Distrib. Comput.
DOI 10.1007/s00446-011-0143-7

Distributed data clustering in sensor networks

Ittay Eyal · Idit Keidar · Raphael Rom

Received: 27 February 2011 / Accepted: 12 October 2011

Abstract Low overhead analysis of large distributed data sets is necessary for current data centers and for future sensor networks. In such systems, each node holds some data value, e.g., a local sensor read, and a concise picture of the global system state needs to be obtained. In resource-constrained environments like sensor networks, this needs to be done without collecting all the data at any location, i.e., in a *distributed* manner. To this end, we address the *distributed clustering problem*, in which numerous interconnected nodes compute a *clustering* of their data, i.e., partition these values into multiple clusters, and describe each cluster concisely. We present a *generic algorithm* that solves the distributed clustering problem and may be implemented in various topologies, using different clustering types. For example, the generic algorithm can be instantiated to cluster values according to distance, targeting the same problem as the famous k-means clustering algorithm. However, the distance criterion is often not sufficient to provide good clustering results. We present an instantiation of the generic algorithm that describes the values as a *Gaussian Mixture* (a set of weighted normal distributions), and uses machine learning tools for clustering decisions. Simulations show the robustness, speed and scalability of this algorithm. We prove that any implementation of the generic algorithm converges over any connected topology, clustering criterion and cluster representation, in fully asynchronous settings.

Keywords Sensor networks · Distributed clustering · Robust aggregation

1 Introduction

To analyze large data sets, it is common practice to employ *clustering* [6]: In clustering, the data values are *partitioned* into several *clusters*, and each cluster is described concisely using a *summary*. This classical problem in machine learning is solved using various heuristic techniques, which typically base their decisions on a view of the complete data set, stored in some central database.

However, it is sometimes necessary to perform clustering on data sets that are distributed among a large number of nodes. For example, in a grid computing system, load balancing can be implemented by having heavily loaded machines stop serving new requests. But this requires analysis of the load of all machines. If, e.g., half the machines have a load of about 10%, and the other half is 90% utilized, the system's state can be summarized by partitioning the machines into two clusters—lightly loaded and heavily loaded. A machine with 60% load is associated with the heavily loaded cluster, and should stop taking new requests. But, if the cluster averages were instead 50 and 80%, it would have been associated with the former, i.e., lightly loaded, and would keep serving new requests. Another scenario is that of sensor networks with thousands of nodes monitoring conditions like seismic activity or temperature [1,21].

In both of these examples, there are strict constraints on the resources devoted to the clustering mechanism. Large-scale

A preliminary version of this paper appears in the proceedings of the 29th Symposium on Principles of Distributed Computing (PODC) [9].

I. Eyal (✉) · I. Keidar · R. Rom
Department of Electrical Engineering,
The Technion—Israel Institute of Technology,
32000 Technion city, Haifa, Israel
e-mail: ittay@tx.technion.ac.il

I. Keidar
e-mail: idish@ee.technion.ac.il

R. Rom
e-mail: rom@ee.technion.ac.il

Published online: 01 November 2011

computation clouds allot only limited resources to monitoring, so as not to interfere with their main operation, and sensor networks use lightweight nodes with minimal hardware. These constraints render the collection of all data at a central location infeasible, and therefore rule out the use of centralized clustering algorithms.

In this paper, we address the problem of *distributed clustering*. A detailed account of previous work appears in Sect. 2, and a formal definition of the problem appears in Sect. 3.

A solution to distributed clustering ought to summarize data within the network. There exist distributed algorithms that calculate scalar aggregates, such as sum and average, of the entire data set [14, 10]. In contrast, a clustering algorithm must partition the data into clusters, and summarize each cluster separately. In this case, it seems like we are facing a Catch-22 [13]: Had the nodes had the summaries, they would have been able to partition the values by associating each one with the summary it fits best. Alternatively, if each value was labeled a cluster identifier, it would have been possible to distributively calculate the summary of each cluster separately, using the aforementioned aggregation algorithms.

In Sect. 4 we present a generic distributed clustering algorithm to solve this predicament. In our algorithm, all nodes obtain a clustering of the complete data set without actually hearing all the data values. The double bind described above is overcome by implementing adaptive compression: A clustering can be seen as a lossy compression of the data, where a cluster of similar values can be described succinctly, whereas a concise summary of dissimilar values loses a lot of information. Our algorithm tries to distribute the values between the nodes. At the beginning, it uses minimal compression, since each node has only little information to store and send. Once a significant amount of information is obtained, a node may perform efficient compression, joining only similar values.

Our algorithm captures a large family of algorithms that solve various instantiations of the problem—with different approaches, clustering values from any multidimensional domain and with different data distributions, using various summary representations, and running on arbitrary connected topologies. A common approach to clustering is k-means, where each cluster is summarized by its *centroid* (average of the values in the cluster), and partitioning is based on distance. A k-means approach is a possible implementation of our generic algorithm. The result of this implementation, however, would differ from that of the classical centralized k-means algorithm.

Since the summary of clusters as centroids is often insufficient in real life, machine learning solutions typically also take the variance into account, and summarize values as a weighted set of Gaussians (normal distributions), which is called a *Gaussian Mixture (GM)* [20]. In Sect. 5, we present a novel distributed clustering algorithm that employs this approach, also as an instance of our generic algorithm. The GM algorithm makes clustering decisions using a popular machine learning heuristic, *Expectation Maximization (EM)* [5]. We present in Sect. 5.2 simulation results demonstrating the effectiveness of this approach. These results show that the algorithm converges with high speed. It can provide a rich description of multidimensional data sets. Additionally, it can detect and remove outlying erroneous values, thereby enabling robust calculation of the average.

The centroids and GM algorithms are but two examples of our generic algorithm; in all instances, nodes independently strive to estimate the clustering of the data. This raises a question that has not been dealt with before: does this process converge? One of the main contributions of this paper, presented in Sect. 6, is a formal proof that indeed any implementation of our generic algorithm converges, s.t. all nodes in the system learn *the same* clustering of the complete data set. We prove that convergence is ensured under a broad set of circumstances: arbitrary asynchrony, an arbitrary connected topology, and no assumptions on the distribution of the values.

Note that in the abstract settings of the generic algorithm, there is no sense in defining the destination clustering the algorithm converges to precisely, or in arguing about its quality, since these are application-specific and usually heuristic in nature. Additionally, due to asynchrony and lack of constraints on topology, it is also impossible to bound the convergence time.

In summary, this paper makes the following contributions:

- It provides a generic algorithm that captures a range of algorithms solving this problem in a variety of settings (Sect. 4).
- It provides a novel distributed clustering algorithm based on Gaussian Mixtures, which uses machine learning techniques to make clustering decisions (Sect. 5).
- It proves that the generic algorithm converges in very broad circumstances, over any connected topology, using any clustering criterion, in fully asynchronous settings (Sect. 6).

2 Related work

Kempe et al. [14] and Nath et al. [18] present approaches for calculating aggregates such as sums and means using gossip. These approaches cannot be directly used to perform clustering, though this work draws ideas from [14], in particular the concept of weight diffusion, and the tracing of value weights.

In the field of machine learning, clustering has been extensively studied for centrally available data sets (see [6] for a comprehensive survey). In this context, parallelization is sometimes used, where multiple processes cluster partial data

sets. Parallel clustering differs from distributed clustering in that all the data is available to all processes, or is carefully distributed among them, and communication is cheap.

Centralized clustering solutions typically overcome the Catch-22 issue explained in the introduction by running multiple iterations. They first estimate a solution, and then try to improve it by re-partitioning the values to create a better clustering. K-means [16] and Expectation Maximization [5] are examples of such algorithms. Datta et al. [4] implement the k-means algorithm distributively, whereby nodes simulate the centralized version of the algorithm. Kowalczyk and Vlassis [15] do the same for Gaussian Mixture estimation by having the nodes distributively simulate Expectation Maximization. These algorithms require multiple aggregation iterations, each similar in length to one complete run of our algorithm. The message size in these algorithms is similar to ours, dependent only on the parameters of the dataset, and not on the number of nodes. Finally, they demonstrate convergence through simulation only, but do not provide a convergence proof.

Haridasan and van Renesse [12] and Sacha et al. [19] estimate distributions in sensor networks by estimating histograms. Unlike this paper, these solutions are limited to single dimensional data values. Additionally, both use multiple iterations to improve their estimations. While these algorithms are suitable for certain distributions, they are not applicable for clustering, where, for example, small sets of distant values should not be merged with others. They also do not prove convergence.

3 Model and problem definitions

3.1 Network model

The system consists of a set of n nodes, connected by communication channels, s.t. each node i has a set of neighbors $\texttt{neighbors}_i \subset \{1, \ldots, n\}$, to which it is connected. The channels form a static directed connected network. Communication channels are asynchronous but reliable links: A node may send messages on a link to a neighbor, and eventually every sent message reaches its destination. Messages are not duplicated and no spurious messages are created.

Time is discrete, and an execution is a series of events occurring at times $t = 0, 1, 2, \ldots$.

3.2 The distributed clustering problem

At time 0, each node i takes an input val_i—a value from a domain $\mathcal{D}$. In all the examples in this paper, $\mathcal{D}$ is a d-dimensional Cartesian space $\mathcal{D} = \mathbb{R}^d$ (with $d \in \mathbb{N}$). However, in general, $\mathcal{D}$ may be any domain.

