ESANN 2014 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 23-25 April 2014, i6doc.com publ., ISBN 978-287419095-7.
Available from http://www.i6doc.com/fr/livre/?GCOI=28001100432440.

Lightning Fast Asynchronous Distributed K-Means Clustering*

Árpád Berta, István Hegedűs and Róbert Ormándi

University of Szeged, Szeged, Hungary
{berta,ihegedus,ormandi}@inf.u-szeged.hu

Abstract. One of the most fundamental data processing approach is the clustering. This is even true in distributed architectures. Here, we focus on the problem of designing efficient and fast K-Means approaches which work in fully distributed, asynchronous networks without any central control. We assume that the network has a huge number of computational units (even orders of magnitude more than the number of computational units in a general cloud). Our approaches apply online learning clustering models which take different random walks in the network, while they update themselves using the data points stored by the computational units, and various ensemble techniques combine them to get a faster convergence. We define different instantiations of the general framework that apply various ensemble techniques. We evaluate them empirically against several state-of-the-art distributed baseline algorithms in different computational scenarios. The experiments show that our methods are not only robust against network failure, but they also provide accurate clustering and converge extremely fast.

1 Introduction

The continuously increasing amount of data—which we want to access from any of our computational devices (mobiles, tablets, and PCs), share and comment—raises the need for novel distributed architectures. Typical examples are the Map-Reduce [1] based approaches as general purpose cloud computing frameworks and the GraphLab [2] for performing distributed machine learning.

Although these architectures scale very well, their scalability still has a strict upper-bound, which could be a serious issue considering a long-term time horizon with the exponential spreading of different mobile and wearable computers in the near future. Additionally, the application of these solutions inherently suffers from privacy issues due to the fact that the data leaves the device of the data owner.

Here, we introduce a novel system model that uses a huge number of personal computational units without additional elements (that is, background infrastructure is not required). We show that with a careful design (considering both the algorithmic and the system aspects of the design space), very efficient algorithms can be proposed on the top of that system model. Particularly, we design efficient K-MEANS algorithms in the fully distributed system model (described in Sec. 2).

Our particular contribution is twofold. First, we implement the well-known K-MEANS clustering algorithm in the Gossip Learning Framework (GOLF) [3]. Second, we propose two ensemble learning based algorithm variants which dramatically speed up the convergence of the original algorithm. Finally, we provide a thorough empirical evaluation of the proposed algorithms using different baseline approaches.

*This work was supported by the European Union and the European Social Fund through project FuturICT.hu (grant no.: TAMOP-4.2.2.C-11/1/KONV-2012-0013).

2 Background and System Model

We considered the fully distributed, asynchronous networks as our system model. This model is significantly different from the system models of the sensor networks, the GRIDs, the clouds and even the P2P networks, where any kind of synchronization (even heart beat synchronization) is allowed.

We assume that our system model consists of a very large number—even billions—of *nodes* (computational units or peers, which are typically PCs, mobiles or smart devices). We also assume that these nodes are connected via a network (Internet) and each of them has a unique *address*. The nodes can communicate with a limited number of other nodes (called *neighbors*), whose address is known locally, via *messaging*. The set of neighbors of a node can change over the time (dynamic network). The peers as graph nodes and the neighborhood relation defined above as edges define a graph called *overlay network*. All the communication is performed along the edges of this graph.

We assume that a service, called *peer sampling service*, exists which is responsible for managing the overlay network. We require that this service provides the addresses of uniform random nodes from the network that are probably online at the time of the request. We applied the NEWSCAST protocol [4] as the peer sampling service.

We do not make any assumption about the reliability of the network. That is, messages can have arbitrary delay or even can be lost. Additionally, we do not make any synchronicity assumption, meaning that we cannot perform any global operations regarding the processing time (e.g. waiting until each node ends some kind of activity). Broadcasting of the messages is not allowed as well. The nodes can join and leave the network at any time without prior notification (churn). Moreover, there is no network infrastructure in the sense that each node has an equal role and they apply exactly the same protocol.

As for data model, we assume that each node stores its local data and the amount of this data is insufficient to perform any local statistical processing. Moreover, we assume that this data never leaves the node that stores it. This restriction clearly allows us to support applications that require extreme privacy. We also assume that the clustering of new data points can be performed locally; that is, the clustering models have to be available in the nodes at any time.

3 Related Work

In [5], a distributed clustering method was presented, where each node contains only one data point. This data is used for the local initialization of the clustering models. Each node iteratively receives centroids that are added to a local model, then in the oversized models it merges the closest centroids. Since this approach is consistent with our data model, we consider it as a baseline method in our evaluations and refered to it as GREEDY-MATCHING-K-MEANS.

