM** as an identifier of our P2P k-means clustering algorithm. The first method we use for comparison is **cent** that represents a centralized version of k-means algorithm. Then, we have **fedps** that is implemeted as a distributed version of k-means using FL paradigm, such that there is a dedicated centralized node responsible for model aggregation for other nodes in the system. The third method is **agml**, a dicentralized version of P2P k-means that allows nodes to exchage summaries representing their local data, then apply clustering using generated summaries [5]. Also, we implemented **gdc** as a P2P k-means algothim that that allows nodes to generate and exchange a global set of artificial points to act as a proxy for the entire dataset [16]. Finally, we use **golf** that is a P2P k-means implemented using gossip protocol [2].

5.2 Datasets

We have performed the comparison using some real-world as well as synthetic benchmark datasets available from UCI Machine Learning[3] and Fundamental clustering problem suite[4].

Real-world datasets: we use *Daily and Sports Activities* as well as *PAMAP2* datasets. These two datasets comprise motion sensor data of some daily and sports activities each performed by different persons in their own style. Our objective is to cluster these activities into 3 categories: 1) low intense activities such as sitting and standing, 2) moderate activities such as walking or running on a treadmill, and 3) intense activities such as rowing and cycling. The first dataset contains 9,120 data points, while the second one contains 27,582 data points. We used 2D representation of the data.

Synthetic datasets: we use *Energy Time* and *S1* datasets that are generating as Gaussian clusters. The first dataset contains 4,096 data points into 2 Gaussian clusters. The second dataset has 5,000 data points clustered into 15 Gaussian clusters.



(a) Sync/IID (b) Sync/non-IID

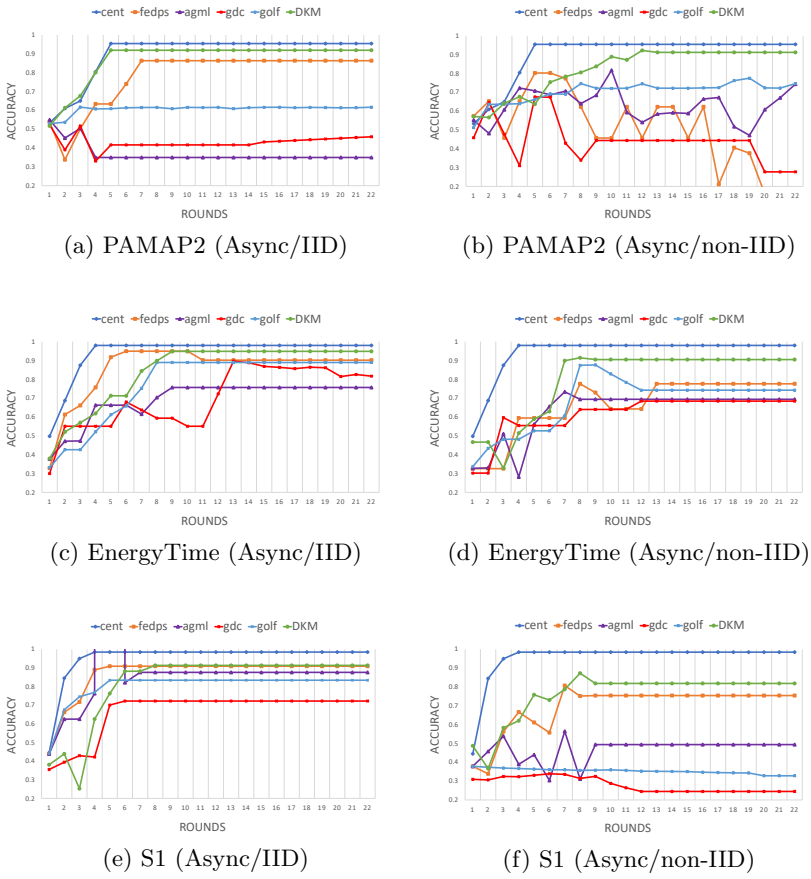
(c) Async/IID (d) Async/non-IID

Fig. 1: Evaluation using Daily and Sports Activities dataset.

5.3 Skewed Data and Asynchronous Computations

For the following experiments, we create a P2P network with 100 nodes. Each node has its own local data repository and can communicate with any random subset of peers in the network. For IID test cases, we evenly distribute the training sets among the peers. Also, we distribute the data in non-IID manner,

[3] https://archive.ics.uci.edu/ml/datasets.php?format=&task=clu
[4] https://www.uni-marburg.de/fb12/arbeitsgruppen/datenbionik/data



(a) PAMAP2 (Async/IID)

(b) PAMAP2 (Async/non-IID)

(c) EnergyTime (Async/IID)

(d) EnergyTime (Async/non-IID)

(e) S1 (Async/IID)

(f) S1 (Async/non-IID)

Fig. 2: Evaluation using PAMAP2, EnergyTime, and S1 datasets in the cases of asynchronous computations.

such that allow one third of the nodes to obtain 50% of the data points per each cluster, whereas the remaining data points are distributed randomly among the remaining nodes.

We also create some highly unbalanced distribution, in which one third of the nodes in the network have missing classes among their allocated data points. Additionally, we create asynchronous computation scenario by assign nodes different speed to perform the exchange rounds. We split the network randomly in three parts, the first part remains idle, the second part performs only one exchange per computation round, while the last set are actively participating by performing up to three exchanges in one round.