A *weighted value* is a pair $\langle val, \alpha \rangle \in \mathcal{D} \times (0, 1]$, where α is a *weight* associated with a value *val*. We associate a weight of 1 to a whole value, so, for example, $\langle val_i, 1/2 \rangle$ is half of node i's value. A set of weighted values is called a *cluster*:

Definition 1 (*Cluster*) A cluster c is a set of weighted values with unique values. The cluster's weight, $c.weight$, is the sum of the value weights:

$$c.weight \triangleq \sum_{\langle val,\alpha \rangle \in c} \alpha.$$

A cluster may be *split* into two new clusters, each consisting of the same values as the original cluster, but associated with half their original weights. Similarly, multiple clusters may be *merged* to form a new one, consisting of the union of their values, where each value is associated with the sum of its weights in the original clusters.

A cluster can be concisely described by a *summary* in a domain $\mathcal{S}$, using a function f that maps clusters to their summaries: $f : (\mathcal{D} \times (0, 1])^* \to \mathcal{S}$. The domain $\mathcal{S}$ is a pseudo-metric space (like metric, except the distance between distinct points may be zero), with a distance function $d_S : \mathcal{S}^2 \to \mathbb{R}$. *For example, in the centroids algorithm, the function f calculates the weighted average of samples in a cluster.*

A cluster c may be partitioned into several clusters, each holding a subset of its values and summarized separately.[1] The set of weighted summaries of these clusters is called a *clustering* of c. Weighted values in c may be split among clusters, so that different clusters contain portions of a given value. The sum of weights associated with a value *val* in all clusters is equal to the sum of weights associated with *val* in c. Formally:

Definition 2 (*Clustering*) A clustering C of a cluster c into J clusters $\{c_j\}_{j=1}^J$ is the set of weighted summaries of these clusters: $C = \{\langle f(c_j), c_j.weight \rangle\}_{j=1}^J$ s.t.

$$\forall val : \sum_{\langle val,\alpha \rangle \in c} \alpha = \sum_{j=1}^{J} \left(\sum_{\langle val,\alpha \rangle \in c_j} \alpha \right).$$

A clustering of a value set $\{val_j\}_{j=1}^l$ is a clustering of the cluster $\{\langle val_j, 1 \rangle\}_{j=1}^l$.

The number of clusters in a clustering is bounded by a system parameter k.

A clustering algorithm strives to partition the samples into clusters in a way that optimizes some criterion, for example, minimizes some distance metric among values assigned to the same cluster (as in k-means). In this paper, we are not concerned with the nature of this criterion, and leave it up to the application to specify the choice thereof

[1] Note that partitioning a cluster is different from splitting it, because, when a cluster is split, each part holds the same values.

A clustering algorithm maintains at every time t a clustering $clustering_i(t)$, yielding an infinite series of clusterings. For such a series, we define convergence:

Definition 3 (*Clustering convergence*) A series of clusterings

$$\left\{\{\langle f(c_j(t)), c_j(t).weight\rangle\}_{j=1}^{J_t}\right\}_{t=1}^{\infty}$$

converges to a destination clustering, which is a set of l clusters $\{dest_x\}_{x=1}^{l}$, if for every $t \in 0, 1, 2, \ldots$ there exists a mapping ψ_t between the J_t clusters at time t and the l clusters in the destination clustering $\psi_t : \{1, \ldots, J_t\} \to \{1, \ldots, l\}$, such that:

1. The summaries converge to the clusters to which they are mapped by ψ_t:

$$\max_j \left\{d_S(f(c_j(t)), f(dest_{\psi_t(j)}))\right\} \xrightarrow{t\to\infty} 0 \ .$$

2. For each cluster x in the destination clustering, the relative amount of weight in all clusters mapped to x converges to x's relative weight in the clustering:

$$\forall 1 \le x \le l : \frac{\sum_{\{j|\psi_t(j)=x\}} c_j(t).weight}{\sum_{j=1}^{J_t} c_j(t).weight} \xrightarrow{t\to\infty} \frac{dest_x.weight}{\sum_{y=1}^{l} dest_y.weight} \ .$$

We are now ready to define the problem addressed in this paper, where a set of nodes strive to learn a common clustering of their inputs. As previous works on aggregation in sensor networks [14,18,2], we define a converging problem, where nodes continuously produce outputs, and these outputs converge to such a common clustering.

Definition 4 (*Distributed clustering*) Each node i takes an input val_i at time 0 and maintains a clustering $\texttt{clustering}_i(t)$ at each time t, s.t. there exists a clustering of the input values $\{val_i\}_{i=1}^{n}$ to which the clustering in all nodes converge.

4 Generic clustering algorithm

We now present our generic algorithm that solves the Distributed Clustering Problem. At each node, the algorithm builds a clustering, which converges over time to one that describes all input values of all nodes. In order to avoid excessive bandwidth and storage consumption, the algorithm maintains clusterings as weighted summaries of clusters, and not the actual sets of weighted values. By slight abuse of terminology, we refer by the term cluster to both a set of weighted values c, and its summary–weight pair $\langle c.\texttt{summary}, c.weight\rangle$.

A node starts with a clustering of its own input value. It then periodically splits its clustering into two new ones, which have the same summaries but half the weights of the originals; it sends one clustering to a neighbor, and keeps the other. Upon receiving a clustering from a neighbor, a node merges it with its own, according to an application-specific merge rule. The algorithm thus progresses as a series of merge and split operations.

We begin with an illustrative example in Sect. 4.1 which summarizes clusters as their *centroids*—the averages of their weighted values.

Then, in Sect. 4.2, we present the generic distributed clustering algorithm. It is instantiated with a domain $\mathcal{S}$ of summaries used to describe clusters, and with application-specific functions that manipulate summaries and make clustering decisions. We use the centroid algorithm as an example instantiation.

In Sect. 4.3, we enumerate a set of requirements on the functions the algorithm is instantiated with. We then show that in any instantiation of the generic algorithm with functions that meet these requirements, the weighted summaries of clusters are the same as those we would have obtained, had we applied the algorithm's operations on the original clusters, and then summarized the results.

4.1 Example: centroids

We begin by considering the example case of centroid summaries, where a cluster is described by its centroid and weight $\langle c.\mu, c.w\rangle$. Initially, the centroid is the sensor's read value, and the weight is 1, so at node i the cluster is $\langle val_i, 1\rangle$. A node occasionally sends half of its clusters to a neighbor. A node with clusters $\langle c_1.\mu, c_1.w\rangle, \langle c_2.\mu, c_2.w\rangle$ would keep $\langle c_1.\mu, \frac{1}{2}c_1.w\rangle, \langle c_2.\mu, \frac{1}{2}c_2.w\rangle$ and send to a neighbor a message with the pair $\langle c_1.\mu, \frac{1}{2}c_1.w\rangle, \langle c_2.\mu, \frac{1}{2}c_2.w\rangle$. The neighbor receiving the message will consider the received clusters with its own, and merge clusters with close centroids. Merge is performed by calculating the weighted sum. For example, the merge of two clusters $\langle c.\mu, c.w\rangle$ and $\langle d.\mu, d.w\rangle$ is

$$\left\langle \frac{\frac{1}{2}c.w \cdot c.\mu + d.w \cdot d.\mu}{\frac{1}{2}c.w + d.w}, \frac{1}{2}c.w + d.w \right\rangle .$$

We now proceed to describe the generic algorithm.

4.2 Algorithm

The algorithm for node i is shown in Algorithm 1 (at this stage, we ignore the parts in dashed frames). The algorithm is generic, and it is instantiated with the summary domain $\mathcal{S}$ and the functions $\texttt{valToSummary}$, $\texttt{partition}$

Algorithm 1: Generic distributed data clustering algorithm. Dashed frames show auxiliary code

1 **state**
2 $clustering_i$, initially $\{\langle \texttt{valToSummary}(val_i), 1 \text{ [}, e_i \text{]} \rangle\}$
3 Periodically do **atomically**
4 Choose $j \in neighbors_i$ (Selection has to ensure fairness)
5 $old \leftarrow clustering_i$
6 $clustering_i \leftarrow \bigcup_{c \in old} \{\langle c.\texttt{summary}, half(c.weight) \text{ [}, \frac{half(c.weight)}{c.weight} \cdot c.aux \text{]} \rangle\}$
7 send $(j, \bigcup_{c \in old} \{\langle c.\texttt{summary}, c.weight - half(c.weight) \text{ [}, \left(1 - \frac{half(c.weight)}{c.weight}\right) \cdot c.aux \text{]} \rangle\})$
8 Upon receipt of *incoming* do **atomically**
9 $bigSet \leftarrow clustering_i \cup incoming$
10 $M \leftarrow \texttt{partition}(bigSet)$ (The function `partition` returns a set of cluster sets)
11 $clustering_i \leftarrow \bigcup_{x=1}^{|M|} \left\{ \langle \texttt{mergeSet}\left(\bigcup_{c \in M_x} \{\langle c.\texttt{summary}, c.weight \rangle\}\right), \sum_{c \in M_x} c.weight \text{ [}, \sum_{c \in M_x} c.aux \text{]} \rangle \right\}$
12 **function** $half(\alpha)$
13 return the multiple of q which is closest to $\alpha/2$.

and `mergeSet`. *The functions of the centroids example are given in Algorithm 2. The summary domain $\mathcal{S}$ in this case is the same as the value domain, i.e., $\mathbb{R}^d$.*

Initially, each node produces a clustering with a single cluster, based on the single value it has taken as input (Line 2). The weight of this cluster is 1, and its summary is produced by the function $\texttt{valToSummary} : \mathcal{D} \to \mathcal{S}$. *In the centroids example, the initial summary is the input value (Algorithm 2,* `valToSummary` *function).*

A node occasionally sends data to a neighbor (Algorithm 1, Lines 3–7): It first splits its clustering into two new ones. For each cluster in the original clustering, there is a matching cluster in each of the new ones, with the same summary, but with approximately half the weight. Weight is quantized, limited to multiples of a system parameter q ($q, 2q, 3q, \ldots$). This is done in order to avoid a scenario where it takes infinitely many transfers of infinitesimal weight to transfer a finite weight from one cluster to another (Zeno effect). We assume that q is small enough to avoid quantization errors: $q \ll \frac{1}{n}$. In order to respect the quantization requirement, the weight is not multiplied by exactly 0.5, but by the closest factor for which the resulting weight is a multiple of q (function *half* in Algorithm 1). One of the clusters is attributed the result of *half* and the other is attributed the complement, so that the sum of weights is equal to the original, and system-wide conservation of weight is maintained. Note that despite the weight quantization, values and summaries may still be continuous, therefore convergence may still be continuous.