Considering the P2P environments, one of the first K-MEANS clustering was described in [6]. Here, the same randomly initialized centroids were set for each node. Then, the algorithm exchanges and aggregates cluster assignments between the direct neighbors. These operations require local synchronization, hence this approach is unsuitable for our system model.

Another P2P K-MEANS approach was proposed in [7]. The algorithm iteratively performs local assignment and global centroid recomputation using the PUSH-SUM [8] averaging algorithm. As this protocol is consistent with our data and system models,

Algorithm 1 The GoLF Learning Scheme

```
1: initModel()
2: loop
3:     wait($\Delta$)
4:     $peer \leftarrow$ getRandomPeer()
5:     send $cModel$ to $peer$

6: function onReceiveModel($m$)
7:     $cModel \leftarrow$ createModel($m, pModel$)
8:     $pModel \leftarrow m$
```

Algorithm 2 Simple-K-Means algorithm

```
1: function update($m$)
2:     $x \leftarrow$ getLocalData()
3:     if $m$.hasEmpty() then
4:         $m$.addCentroid($x$)
5:     else
6:         $idx \leftarrow$ performClust($m, x$)
7:         $c \leftarrow m.centroids(idx)$
8:         $m.centroids(idx) \leftarrow$
               $(1 - m.\alpha) \cdot c + m.\alpha \cdot x$
9:     return $m$

10: function performClust($m, x$)
11:     $minDist \leftarrow \infty$
12:     for $c \in m.centroids$ do
13:         $dist \leftarrow$ euclideanDist($c, x$)
14:         $minDist \leftarrow$ min($dist, minDist$)
15:     return $minDist.centroid.index$

16: function createModelNaive($m_1, m_2$)
17:     return update($m_1$)                    ▷ $m_2$ is not used here
```

we use it as a baseline method (referred to as FTD-K-Means) in our evaluations.

4 Algorithms

The basic skeleton of GoLF is shown in Alg. 1. This algorithm runs on each node of the network. The protocol has two distinct parts, namely the *active part* (in lines 1-5) and the *message handler part* (the function onReceiveModel).

The active part is mainly responsible for performing the periodic activities. First, in the initialization (function initModel) the value of the current model (variable $cModel$) is initialized either by loading the previous value or—on the first time run—initializing a new one. In the main loop of the algorithm the protocol periodically, denoted by Δ (line 3), selects a random neighbor (line 4) using the peer sampling service, and sends a copy of its current model to that neighbor (line 5). We modeled the waiting interval as a random variable $\mathcal{N}(\mu, \sigma)$, where $\mu = \Delta$ and $\sigma = \Delta/10$.

The message handler function of GoLF runs when a node receives a model. In this function the received model is passed through the abstract function CreateModel together with the previously received, non-updated model referred to as $pModel$ (line 7). The resulting new model is then stored in the variable $cModel$. The various implementations of the function CreateModel result different approaches of the distributed K-Means algorithm. Finally the received model m is stored in variable $pModel$ for further processing (line 8).

The function update (Alg 2) is responsible for improving the online clustering model using the data point available on the current node. While the model is not fully initialized, it collects the data point as centroid (at line 4). When each centroid of the model m becomes valid—that is, the model is fully initialized—the function update improves the closest centroid c to the given data point x by computing the moving average of the centroid c and the point x applying the parameter α of the model as weight (between lines 6-8). Here α is the coefficient of the moving average which determines how fast the older data points are discounted. In our experiments, we set $\alpha = 1/125$.

The clustering function performClust, shown in Alg 2, simply finds the index of the closest centroid from the model to the data point by applying Euclidean distance.

Now we describe how these functions can be used in the above mentioned GoLF skeleton. One can get our simplest K-Means variant, referred to as Simple-K-

Algorithm 3 ENSEMBLE-K-MEANS algorithms

```
 1: function CREATEMODELENS($m_1, m_2$)
 2:     return update(merge($m_1, m_2$))

 3: function MERGE($m_1, m_2$)
 4:     $m \leftarrow$ createEmptyModel()
 5:     if $m_1$.hasEmpty() or $m_2$.hasEmpty() then
 6:         $u \leftarrow m_1.centroids \cup m_2.centroids$
 7:         $m.centroids \leftarrow$ shrink($m.K, u$)
 8:     else
 9:         $\pi \leftarrow$ match($m_1, m_2$)
10:         for $i \in centroidindices$ do
11:             $c_1 \leftarrow m_1.centroids(i)$
12:             $c_2 \leftarrow m_2.centroids(\pi(i))$
13:             $m.centroids(idx) \leftarrow$ avg($c_1, c_2$)
14:     return $m$
```

MEANS algorithm, by instantiating the CREATEMODEL function of the GOLF as the function CREATEMODELNAIVE, as shown in Alg 2. This implementation is quite straightforward and ignores the ensemble component of GOLF by calling directly the function UPDATE with the first model leaving out the second model parameter.