Figure 1 shows the evaluation of different P2P methods using the first dataset. As show, we report the accuracy in different use-cases: first (a) when we have IID data distribution and synchronous computation when all nodes have the same exchange pace. Then (b) when data becomes non-IID distributed. In (c) and

(d) cases, we report the accuracy in case of asynchronous computations when data is distributed in IID and non-IID manner. The results confirm that general P2P methods work when data is uniformly distributed and nodes update their centroids with the same frequencies.

Figure 2 reports the results of the remaining datasets in asynchronous computation scenarios. Results using PAMAP2, EnergyTime, and S1 confirm that our algorithm (DKM) is the only method capable of achieving accuracy comparable to the centralized version when we explore the expected real-world case scenarios of having non-IID data and asynchronous computations, while other methods fail to achieve acceptable accuracy.

6 Conclusion

This paper presents a novel decentralized as well as adaptive k-means clustering algorithm that is highly beneficial for dynamic and fully distributed environments. Our main contribution is to provide a decentralized k-means method for skewed data distribution and asynchronous computations in P2P networks. We integrate HyperLogLog counters with our k-means algorithm to efficiently handle data skewness in such dynamic execution environment. Furthermore, our clustering algorithm allows nodes to individually determine the number of clusters that fits their local data. Our experimental evaluation confirms the ability of our algorithm to adapt to difficult scenarios in which existing P2P k-means methods fail to generate acceptable results.

Acknowledgements
This research has been conducted within the "BIDAF: A Big Data Analytics Framework for a Smart Society"[5] project funded by the Swedish Knowledge Foundation.

References

1. Maria-Florina F Balcan, Steven Ehrlich, and Yingyu Liang. Distributed k-means and k-median clustering on general topologies. In *Advances in Neural Information Processing Systems*, pages 1995–2003, 2013.
2. Árpád Berta, István Hegedűs, and Róbert Ormándi. Lightning fast asynchronous distributed k-means clustering. 2014.
3. Andrei Z Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*, pages 21–29. IEEE, 1997.
4. Petros Drineas, Alan Frieze, Ravi Kannan, Santosh Vempala, and V Vinay. Clustering large graphs via the singular value decomposition. *Machine learning*, 56(1-3):9–33, 2004.
5. Jerome Fellus, David Picard, and Philippe-Henri Gosselin. Decentralized k-means using randomized gossip protocols for clustering large datasets. In *2013 IEEE 13th International Conference on Data Mining Workshops*, pages 599–606. IEEE, 2013.
6. Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *Discrete Mathematics and Theoretical Computer Science*, pages 137–156. Discrete Mathematics and Theoretical Computer Science, 2007.

[5] http://bidaf.sics.se/

7. George Forman and Bin Zhang. Distributed data clustering can be efficient and exact. *SIGKDD explorations*, 2(2):34–38, 2000.
8. Lodovico Giaretta and Šarunas Girdzijauskas. Gossip learning: Off the beaten path. In *2019 IEEE International Conference on Big Data (IEEE Big Data 2019), December 9-12, 2019, Los Angeles, CA, USA*, 2019.
9. Stefan Heule, Marc Nunkesser, and Alexander Hall. Hyperloglog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 683–692. ACM, 2013.
10. Eshref Januzaj, Hans-Peter Kriegel, and Martin Pfeifle. Towards effective and efficient distributed clustering. In *Workshop on Clustering Large Data Sets (ICDM2003)*, 2003.
11. Hillol Kargupta, Weiyun Huang, Krishnamoorthy Sivakumar, and Erik Johnson. Distributed clustering using collective principal component analysis. *Knowledge and Information Systems*, 3(4):422–448, 2001.
12. Mansour Khelghatdoust and Sarunas Girdzijauskas. Short: Gossip-based sampling in social overlays. In *International Conference on Networked Systems*, pages 335–340. Springer, 2014.
13. Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.
14. Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.
15. James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.
16. Hoda Mashayekhi, Jafar Habibi, Tania Khalafbeigi, Spyros Voulgaris, and Maarten Van Steen. Gdcluster: a general decentralized clustering algorithm. *IEEE transactions on knowledge and data engineering*, 27(7):1892–1905, 2015.
17. Giang Hoang Nguyen, Abdesselam Bouzerdoum, and Son Lam Phung. Learning pattern classification tasks with imbalanced data sets. In *Pattern recognition*. IntechOpen, 2009.
18. Róbert Ormándi, István Hegedűs, and Márk Jelasity. Gossip learning with linear models on fully distributed data. *Concurrency and Computation: Practice and Experience*, 25(4):556–571, 2013.
19. Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.
20. Amira Soliman, Leila Bahri, Barbara Carminati, Elena Ferrari, and Sarunas Girdzijauskas. Diva: Decentralized identity validation for social networks. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*, pages 383–391. ACM, 2015.
21. Amira Soliman, Leila Bahri, Sarunas Girdzijauskas, Barbara Carminati, and Elena Ferrari. Cadiva: cooperative and adaptive decentralized identity validation model for social networks. *Social Network Analysis and Mining*, 6(1):36, Jun 2016.
22. Amira Soliman and Sarunas Girdzijauskas. Dlsas: Distributed large-scale anti-spam framework for decentralized online social networks. In *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, pages 363–372. IEEE, 2016.
23. Dimitris K Tasoulis and Michael N Vrahatis. Unsupervised distributed clustering. In *Parallel and distributed computing and networks*, pages 347–351, 2004.