If the communication topology is dense, it is possible to perform scalable random peer sampling [17], even under message loss [11], in order to achieve data propagation guarantees.

Algorithm 2: Centroid Functions

1 **function** `valToSummary`(val)
2 return val
3 **function** `mergeSet`(`clusters`)
4 return $\left(\sum_{\substack{\langle avg, m \rangle \in \\ \texttt{clusters}}} m \right)^{-1} \times \sum_{\substack{\langle avg, m \rangle \in \\ \texttt{clusters}}} m \cdot avg$
5 **function** `partition`($bigSet$)
6 $M \leftarrow \{\{c\}\}_{c \in bigSet}$
7 If there are sets in M whose clusters' weights are q, then unify them arbitrarily with others
8 **while** $|M| > k$ **do**
9 let M_x and M_y be the (different) cluster sets in M whose centroids are closest
10 $M \leftarrow M \setminus \{M_x, M_y\} \cup (M_x \cup M_y)$
11 return M

The node then keeps one of the new clusterings, replacing its original one (Line 6), and sends the other to some neighbor j (Line 7). The selection of neighbors has to ensure fairness in the sense that in an infinite run, each neighbor is chosen infinitely often; this can be achieved, e.g., using round robin. Alternatively, the node may implement gossip communication patterns: It may choose a random neighbor and send data to it (push), or ask it for data (pull), or perform a bilateral exchange (push-pull).

When a message with a neighbor's clustering reaches the node, an event handler (Lines 8–11) is called. It first combines the two clusterings of the nodes into a set *bigSet* (Line 9). Then, an application-specific function `partition` divides the clusters in *bigSet* into sets $M = \{M_x\}_{x=1}^{|M|}$ (Line 10). The clusters in each of the sets in M are merged into a

single cluster, together forming the new clustering of the node (Line 11). The summary of each merged cluster is calculated by another application-specific function, `mergeSet`, and its weight is the sum of weights of the merged clusters.

To conform with the restrictions of k and q, the partition function must guarantee that (1) $|M| \leq k$; and (2) no M_x includes a single cluster of weight q (that is, every cluster of weight q is merged with at least one other cluster).

Note that the parameter k forces lossy compression of the data, since merged values cannot later be separated. At the beginning, only a small number of data values is known to the node, so it performs only a few (easy) clustering decisions. As the algorithm progresses, the number of values described by the node's clustering increases. By then, it has enough knowledge of the data set, so as to perform correct clustering decisions, and achieve a high compression ratio without losing valuable data.

In the centroids algorithm, the summary of the merged set is the weighted average of the summaries of the merged clusters, calculated by the implementation of `mergeSet` *shown in Algorithm 2. Merging decisions are based on the distance between cluster centroids. Intuitively, it is best to merge close centroids, and keep distant ones separated. This is done greedily by* `partition` *(shown in Algorithm 2) which repeatedly merges the closest sets, until the k bound is reached. For* $k = 1$*, the algorithm is reduced to push-sum.*

4.3 Auxiliaries and instantiation requirements

For the algorithm to perform a meaningful and correct clustering of the data, its functions must respect a set of requirements. In Sect. 4.3.1 we specify these requirements and in Sect. 4.3.2 we show that the centroids algorithm described above meets these requirements. In Sect. 4.3.3 we prove that these requirements ensure that the summaries described by the algorithm indeed represent clusters.

4.3.1 Instantiation requirements

To phrase the requirements, we describe a cluster in $\langle \mathcal{D}, (0, 1] \rangle^*$ as a vector in the *Mixture Space*—the space $\mathbb{R}^n$ (n being the number of input values), where each coordinate represents one input value. A cluster is described in this space as a vector whose j'th component is the weight associated with val_j in that cluster. For a given input set, a vector in the mixture space precisely describes a cluster. We can therefore redefine f as a mapping from mixture space vectors of clusters to cluster summaries, according to the input set $I \in \mathcal{D}^n$. We denote this mapping $f_I : \mathbb{R}^n \to \mathcal{S}$.

We define the distance function $d_M : (\mathbb{R}^n)^2 \to \mathbb{R}$ between two vectors in the mixture space to be the angle between them. Clusters consisting of similar weighted values are close in the mixture space (according to d_M). Their summaries should be close in the summary space (according to d_S), with some scaling factor ρ. Simply put—clusters consisting of similar values (i.e., close in d_M) should have similar summaries (i.e., close in d_S). Formally:

R1 For any input value set I,

$$\exists \rho : \forall v_1, v_2 \in (0, 1]^n : d_S(f_I(v_1), f_I(v_2)) \leq \rho \cdot d_M(v_1, v_2).$$

In addition, operations on summaries must preserve the relation to the clusters they describe. Intuitively, this means that operating on summaries is similar to performing the various operations on the value set, and then summarizing the results.

R2 Initial values are mapped by f_I to their summaries:

$$\forall i, 1 \leq i \leq n : \texttt{valToSummary}(val_i) = f_I(e_i).$$

R3 Summaries are oblivious to weight scaling:

$$\forall \alpha > 0, v \in (0, 1]^n : f_I(v) = f_I(\alpha v).$$

R4 Merging a summarized description of clusters is equivalent to merging these clusters and then summarizing the result:[2]

$$\texttt{mergeSet}\left(\bigcup_{v \in V} \langle \{ f_I(v), \|v\|_1 \rangle \} \right) = f_I\left(\sum_{v \in V} v \right).$$

4.3.2 The centroids case

We show now that the centroids algorithm respects the requirements. Recall that f_I in this case is the weighted average of the samples, and let d_S be the L^2 distance between centroids. We show that the requirements are respected.

Claim For the centroids algorithm, as described in Algorithm 2, the requirements R1–R4 are respected.

Proof Let ρ be the maximal L^2 distance between values, and let $\tilde{v}$ be the L^2 normalized vector v. We show that R1 holds with this ρ.

[2] Denote by $\|v\|_p$ the L^p norm of v.

$$
\begin{aligned}
& d_S(f_I(v_1), f_I(v_2)) \\
& \overset{(1)}{\leq} \rho \|v_1 - v_2\|_1 \\
& \overset{(2)}{\leq} \rho \cdot n^{-1/2} \|\tilde{v}_1 - \tilde{v}_2\|_1 \\
& \overset{(3)}{\leq} \rho \|\tilde{v}_1 - \tilde{v}_2\|_2 \\
& \overset{(4)}{\leq} \rho \cdot d_M(\tilde{v}_1, \tilde{v}_2) = \rho \cdot d_M(v_1, v_2)
\end{aligned}
$$

(1) Each value may contribute at most ρ the coordinate difference.
(2) The normalization from L^1 to L^2 may factor each dimension by no less than $n^{-1/2}$.
(3) The L^1 norm is smaller than $\sqrt{n}$ times the L^2 norm, so a factor of $\sqrt{n}$ is added that annuls the $n^{-1/2}$ factor.
(4) Recall that d_M is the angle between the two vectors. The L^2 difference of normalized vectors is smaller than the angle between them.

It is readily seen that requirements R2–R4 also hold. □

4.3.3 Auxiliary correctness

Returning to the generic case, we show that the weighted summaries maintained by the algorithm to describe clusters that are merged and split, indeed do so. To do that, we define an auxiliary algorithm. This is an extension of Algorithm 1 with the auxiliary code in the dashed frames. Clusters are now triplets, containing, in addition to the summary and weight, the cluster's mixture space vector $c.aux$.

At initialization (Line 2), the auxiliary vector at node i is e_i (a unit vector whose i'th component is 1). When splitting a cluster (Lines 6–7), the vector is factored by about $1/2$ (the same ratio as the weight). When merging a set of clusters, the mixture vector of the result is the sum of the original clusters' vectors (Line 11).

The following lemma shows that, at all times, the summary maintained by the algorithm is indeed that of the cluster described by its mixture vector:

Lemma 1 (Auxiliary correctness) *The generic algorithm, instantiated by functions satisfying R2–R4, maintains the following invariant: For any cluster c either in a node's clustering* ($c \in \texttt{clustering}_i$) *or in transit in a communication channel, the following two equations hold:*

$$f_I(c.aux) = c.\texttt{summary} \quad (1)$$

$$\|c.aux\|_1 = c.weight \quad (2)$$

Proof By induction on the global states of the system.