Implementing the function CREATEMODEL of GOLF, as proposed in Alg. 3 (called CREATEMODELENS), we get more sophisticated distributed K-MEANS variants that exploit the benefits of the ensemble component of GOLF. First, the implementation applies a function MERGE, which is responsible for combining two K-MEANS models into one. After combining the models, the function UPDATE of K-MEANS (see Alg. 2) is applied on the resulting model.The function MERGE is shown in Alg. 3 as well. If either of the models is not initialized (condition at line 5) the function MERGE collects the centroids by unifying the centroids of the two original models (at line 6). Then, the resulting centroid set is shrunk to a K–sized set by randomly removing centroids (at line 7). If both parameter models are initialized—that is, they store exactly K valid centroids—the function MERGE performs an assignment among the centroids by calling the function MATCH (line 9). This assignment is encoded by a permutation of the centroid indices from model m_2. The function MERGE produces a new model that contains centroids, which are the averages of the centroids of the original models along the assignment (see lines 9-13 of Alg. 3).

By implementing various MATCH functions, one can get different ensemble based fully distributed K-MEANS algorithms. Here we implemented two approaches. The first one is the so-called IDENTITY-MATCHING-K-MEANS, which simply applies the identity mapping as a matching. This is clearly a naive approach, as it does not consider any similarities among the centroids of the two models.

We implemented a more sophisticated matching function as well that solves the assignment problem among the centroids of the two models analytically. That is, it finds a coupling among the centroids of the two models that minimizes the sum of the distances between the coupled centroids. This problem can be solved efficiently by applying the Hungarian method [9]. Here this algorithm is called HUNGARIAN-MATCHING-K-MEANS.

5 Experimental Evaluation

In our experiments we applied various K-MEANS implementations. Accordingly, each of these methods converges to the same optimal solution (aside from the variance of the empirical evaluation). In this way, the only factor that we can examine in terms of performance is the *speed of the convergence*. The algorithms used as baselines were the following: WEKA K-MEANS [10], SIMPLE-K-MEANS (introduced in Sec. 4), GREEDY-MATCHING-K-MEANS [5] and FTD-K-MEANS [7].

In our evaluations, we used three well-known benchmark datasets, namely the Reuters (Dexter), the Spambase, and the Malicious URL's datasets taken from the UCI repository [11]. In the case of Malicious URL's dataset, we applied a feature selection by

ESANN 2014 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 23-25 April 2014, i6doc.com publ., ISBN 978-287419095-7. Available from http://www.i6doc.com/fr/livre/?GCOI=28001100432440.

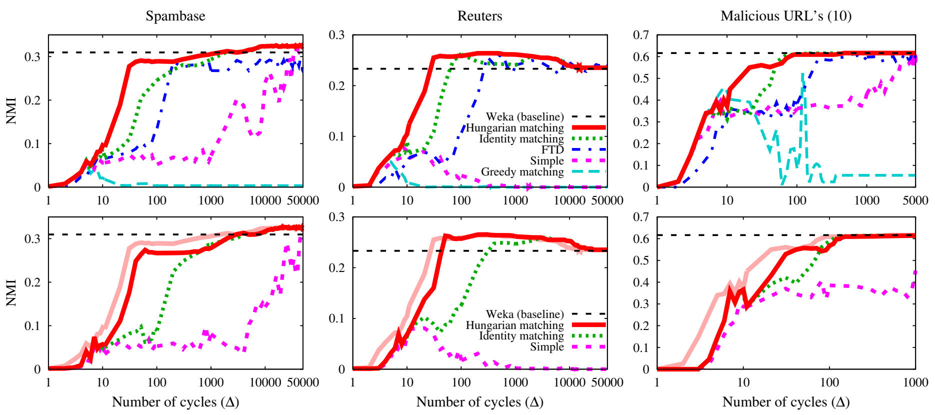

Fig. 1: Comparison of various K-MEANS results in non-failure (upper row) and in failure (lower row) scenarios. We also present the results of the HUNGARIAN-MATCHING-K-MEANS from non-failure scenario with lighter color in the lower row.

keeping the ten most informative features. As an evaluation metric we applied the *Normalized Mutual Information (NMI)* [12] performance measure.