Basis Initialization puts at time 0, at every node i the auxiliary vector e_i, a weight of 1, and the summary `valToSummary` of value i. Requirement R2 thus ensures that Eq. 1 holds in the initial state, and Eq. 2 holds since $\|e_i\| = 1$. Communication channels are empty.

Assumption At time $j-1$ the invariant holds.

Step Transition j may be either send or receive. Each of them removes clusters from the set, and produces a cluster or two. To prove that at time j the invariant holds, we need only show that in both cases the new cluster(s) maintain the invariant.

Send We show that the mapping holds for the kept cluster c_{keep}. A similar proof holds for the sent one c_{send}. Proof of Eq. 1:

$$
\begin{aligned}
c_{\text{keep}}.\texttt{summary} &\overset{\text{line 6}}{=} c.\texttt{summary} \\
&\overset{\text{induction assumption}}{=} f_I(c.aux) \\
&\overset{\text{R3}}{=} f_I\left(\frac{half(c.weight)}{c.weight} \cdot c.aux\right) \\
&\overset{\text{auxiliary line 6}}{=} f_I(c_{\text{keep}}.aux)
\end{aligned}
$$

Proof of Eq. 2:

$$
\begin{aligned}
c_{\text{keep}}.weight &\overset{\text{line 6}}{=} half(c.weight) \\
&= \frac{half(c.weight)}{c.weight} \cdot c.weight \\
&\overset{\text{induction assumption}}{=} \frac{half(c.weight)}{c.weight} \cdot \|c.aux\|_1 \\
&= \left\| \frac{half(c.weight)}{c.weight} \cdot c.aux \right\|_1 \\
&\overset{\text{auxiliary line 6}}{=} \|c_{\text{keep}}.aux\|_1
\end{aligned}
$$

Receive We prove that the mapping holds for each of the m produced clusters. Each cluster c_x is derived from a set M_x. Proof of Eq. 1:

$$
\begin{aligned}
& c_x.\texttt{summary} \\
& \overset{\text{line 11}}{=} \texttt{mergeSet}\left(\bigcup_{c \in M_x} \{\langle c.\texttt{summary}, c.weight\rangle\}\right) \\
& \overset{\text{induction assumption}}{=} \texttt{mergeSet}\left(\bigcup_{c \in M_x} \{\langle f_I(c.aux), \|c.aux\|_1\rangle\}\right) \\
& \overset{\text{R4}}{=} f_I\left(\sum_{c \in M_x} c.aux\right) \\
& \overset{\text{auxiliary line 11}}{=} f_I\,(c_x.aux)
\end{aligned}
$$

Proof of Eq. 2:

$$
\begin{aligned}
c_x.weight &\overset{\text{line 11}}{=} \sum_{c \in M_x} c.weight \\
&\overset{\substack{\text{induction}\\ \text{assumption}}}{=} \sum_{c \in M_x} \|c.aux\|_1 \\
&= \left\| \sum_{c \in M_x} c.aux \right\|_1 \\
&\overset{\substack{\text{auxiliary}\\ \text{line 11}}}{=} \|c_x.aux\|_1
\end{aligned}
$$

□

5 Gaussian clustering

When clustering value sets from a metric space, the centroids solution is seldom sufficient. Consider the example shown in Fig. 1, where we need to associate a new value with one of two existing clusters. Figure 1a shows the information that the centroids algorithm has for clusters A and B, and a new value. The algorithm would associate the new value to cluster A, on account of it being closer to its centroid. However, Fig. 1b shows the set of values that produced the two clusters. We see that it is more likely that the new value in fact belongs to cluster B, since it has a much larger variance.

The field of machine learning suggests the heuristic of clustering data using a Gaussian Mixture (a weighted set of normal distributions), allowing for a rich and accurate description of multivariate data. Figure 1b illustrates the summary employed by GM: An ellipse depicts an equidensity line of the Gaussian summary of each cluster. Given these Gaussians, one can easily classify the new value correctly.

We present in Sect. 5.1 the GM algorithm—a new distributed clustering algorithm, implementing the generic one by representing clusters as Gaussians, and clusters as Gaussian Mixtures. Contrary to the classical machine learning algorithms, ours performs the clustering without collecting the data in a central location. Nodes use the popular machine learning tool of Expectation Maximization to make clustering decisions (Sect. 5.1). A taste of the results achieved by our GM algorithm is given in Sect. 5.2 via simulation. It demonstrates the clustering of multidimensional data and more. Note that due to the heuristic nature of EM, the only possible evaluation of our algorithm's quality is empirical.

5.1 Generic algorithm instantiation

The summary of a cluster is a tuple $\langle \mu, \sigma \rangle$, comprised of the average of the weighted values in the cluster $\mu \in \mathbb{R}^d$ (where $\mathcal{D} = \mathbb{R}^d$ is the value space), and their covariance matrix $\sigma \in \mathbb{R}^{d \times d}$. Together with the weight, a cluster is described by a weighted Gaussian, and a clustering consists of a weighted set of Gaussians, or a *Gaussian Mixture*.

Let $v = (v_1, \ldots, v_n)$ be an auxiliary vector; we denote by $\tilde{v}$ a normalized version thereof:

$$\tilde{v} = \frac{v}{\sum_{j=1}^{s} v_j}$$

Recall that v_j represents the weight of val_j in the cluster. The centroid $\mu(v)$ and covariance matrix $\sigma(v)$ of the weighted values in the cluster are calculated as follows:

$$\mu(v) = \sum_{j=1}^{n} \tilde{v}_j \cdot val_j \quad \text{, and}$$

$$\sigma(v) = \frac{1}{1 - \sum_{k=1}^{n} \tilde{v}_k^2} \sum_{j=1}^{n} \tilde{v}_j (val_j - \mu)(val_j - \mu)^T .$$

We use them to define the mapping f_I from the mixture space to the summary space:

$$f_I(v) = \langle \mu(v), \sigma(v) \rangle .$$

Note that the use of the normalized vector $\tilde{v}$ makes both $\mu(v)$ and $\sigma(v)$ invariant under weight scaling, thus fulfilling Requirement R3.

We define d_S as in the centroids algorithm. Namely, it is the L^2 distance between the centroids of clusters. This fulfills requirement R1 (see Sect. 4.3.2).

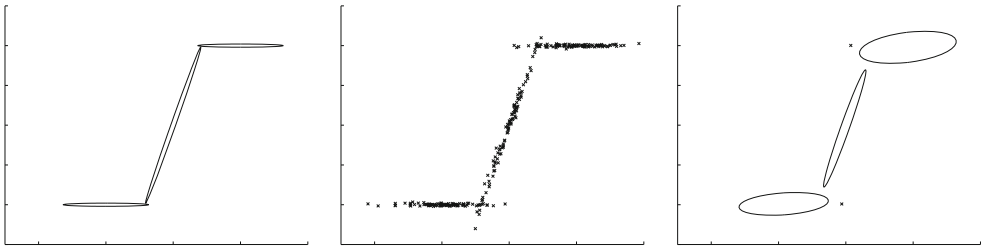
The function `valToSummary` returns a cluster with an average equal to *val*, a zero covariance matrix, and a weight of 1. Requirement R2 is trivially satisfied.

To describe the function `mergeSet` we use the following definitions: Denote the weight, average and covariance matrix of cluster x by w_x, μ_x and σ_x, respectively. Given the summaries and weights of two clusters a and b, one can calculate the summary of a cluster c created by merging the two:

$$
\begin{aligned}
\mu_c &= \frac{w_a}{w_a + w_b}\mu_a + \frac{w_b}{w_a + w_b}\mu_b \\
\sigma_c &= \frac{w_a}{w_a + w_b}\sigma_a + \frac{w_b}{w_a + w_b}\sigma_b \\
&\quad + \frac{w_a \cdot w_b}{(w_a + w_b)^2} \cdot (\mu_a - \mu_b) \cdot (\mu_a - \mu_b)^T
\end{aligned}
$$

(a) Centroids and a new value **(b)** Gaussians and a new value

Fig. 1 Associating a new value when clusters are summarized **a** as centroids and **b** as Gaussians



(a) Distribution **(b)** Values **(c)** Result

Fig. 2 Gaussian Mixture clustering example. The three Gaussians in (**a**) were used to generate the data set in (**b**). The GM algorithm produced the estimation in (**c**)

This merging function maintains the average and covariance of the original values [20], therefore it can be iterated to merge a set of summaries and implement `mergeSet` in a way that conforms to R4.

Expectation maximization partitioning

To complete the description of the GM algorithm, we now explain the `partition` function. When a node has accumulated more than k clusters, it needs to merge some of them. In principle, it would be best to choose clusters to merge according to Maximum Likelihood, which is defined in this case as follows: We denote a Gaussian Mixture of x Gaussians *x-GM*. Given a too large set of $l \geq k$ clusters, an *l-GM*, the algorithm tries to find the *k-GM* probability distribution for which the *l-GM* has the maximal likelihood. However, computing Maximum Likelihood is NP-hard. We therefore instead follow common practice and approximate it with the *Expectation Maximization* algorithm [16].

Our goal is to re-classify GM_{old}, an *l-GM* with $l > k$, to GM_{new}, a *k-GM*. Denote by V the d dimensional space in which the distributions are defined. Denote by $f_X(v)$ the probability density at point v of distribution X. If X is a weight distribution such as a Gaussian mixture, it is normalized s.t. it constitutes a probability density.

The *likelihood* that the samples concisely described by GM_{old} are the result of the probability distribution described by (the normalized) GM_{new} is:

$$L = \sum_{c \in GM_{\text{new}}} \sum_{g \in GM_{\text{old}}} \left(\int_{v \in V} w_c f_c(v) \cdot w_g f_g(v) dv \right)$$

The merge employs the Expectation Maximization algorithm to approximate Maximum Likelihood. It arbitrarily groups the clusters in GM_{old} into k sets, and merges each set into a single Gaussian, forming a *k-GM* GM_{new}. It then alternately regroups GM_{old}'s clusters to maximize their likelihood w.r.t. GM_{new}, and recalculates GM_{new} according to this grouping. This process is repeated until convergence.

5.2 Simulation results

Due to the heuristic nature of the Gaussian Mixture clustering and of EM, the quality of their results is typically evaluated experimentally. In this section, we briefly demonstrate the effectiveness of our GM algorithm through simulation. First, we demonstrate the algorithm's ability to cluster multidimensional data, which could be produced by a sensor network. Then, we demonstrate a possible application using the algorithm to calculate the average while removing erroneous data reads and coping with node failures. This result also demonstrates the convergence speed of the algorithm.

In both cases, we simulate a fully connected network of 1,000 nodes. Like previous works [7,12], we measure progress in rounds, where in each round each node sends a clustering to one neighbor. Nodes that receive clusterings from multiple neighbors accumulate all the received clusters and run EM once for the entire set.

More simulation results and analysis can be found in [8].

5.2.1 Multidimensional data clustering

As an example input, we use data generated from a set of three Gaussians in $\mathbb{R}^2$. Values are generated according to the distribution shown in Fig. 2a, where the ellipses are equidensity contours of normal distributions. This input might describe temperature readings taken by a set of sensors positioned on a fence located by the woods, and whose right side is close to a fire outbreak. Each value is comprised of the sensor's location x and the recorded temperature y. The generated input values are shown in Fig. 2b. We run the GM algorithm with this input until its convergence; $k = 7$ and q is set by floating point accuracy.

The result is shown in Fig. 2c. The ellipses are equidensity contours, and the x's are singleton clusters (with zero variance). This result is visibly a usable estimation of the input data.

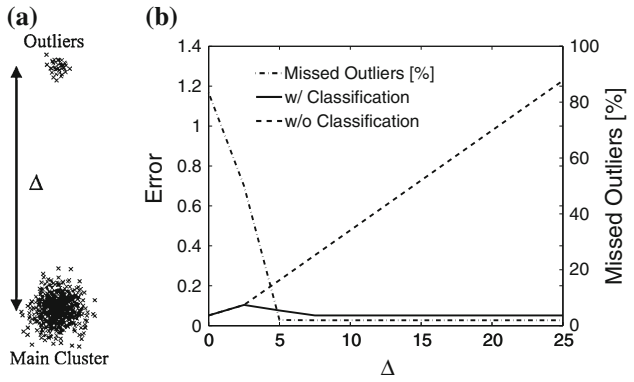


Fig. 3 Effect of the separation of erroneous samples on the calculation of the average: A 1,000 values are sampled from two Gaussian distributions (**a**). As the erroneous samples' distribution moves away from the good one, the regular aggregation error grows linearly. However, once the distance is large enough, our protocol can remove the erroneous samples, which results in an accurate estimation of the mean

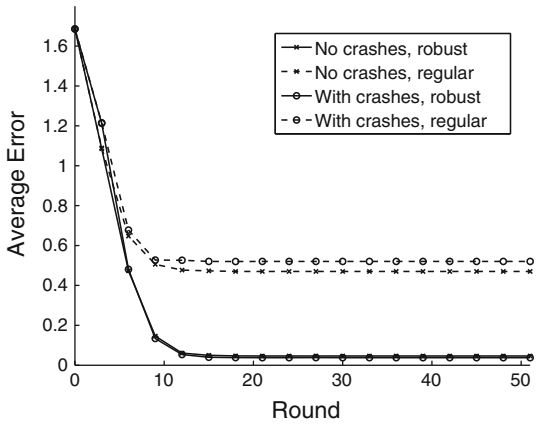


Fig. 4 The effect of crashes on convergence speed and on the accuracy of the mean

5.2.2 Robustness

Erroneous samples removal As an example application, we use the algorithm to calculate a statistically robust average. We consider a sensor network of 1,000 sensors reading values in $\mathbb{R}^2$. Most of these values are sampled from a given Gaussian distribution and we wish to calculate their average. Some values, however, are erroneous, and are unlikely to belong to this distribution. They may be the result of a malfunctioning sensor, or of a sensing error, e.g., an animal sitting on an ambient temperature sensor. These values should be removed from the statistics.

We use 950 values from the standard normal distribution, i.e., with a mean $(0, 0)$ and a unit covariance matrix I. Fifty additional values are distributed normally with covariance matrix $0.1 \cdot I$ and mean $(0, \Delta)$, with Δ ranging between 0 and 25. The distribution of all values is illustrated in Fig. 3a.

For each value of Δ, the protocol is run until convergence. We use $k = 2$, so that each node has at most 2 clusters at any given time—hopefully one for good values and one for the erroneous values.

The results are shown in Fig. 3. The dotted line shows the average weight ratio belonging to erroneous samples yet incorrectly assigned to the good cluster. Erroneous samples are defined to be values with probability density lower than $f_{min} = 5 \times 10^{-5}$ (for the standard normal distribution). The other two lines show the error in calculating the mean, where error is the average over all nodes of the distance between the estimated mean and the true mean $(0, 0)$. The solid line shows the result of our algorithm, which removes erroneous samples, while the dashed line shows the result of regular average aggregation, which does not.

We see that when the erroneous samples are close to the good values, the number of misses is large—the proximity of the clusters makes their separation difficult. However, due to the small distance, this mistake hardly influences the estimated average. As the erroneous samples' mean moves further from the true mean, their identification becomes accurate and their influence is nullified.

Note that even for large Δ's, a certain number of erroneous samples is still missed. These are values from the good distribution, relatively close to the main cluster, yet with probability density lower than f_{min}. The protocol mistakenly considers these to be good values. Additionally, around $\Delta = 5$ the miss rate is dropped to its minimum, yet the robust error does not. This is due to the fact that bad values are located close enough to the good mean so that their probability density is higher than f_{min}. The protocol mistakes those to belong to f_G and allows them to influence the mean. That being said, for all Δ's, the error remains small, confirming the conventional wisdom that "clustering is either easy or not interesting".

Crash robustness and convergence speed We next examine how crash failures impact the results obtained by our protocol. Figure 4 shows that the algorithm is indifferent to crashes of nodes. The source data is similar to the one above, with $\Delta = 10$. After each round, each node crashes with probability 0.05. We show the average node estimation error of the mean in each round. As we have seen above, our protocol achieves a lower error then the regular one.

Figure 4 also demonstrates the convergence speed of our algorithm. With and without crashes, the convergence speed of our algorithm is equivalent to that of the regular average aggregation algorithm.

5.2.3 Scalability

To evaluate convergence time we measure the number of rounds until the estimations at the different nodes are the same. The samples are taken from a Gaussian mixture of two

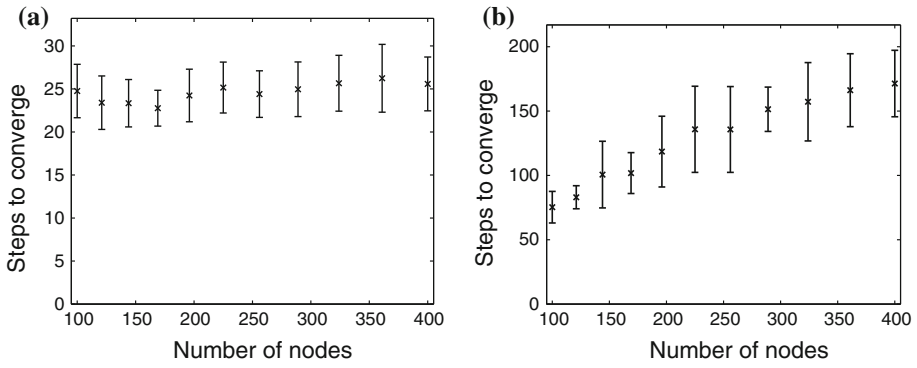


Fig. 5 Convergence time of the distributed clustering algorithm as a function of the number of nodes **a** in a fully connected topology and **b** in a grid topology

Gaussians of the same weight with variance 1 at distance 5 from each other. Since there is no scalar convergence value, the $\varepsilon - \delta$ measure which is used, e.g., for analysing Push-Sum, is not applicable to this scenario. Instead, we use the Kolmogorov-Smirnov (KS) statistic as the measure of difference between two distributions.[3] For a range of network sizes, we let the algorithm run until the maximal KS-statistic between the estimations of any pair of nodes.[4] falls below an arbitrary threshold of 0.01 The results for a complete topology and a grid topology (with samples taken independently of the grid coordinates) are shown in Fig. 5a, b, respectively. For each network size we show the average convergence time with the 95% confidence interval.

As expected, the scalability in a grid topology is worse than in a complete topology. The trends shown in these figures match those calculated by Boyd et al. [3] for the Push-Sum algorithm.