The experiments were carried out in the event based engine of the PeerSim simulator [13]. In each experiment the size of the network was equal to the size of the used dataset. In our experimental scenarios we modeled message drop, message delay, and churn. The drop probability of each message was 0.5. Message delay was modeled as a uniform random delay from the interval $[\Delta, 4\Delta]$, where Δ is the gossip period, as shown in Algorithm 1. We also modeled realistic churn provided by a probabilistic model originally proposed in [14].

5.1 Results

We summarize our main results in Fig 1. Let us first discuss the "No failure" scenarios. As can be seen, the proposed algorithms (IDENTITY-MATCHING-K-MEANS and HUNGARIAN-MATCHING-K-MEANS) and one of the baseline algorithms (the FTD-K-MEANS) converge in the cases of all the benchmark datasets. We also see that the HUNGARIAN-MATCHING-K-MEANS algorithm has by far the fastest convergence speed (the x axis is logarithmic), followed by IDENTITY-MATCHING-K-MEANS in terms of the convergence speed, and FTD-K-MEANS comes third. Surprisingly, the SIMPLE-K-MEANS algorithm does not converge in one case out of the three, and the GREEDY-MATCHING-K-MEANS does not converge at all. Moreover, the convergence speed of the SIMPLE-K-MEANS algorithm is far from being acceptable. Based on these, we can draw several conclusions. First, the HUNGARIAN-MATCHING-K-MEANS, IDENTITY-MATCHING-K-MEANS and FTD-K-MEANS algorithms seem to be the most promising (in decreasing order based on their convergence speed) in our system and data model. Second, the Reuters dataset, which has by far the highest dimensionality in the benchmark datasets, seems to provide the most complex learning task, since two of the baseline algorithms do not converge on it. Third, nevertheless, the GREEDY-MATCHING-K-MEANS algorithm fits into our data model, it seems that our system model makes it an inappropriate choice.

Turning to the discussion of the "All failures" scenarios, we should mention that two of our baseline algorithms (GREEDY-MATCHING-K-MEANS and FTD-K-MEANS) are inappropriate in this setting, hence we ignored them. In the lower row of Fig. 1, we see the results corresponding to the failure scenarios. The main observations here are that our proposals achieve almost the same convergence speed as that in the case of non-failure scenarios (the light red line denotes the convergence speed of the HUNGARIAN-MATCHING-K-MEANS algorithm in that setting). The HUNGARIAN-MATCHING-K-MEANS remains the fastest algorithm, while the IDENTITY-MATCHING-K-MEANS is the second fastest.

6 Conclusion

In this paper, we proposed two instantiations of the well known and widely used K-MEANS clustering algorithm that work in fully distributed environments. With the presented methods the nodes in a P2P network can continuously improve the performance of the models by local updates, and the models are available for clustering at any time without extra communication. Our experiments clearly showed that the proposed methods can achieve the expected clustering performance and converge extremely fast. Comparing with various baseline algorithms, the HUNGARIAN-MATCHING-K-MEANS method achieved the fastest convergence speed on real datasets even in environments with extreme network failures.

References

[1] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[2] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. Graphlab: A new framework for parallel machine learning. *arXiv:1006.4990*, 2010.

[3] R. Ormándi, I. Hegedűs, and M. Jelasity. Gossip learning with linear models on fully distributed data. *Concurrency and Computation: Practice and Experience*, 25(4):556–571, 2012.

[4] N. Tölgyesi and M. Jelasity. Adaptive peer sampling with newscast. In *Euro-Par 2009*, volume 5704 of *LNCS*, pages 523–534. Springer, 2009.

[5] I. Eyal, I. Keidar, and R. Rom. Distributed data clustering in sensor networks. *Distributed Computing*, 24(5):207–222, 2011.

[6] S. Bandyopadhyay, C. Giannella, U. Maulik, H. Kargupta, Kun Liu, and S. Datta. Clustering distributed data streams in peer-to-peer environments. *Inf. Sci.*, 176(14):1952–1985, 2006.

[7] G. Di Fatta, F. Blasa, S. Cafiero, and G. Fortino. Fault tolerant decentralised k-means clustering for asynchronous large-scale networks. *J. Par. Distr. Comp.*, 73(3):317–329, 2013.

[8] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Proc. 44th FOCS'03*, pages 482–491. IEEE Computer Society, 2003.

[9] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[10] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of 5th Berkeley Symposium on Math. Stat. and Prob.*, pages 281–297, 1967.

[11] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[12] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[13] A. Montresor and M. Jelasity. Peersim: A scalable P2P simulator. In *Proc. 9th IEEE Intl. Conf. on Peer-to-Peer Comp.*, pages 99–100. IEEE, 2009. extended abstract.

[14] D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. In *Proc. 6th ACM Conf. on Internet measurement (IMC'06)*, pages 189–202. ACM, 2006.