6 Convergence proof

We now prove that the generic algorithm presented in Sect. 4 solves the distributed clustering problem. To prove convergence, we consider the pool of all the clusters in the system, at all nodes and communication channels. This pool is in fact, at all times, a clustering of the set of all input values. In Sect. 6.1 we prove that the pool of all clusters converges, i.e., roughly speaking, it stops changing. Then, in Sect. 6.2, we prove that the clusterings in all nodes converge to the same destination.

6.1 Collective convergence

In this section, we ignore the distributive nature of the algorithm, and consider all the clusters in the system (at both processes and communication channels) at time t as if they belonged to a single multiset $pool(t)$. A run of the algorithm can therefore be seen as a series of splits and merges of clusters.

To argue about convergence, we first define the concept of cluster descendants. Intuitively, for $t_1 \leq t_2$, a cluster $c_2 \in pool(t_2)$ is a descendant of a cluster $c_1 \in pool(t_1)$ if c_2 is equal to c_1, or is the result of operations on c_1. Formally:

Definition 5 (*Cluster genealogy*) We recursively define the descendants of a cluster $c \in pool(t)$. First, at t, the descendant set is simply $\{c\}$. Next, consider $t_1 > t$.

Assume the t_1'th operation in the execution is splitting (and sending) (Lines 3–7) a set of clusters $\{c_x\}_{x=1}^{l} \subset pool(t_1 - 1)$. This results in two new sets of clusters, $\{c_x^1\}_{x=1}^{l}$ and $\{c_x^2\}_{x=1}^{l}$, being put in $pool(t_1)$ instead of the original set. If a cluster c_x is a descendant of c at $t_1 - 1$, then the clusters c_x^1 and c_x^2 are descendants of c at t_1.

Assume the t_1'th operation is a (receipt and) merge (Lines 8–11), then some m ($1 \leq m \leq k$) sets of clusters $\{M_x\}_{x=1}^{m} \subset pool(t_1 - 1)$ are merged and are put in $pool(t_1)$ instead of the merged ones. For every M_x, if any of its clusters is a descendant of c at $t_1 - 1$, then its merge result is a descendant of c at t_1.

By slight abuse of notation, we write $v \in pool(t)$ when v is the mixture vector of a cluster c, and $c \in pool(t)$; vector genealogy is similar to cluster genealogy.

We now state some definitions and the lemmas used in the convergence proof. We prove that, eventually, the descendants of each vector in the pool converge (normalized) to one destination. To do that, we investigate the angles between a vector v and the axes unit vectors. Note that all angles are between zero and $\pi/2$. For $i \in \{1, \ldots, d\}$, we call the angle between v and the i'th axis v's i'th *reference angle* and denote it by φ_i^v. We denote by $\varphi_{i,\max}(t)$ the maximal i'th reference angle over all vectors in the pool at time t:

$$\varphi_{i,\max}(t) \triangleq \max_{v \in \mathtt{pool}(t)} \varphi_i^v \ .$$

We now show that the i'th reference angle is monotonically decreasing for any $1 \leq i \leq n$. To achieve this, we use

[3] The Kolmogorov-Smirnov statistic for two distributions is the maximal difference between their cumulative distribution functions.

[4] To shorten simulation time, we calculate the statistics for $4n$ random pairs of nodes.

Lemma 2 which states that the sum of two vectors has an i'th reference angle not larger than the larger i'th reference angle of the two. Its proof is deferred to Appendix A.

Lemma 2 (Decreasing reference angle) *The sum of two vectors in the mixture space is a vector with a smaller i'th reference angle than the larger i'th reference angle of the two, for any $1 \leq i \leq n$.*

We are now ready to prove that the maximal reference angle is monotonically decreasing:

Lemma 3 *For $1 \leq i \leq n$, $\varphi_{i,max}(t)$ is monotonically decreasing.*

Proof The pool changes in split and merge operations. In case of a split, the new vectors have the same angles as the split one, so $\varphi_{i,\max}$ is unchanged. In case of a merge, a number of vectors are replaced by their sum. This can be seen as the result of a series of steps, each of which replaces two vectors by their sum. The sum of two vectors is a vector with a no larger i'th reference angle than the larger of the i'th reference angles of the two (Lemma 2). Therefore, whenever a number of vectors are replaced by their sum, the maximal reference angle may either remain the same of decrease. □

Since the maximal reference angles are bounded from below by zero, Lemma 3 shows that they converge, and we can define

$$\hat{\varphi}_{i,\max} \stackrel{\Delta}{=} \lim_{t\to\infty} \varphi_{i,\max}(t) \ .$$

By slight abuse of terminology, we say that the i'th reference angle of a vector $v \in pool(t)$ *converges* to φ, if for every ε there exists a time t', after which the i'th reference angles of all of v's descendants are in the ε neighborhood of φ.

We proceed to show that there exists a time after which the pool is partitioned into clusters, and the vectors from each cluster merge only with one another. Moreover, the descendants of all vectors in a cluster converge to the same reference angle. More specifically, we show that the vectors in the pool are partitioned into clusters by the algorithm according to the i'th reference angle their descendants converge to (for any $1 \leq i \leq n$). We further show that, due to the quantization of weight, a gap is formed between descendants that converge to the maximal reference angle, and those that do not, as those that do not remain within some minimum distance ε from $\hat{\varphi}_{i,\max}$.

Since the i'th maximal reference angle converges (Lemma 2), for every ε there exists a time after which there are always vectors in the ε neighborhood of $\hat{\varphi}_{i,\max}$. The weight (sum of L^1 norms of vectors) in this neighborhood changes over time, and due to the quantization of weight there exists a minimal weight q^i_ε such that for every time t there exists a time $t' > t$ when the weight in the neighborhood is q^i_ε.

The following observations immediately follow:

Observation 1 *For every $\varepsilon' < \varepsilon$, the relation $q^i_{\varepsilon'} \leq q^i_\varepsilon$ holds. Moreover, $q^i_\varepsilon - q^i_{\varepsilon'} = l \cdot q$ with $l \in \{0, 1, \ldots\}$.*

Observation 2 *There exists an ε such that for every $\varepsilon' < \varepsilon$, the minimal weight in the ε' neighborhood of $\hat{\varphi}_{i,max}$ is the same as for ε. That is, $q^i_\varepsilon = q^i_{\varepsilon'}$.*

The next lemma shows that vectors from different sides of the gap are never merged. Its proof is deferred to Appendix B.

Lemma 4 *For any ε, there exists an $\varepsilon' < \varepsilon$ such that if a vector v_{out} lies outside the ε-neighborhood of $\hat{\varphi}_{i,max}$ (i.e., has a reference angle smaller than $\hat{\varphi}_{i,max} - \varepsilon$), and a vector v_{in} lies inside the ε'-neighborhood (i.e., has a reference angle larger than $\hat{\varphi}_{i,max} - \varepsilon'$), then their sum v_{sum} lies outside the ε' neighborhood.*

We are now ready to prove the that eventually the vectors are partitioned.

Lemma 5 (Cluster formation) *For every $1 \leq i \leq n$, there exists a time t_i and a set of vectors*

$$V_{i,max} \subset pool(t_i)$$

s.t. the i'th reference angles of the vectors converge to $\hat{\varphi}_{i,max}$, and their descendants are merged only with one another.

Proof For a given i, choose an ε such that for every $\varepsilon' < \varepsilon$ the minimal weights are the same: $q^i_\varepsilon = q^i_{\varepsilon'}$. Such an ε exists according to Observation 2.

According to Lemma 4, there exists an $\tilde{\varepsilon}$ s.t. the sum of a vector inside the $\tilde{\varepsilon}$ neighborhood and a vector outside the ε neighborhood is outside the $\tilde{\varepsilon}$ neighborhood. Choose such an $\tilde{\varepsilon}$.

Denote

$$v_{\text{in},\tilde{\varepsilon}} \stackrel{\Delta}{=} \sum_{v' \in V_{\text{in},\tilde{\varepsilon}}} v' \ , \quad v_{\text{out},\varepsilon} \stackrel{\Delta}{=} \sum_{v' \in V_{\text{out},\varepsilon}} v' \ .$$

Since the $V_{\text{out},\varepsilon}$ vectors have reference angles outside the ε neighborhood, $v_{\text{out},\varepsilon}$ is also outside the ε neighborhood (Lemma 2). $v_{\text{in},\tilde{\varepsilon}}$ may either be inside the $\tilde{\varepsilon}$ neighborhood or outside it. If $v_{\text{in},\tilde{\varepsilon}}$ is inside the $\tilde{\varepsilon}$ neighborhood, then the sum v is outside the ε neighborhood, due to the choice of $\tilde{\varepsilon}$. If it is outside, then v is outside the $\tilde{\varepsilon}$ neighborhood (Lemma 2 again).

Choose a t_i s.t. $t_i > t_{\tilde{\varepsilon}}$ and at t_i the ε neighborhood contains a weight q_ε. Since $q_\varepsilon = q_{\tilde{\varepsilon}}$, the weight in the $\tilde{\varepsilon}$ neighborhood cannot be smaller than $q_{\tilde{\varepsilon}}$, therefore the weight is actually in the $\tilde{\varepsilon}$ neighborhood.

We now show that all operations after t_i keep the descendants of the vectors that were in the $\tilde{\varepsilon}$ neighborhood at t_i

inside that neighborhood, and never mix them with the other vector descendants, all of which remain outside the ε neighborhood.

We prove by induction that the descendants of the vectors that were inside the $\tilde{\varepsilon}$ at t_i are always in this neighborhood, and the descendants of the vectors outside the ε neighborhood at t_i are always outside this neighborhood. The assumption holds at t_i. Assume it holds at t_j. If the step is a send operation (Lines 3–7), it does not change the angle of the descendants, therefore the claim holds at t_{j+1}. If the step is a receive operation (Lines 8–11), then vectors are merged. There is never a merger of vectors from both inside the $\tilde{\varepsilon}$ neighborhood and outside the ε neighborhood, since the result is outside the $\tilde{\varepsilon}$ neighborhood, leaving inside it a weight smaller than $q_{\tilde{\varepsilon}}$. Due to the same reason, the sum of vectors inside the $\tilde{\varepsilon}$ neighborhood is always inside this neighborhood. Finally, the sum of two vectors outside the ε neighborhood is outside the ε neighborhood (Lemma 2).

Due to the choice of ε, for every $\varepsilon' < \tilde{\varepsilon}$ there exists a time after which there are vectors of weight q_ε in the ε' neighborhood of $\hat{\varphi}_{i,\max}$. According to what was shown above, these are descendants of the set of vectors $V_{\text{in},\tilde{\varepsilon}}$ that are never mixed with vectors that are not descendants thereof. This set is therefore the required $V_{i,\max}$. □

We next prove that the pool of auxiliary vectors converges:

Lemma 6 (Auxiliary collective convergence) *There exists a time t, such that the normalized descendants of each vector in $pool(t)$ converge to a specific destination vector, and merge only with descendants of vectors that converge to the same destination.*

Proof By Lemmas 3 and 5, for every $1 \leq i \leq n$, there exist a maximal i'th reference angle, $\hat{\varphi}_{i,\max}$, a time, t_i, and a set of vectors, $V_{i,\max} \subset pool(t_i)$, s.t. the i'th reference angles of the vectors $V_{i,\max}$ converge to $\hat{\varphi}_{i,\max}$, and the descendants of $V_{i,\max}$ are merged only with one another.

The proof continues by induction. At t_i we consider the vectors that are not descendants of $V_{i,\max} \in pool(t_i)$. The descendants of these vectors are never merged with the descendants of the $V_{i,\max}$ vectors. Therefore, the proof applies to them with a new maximal i'th reference angle. This can be applied repeatedly, and since the weight of the vectors is bounded from below by q, we conclude that there exists a time t after which, for every vector v in the pool at time $t' > t$, the i'th reference of v converges. Denote that time $t_{conv,i}$.

Next, let $t_{conv} = \max\{t_{conv,i} | 1 \leq i \leq n\}$. After t_{conv}, for any vector in the pool, all of its reference angles converge. Moreover, two vectors are merged only if all of their reference angles converge to the same destination. Therefore, at t_{conv}, the vectors in $pool(t_{conv})$ can be partitioned into disjoint sets s.t. the descendants of each set are merged only with one another and their reference angles converge to the same values. For a cluster x of vectors whose reference angles converge to $(\varphi_i^x)_{i=1}^n$, its destination in the mixture space is the normalized vector $(\cos \varphi_i^x)_{i=1}^n$. □

We are now ready to derive the main result of this section.

Corollary 1 *The clustering series $pool(t)$ converges.*

Proof Lemma 6 shows that the pool of vectors is eventually partitioned into clusters. This applies to the weighted summaries pool as well, due to the correspondence between summaries and auxiliaries (Lemma 1).

For a cluster of clusters, define its destination cluster as follows: Its weight is the sum of weights of clusters in the cluster at t_{conv}, and its summary is that of the mixture space destination of the cluster's mixture vectors. Using requirement R1, it is easy to see that after t_{conv}, the clustering series $pool(*)$ converges to the set of destination clusters formed this way. □

6.2 Distributed convergence

We show that the clusterings in each node converge to the same clustering of the input values.

Lemma 7 *There exists a time t_{dist} after which each node holds at least one cluster from each cluster of clusters.*

Proof First note that after t_{conv}, once a node has obtained a cluster that converges to a destination x, it will always have a cluster that converges to this destination, since it will always have a descendant of that cluster—no operation can remove it.

Consider a node i that obtains a cluster that converges to a destination x. It eventually sends a descendant thereof to each of its neighbors due to the fair choice of neighbors. This can be applied repeatedly and show that, due to the connectivity of the graph, eventually all nodes hold clusters converging to x. □

Boyd et al. [3] analyzed the convergence of weight based average aggregation. The following lemma can be directly derived from their results:

Lemma 8 *In an infinite run of Algorithm 1, after t_{dist}, at every node, the relative weight of clusters converging to a destination x converges to the relative weight of x (in the destination clustering).*

We are now ready to prove the main result of this section.

Theorem 1 *Algorithm 1, with any implementation of the functions* `valToSummary`, `partition` *and* `mergeSet` *that conforms to Requirements R1–R4, solves the Distributed clustering Problem (Definition 4).*

Proof Corollary 1 shows that pool of all clusters in the system converges to some clustering *dest*, i.e., there exist mappings ψ_t from clusters in the pool to the elements in *dest*, as in Definition 3. Lemma 7 shows that there exists a time t_{dist}, after which each node obtains at least one cluster that converges to each destination.

After this time, for each node, the mappings ψ_t from the clusters of the node at t to the *dest* clusters show convergence of the node's clustering to the clustering *dest* (of all input values). Corollary 1 shows that the summaries converge to the destinations, and Lemma 8 shows that the relative weight of all clusters that are mapped to a certain cluster x in *dest* converges to the relative weight of x.

7 Conclusion

We address the problem of distributed data clustering, where nodes obtain values and must calculate a clustering thereof. We presented a generic distributed data clustering algorithm that solves the problem efficiently by employing adaptive in-network compression. The algorithm is completely generic and captures a wide range of algorithms for various instances of the problem. We presented a specific instance thereof—the Gaussian Mixture algorithm, where clusters are maintained as weighted Gaussians, and merging decisions are done using the Expectation Maximization heuristic. Finally, we provided a proof that any implementation of the algorithm converges.

Acknowledgments We thank Yoram Moses and Nathaniel Azuelos for their valuable advice. This work was partially supported by the Technion Funds for Security Research, by the Israeli Ministry of Industry, Trade and Labor Magnet Consortium, and by European Union Grant No. 216366.

Appendix A: Decreasing reference angle

We prove Lemma 2, showing that the sum of two vectors results in a vector with a reference angle not larger than those of the original vectors. The proof considers the 3-dimensional space spanned by the two summed vectors and the i'th axis. We show in Lemma 9 that it is sufficient to consider the angles of the two vectors in a 2-dimensional space they span.

Recall we denote by $\|v\|_p$ the L^p norm of v. For simplicity, we denote the Euclidean (L^2) norm by $\|v\|$. Denote by $v_1 \cdot v_2$ the scalar product of the vectors v_1 and v_2. Then the angle between two vectors in the mixture space is:

$$\arccos\left(\frac{v_a \cdot v_b}{\|v_a\| \cdot \|v_b\|}\right)$$

We now show that we may prove for 2-dimensions rather than 3:

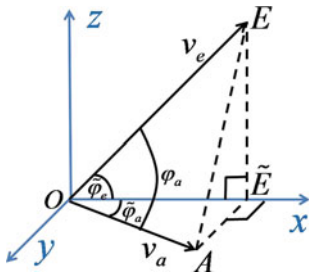


Fig. 6 The angles of the vectors v_a and v_e

Lemma 9 (Reduction to 2 dimensions) *In a 3 dimensional space, let v_a and v_b be two vectors lying on the XY plane with angles not larger than $\pi/2$ with the X axis, and v_a's angle with the X axis is larger than that of v_b. Let v_e be a vector in the XZ plane whose angle with the X axis is smaller than $\pi/2$ and with the Z axis not larger than $\pi/2$. Then v_b's angle with v_e is smaller than that of v_a.*

Proof Let us express the angle of the vector v_a on the XY plane with v_e using the angle of the vector with the X axis, i.e., with the projection of v_e on the XY plane, as shown in Fig. 6. Denote the end point of the vector by A, and the origin by O. Construct a perpendicular line to the X axis passing through A. Denote the point of intersection $\tilde{E}$. From $\tilde{E}$ take a perpendicular line to the XY plane, until intersecting v_e. Denote that intersection point E. OE is the vector e_i and $O\tilde{E}$ is its projection on the XY axis. Denote the angle AOE by φ_a and $AO\tilde{E}$ by $\tilde{\varphi}_a$. Denote the angle $EO\tilde{E}$ by $\tilde{\varphi}_e$.

$$\begin{aligned}
O\tilde{E} &= |v| \cos\tilde{\varphi}_a \\
OE &= \frac{O\tilde{E}}{\cos\tilde{\varphi}_e} = \frac{|v|\cos\tilde{\varphi}_a}{\cos\tilde{\varphi}_e} \\
E\tilde{E} &= O\tilde{E}\tan\tilde{\varphi}_e = |v|\cos\tilde{\varphi}_a\tan\tilde{\varphi}_e \\
A\tilde{E} &= |v|\sin\tilde{\varphi}_a \\
AE &= \sqrt{E\tilde{E}^2 + A\tilde{E}^2} = \sqrt{(|v|\cos\tilde{\varphi}_a\tan\tilde{\varphi}_e)^2 + (|v|\sin\tilde{\varphi}_a)^2}
\end{aligned}$$

Now we can use the law of cosines to obtain:

$$\varphi_a = \arccos\frac{OA^2 + OE^2 - AE^2}{2 \cdot OA \cdot OE} = \arccos(\cos\tilde{\varphi}_a\cos\tilde{\varphi}_e) \tag{3}$$

Since $0 \le \tilde{\varphi}_a \le \pi/2$ and $0 \le \tilde{\varphi}_e \le \pi/2$, we see that φ_a is monotonically increasing with $\tilde{\varphi}_a$. We use similar notation for the vector b, and since $\tilde{\varphi}_b < \tilde{\varphi}_a$, and both are smaller than $\pi/2$, then:

$$\begin{aligned}
&\cos\tilde{\varphi}_a \le \cos\tilde{\varphi}_b \\
&\cos\tilde{\varphi}_a\cos\tilde{\varphi}_e \le \cos\tilde{\varphi}_b\cos\tilde{\varphi}_e \\
&\arccos(\cos\tilde{\varphi}_a\cos\tilde{\varphi}_e) \ge \arccos(\cos\tilde{\varphi}_b\cos\tilde{\varphi}_e) \\
&\varphi_a \ge \varphi_b
\end{aligned} \tag{4}$$

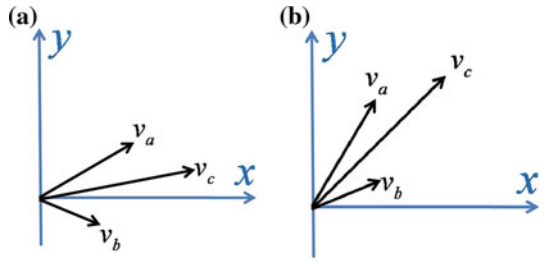


Fig. 7 The possible constructions of two vectors v_a and v_b and their sum v_c, s.t. their angles with the X axis are smaller than $\pi/2$ and v_a's angle is larger than v_b's angle

Now we return to the n dimensional mixture space.

Lemma 2 (restated) *The sum of two vectors in the mixture space is a vector with a smaller i'th reference angle than the larger i'th reference angle of the two, for any* $1 \leq i \leq n$.

Proof Denote the two vectors v_a and v_b, and their i'th reference angles φ_i^a and φ_i^b, respectively. Assume without loss of generality that $\varphi_i^a \geq \varphi_i^b$. Denote the sum vector by v_c

It is sufficient to prove the above in the 3 dimensional space spanned by v_a, v_b and e_i. Align the XYZ axes such that v_a and v_b lie on the XY plane and the projection of e_i on that plane is on the X axis. The vector v_c lies on the XY plane, as it is a linear combination of two vectors on the plane.

By Lemma 9, It is sufficient to show the angle of v_c with the projection of the reference vector is smaller than the angle of v_a with the projection.

The angle between v_c and the X axis is smaller than v_a's angle with it. The only two possible constructions are shown in Fig. 7. □

Appendix B: ε' exists

Lemma 4 (restated) *For any* ε, *there exists an* $\varepsilon' < \varepsilon$ *such that if a vector* v_{out} *lies outside the* ε*-neighborhood of* $\hat{\varphi}_{i,max}$ *(i.e., has a reference angle smaller than* $\hat{\varphi}_{i,max} - \varepsilon$*), and a vector* v_{in} *lies inside the* ε'*-neighborhood (i.e., has a reference angle larger than* $\hat{\varphi}_{i,max} - \varepsilon'$*), then their sum* v_{sum} *lies outside the* ε' *neighborhood.*

Proof Consider the 3 dimensional space spanned by v_{in}, v_{out} and e_i. Align the XYZ axes such that v_{in} and v_{out} lie on the XY plane and the projection of e_i on that plane is aligned with the X axis. Denote this projection by $\tilde{e}_i$. v_{sum} lies on the XY plane, as it is a linear combination of two vectors on the plane. Denote by $\tilde{\varphi}^i_{\text{in}}$, $\tilde{\varphi}^i_{\text{out}}$ and $\tilde{\varphi}^i_{\text{sum}}$ the angles between $\tilde{e}_i$ and the vectors v_{in}, v_{out} and v_{sum}, respectively. Denote by $\tilde{\varphi}^i_{e_i}$ the angle between e_i and its projection $\tilde{e}_i$.

Consider some $\varepsilon' \leq \varepsilon/2$, so that the angle between v_{in} and v_{out} is at least $\varepsilon/2$. Notice that the L^2 norms of v_{in} and v_{out} are bounded between q from below and $\sqrt{s}$ from above. Observing Fig. 7 again, we deduce that there is a lower bound on the difference between the angles:

$$\tilde{\varphi}^i_{\text{sum}} < \tilde{\varphi}^i_{\text{in}} - x_1 \tag{5}$$

Due to the previous bound, and noting that all angles are not larger than $\pi/2$, a constant x_2 exists such that

$$\cos\tilde{\varphi}^i_{\text{in}} < \cos\tilde{\varphi}^i_{\text{sum}} - x_2 \ . \tag{6}$$

Since the reference angles of v_{in} and v_{out} are different, at least one of them is smaller than $\pi/2$, therefore $\tilde{\varphi}^i_{e_i} < \pi/2$ for any such couple. Therefore, $\cos\tilde{\varphi}^i_{e_i}$ is a bounded size, and factoring Inequality 6 we deduce that there exists a constant x_3 such that

$$\cos\tilde{\varphi}^i_{\text{in}}\cos\tilde{\varphi}^i_{e_i} < \cos\tilde{\varphi}^i_{\text{sum}}\cos\tilde{\varphi}^i_{e_i} - x_3 \ . \tag{7}$$

We use the inverse cosine function with Inequality 7 to finally deduce that there exists a constant x_4 such that

$$\arccos(\cos\tilde{\varphi}^i_{\text{in}}\cos\tilde{\varphi}^i_{e_i}) > \arccos(\cos\tilde{\varphi}^i_{\text{sum}}\cos\tilde{\varphi}^i_{e_i}) + x_4 \tag{8}$$

$$\varphi^i_{\text{in}} > \varphi^i_{\text{sum}} + x_4 \tag{9}$$

Therefore, for a given ε, we choose

$$\varepsilon' < \min\left\{\frac{1}{2}x_4, \frac{1}{2}\varepsilon\right\}.$$

With this ε', we obtain $\varphi^i_{\text{sum}} < \hat{\varphi}_{i,\max} - \varepsilon'$, as needed.

References

1. Asada, G., Dong, M., Lin, T., Newberg, F., Pottie, G., Kaiser, W., Marcy, H.: Wireless integrated network sensors: low power systems on a chip. In: ESSCIRC, Elsevier, Den Hague (1998)
2. Birk, Y., Liss, L., Schuster, A., Wolff, R.: A local algorithm for ad hoc majority voting via charge fusion. In: DISC, Springer, Heidelberg (2004)
3. Boyd, S.P., Ghosh, A., Prabhakar, B., Shah, D.: Gossip algorithms: design, analysis and applications. In: INFOCOM, IEEE, Miami (2005)
4. Datta, S., Giannella, C., Kargupta, H.: K-means clustering over a large, dynamic network. In: SDM, SIAM (2006)
5. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the em algorithm. J. Royal Stat. Soc. **39**(1) 1–38 (1977). http://www.jstor.org/stable/2984875
6. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification, 2nd edn. Wiley-Interscience, New York (2000)
7. Eugster, P.T., Guerraoui, R., Handurukande, S.B., Kouznetsov, P., Kermarrec, A.-M.: Lightweight probabilistic broadcast. ACM Trans. Comput. Syst. **21**(4):341–374 (2003)
8. Eyal, I., Keidar, I., Rom, R.: Distributed clustering for robust aggregation in large networks. In: HotDep, IEEE (2009)
9. Eyal, I., Keidar, I., Rom, R.: Distributed data classification in sensor networks. In: PODC, ACM (2010)
10. Flajolet, P., Martin, G.N.: Probabilistic counting algorithms for data base applications. J. Comput. Syst. Sci. **31**(2) 182–209 (1985)
11. Gurevich, M., Keidar, I.: Correctness of gossip-based membership under message loss. SIAM J. Comput. **39**(8), 3830–3859 (2010)

12. Haridasan, M., van Renesse, R.: Gossip-based distribution estimation in peer-to-peer networks. In: International Workshop on Peer-to-Peer Systems (IPTPS 08) (2008)
13. Heller, J.:: Catch-22. Simon & Schuster, New York (1961)
14. Kempe, D., Dobra, A., Gehrke, J.: Gossip-based computation of aggregate information. In: FOCS, IEEE Computer Society, Los Alamitos (2003)
15. Kowalczyk, W., Vlassis, N.A.: Newscast em. In: NIPS (2004)
16. Macqueen, J.B.: Some methods of classification and analysis of multivariate observations. In: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability (1967)
17. Mark Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A.M., van Steen, M.: Gossip based peer sampling. ACM Trans. Comput. Syst. **25**(3) (2007)
18. Nath, S., Gibbons, P.B., Seshan, S., Anderson, Z.R.: Synopsis diffusion for robust aggregation in sensor networks. In: SenSys, ACM, New York (2004)
19. Sacha, J., Napper, J., Stratan, C., Pierre, G.: Reliable distribution estimation in decentralised environments. Submitted for Publication (2009)
20. Salmond, D.J.: Mixture reduction algorithms for uncertain tracking. Tech. rep., RAE Farnborough (UK) (1988)
21. Warneke, B., Last, M., Liebowitz, B., Pister, K.: Smart dust: communicating with a cubic-millimeter computer. Computer **34**(1) (2001). doi:10.1109/2.895117