Contents

1 Clustering for Monitoring Distributed Data Streams 1
Maria Barouti, Daniel Keren, Jacob Kogan and Yaakov Malinovsky
1.1 Introduction 2
1.2 Text Mining application 4
1.3 Monitoring threshold functions through clustering: motivation 5
1.4 Monitoring threshold functions through clustering: implementation 8
1.5 Experimental Results 11
1.5.1 Data 11
1.5.2 Monitoring with Incremental Clustering 13
1.6 Conventional Clustering Algorithms 15
1.6.1 PDDP 15
1.6.2 Batch $k-$means 17
1.6.3 Incremental $k-$means 18
1.6.4 Batch $k-$means followed by incremental $k-$means 19
1.6.5 Node Clustering with Classical Clustering Algorithms . . . 19
1.7 Conclusions 20
Appendix 1: First and Second Moments 21
Appendix 2: Broadcast Count 24
References 26

List of Contributors

Maria Barouti
Math. and Stat, UMBC, Baltimore, MD 21250, USA, e-mail: bmaria2@umbc.edu

Daniel Keren
Department of Computer Science, Haifa University, Haifa 31905, Israel, e-mail: dkeren@cs.haifa.ac.il

Jacob Kogan
Math. and Stat, UMBC, Baltimore, MD 21250, USA, e-mail: kogan@umbc.edu

Yaakov Malinovsky
Math. and Stat, UMBC, Baltimore, MD 21250, USA, e-mail: yaakovm@umbc.edu

Chapter 1
Clustering for Monitoring Distributed Data Streams

Maria Barouti, Daniel Keren, Jacob Kogan and Yaakov Malinovsky

`abstract` Monitoring data streams in a distributed system is a challenging problem with profound applications. The task of feature selection (e.g., by monitoring the information gain of various features) is an example of an application that requires special techniques to avoid a very high communication overhead when performed using straightforward centralized algorithms.

Motivated by recent contributions based on geometric ideas, we present an alternative approach that combines system theory techniques and clustering. The proposed approach enables monitoring values of an arbitrary threshold function over distributed data streams through a set of constraints applied independently on each stream and/or clusters of streams. The clusters are designed to evolve in time and to adapt themselves to the data stream. A correct choice of clusters yields a reduction in communication load. Unlike many clustering algorithms that attempt to collect together similar data items, monitoring requires clusters with *dissimilar* vectors canceling each other as much as possible. In particular, sub–clusters of a good cluster do not have to be good. This novel type of clustering dictated by the problem at hand requires development of new algorithms and/or modification of the existing ones, and the chapter is a step in this direction.

We report experiments on real-world data with a newly devised clustering algorithm. The experiments detect instances where communication between nodes is required, and show that the clustering approach reduces communication load. We

Maria Barouti
Math. and Stat, UMBC, Baltimore, MD 21250, USA, e-mail: bmaria2@umbc.edu

Daniel Keren
Department of Computer Science, Haifa University, Haifa 31905, Israel, e-mail: dkeren@cs.haifa.ac.il

Jacob Kogan
Math. and Stat, UMBC, Baltimore, MD 21250, USA, e-mail: kogan@umbc.edu

Yaakov Malinovsky
Math. and Stat, UMBC, Baltimore, MD 21250, USA, e-mail: yaakovm@umbc.edu

then propose an application of the well known clustering algorithms to the monitoring problem.

1.1 Introduction

In many emerging applications one needs to process a continuous stream of data in real time. Sensor networks [19], network monitoring [14], and real–time analysis of financial data [28], [29] are examples of such applications. Monitoring queries are a particular class of queries in the context of data streams. Previous work in this area deals with monitoring simple aggregates [14], or term frequency occurrence in a set of distributed streams [25]. The current contribution is motivated by results recently reported in [20], [21] where a more general type of monitoring query is described as follows:

Let $\mathbf{S} = \{\mathbf{s}_1, \ldots, \mathbf{s}_n\}$ be a set of data streams collected at n nodes $\mathbf{N} = \{\mathbf{n}_1, \ldots, \mathbf{n}_n\}$. Let $\mathbf{v}_1(t), \ldots, \mathbf{v}_n(t)$ be d-dimensional, real-valued, time varying vectors derived from the streams. For a function $f : \mathbf{R}^d \to \mathbf{R}$ we would like to monitor the inequality

$$f\left(\frac{\mathbf{v}_1(t) + \ldots + \mathbf{v}_n(t)}{n}\right) > 0 \tag{1.1}$$

while minimizing communication between the nodes. Often the threshold might be a constant r other than 0. In what follows, for notational convenience, we shall always consider the inequality $f > 0$, and when one is interested in monitoring the inequality $f > r$ we will modify the threshold function and consider $g = f - r$, so that the inequality $g > 0$ yields $f > r$. In e.g. [20, 17, 18, 9] a few real-life applications of this monitoring problem are described; see also Section 1.2 here.

The difference between monitoring problems involving linear and non-linear functions f is discussed and illustrated by a simple example involving a quadratic function f in [20]. The example demonstrates that, for a non-linear f, it is often very difficult to determine from the values of f at the nodes whether its value evaluated at the average vector is above the threshold or not. The present chapter deals with the information gain function (see Section 1.2 for details), and rather than focus on the values of f we consider the location of the vectors $\mathbf{v}_i(t)$ relative to the boundary of the the subset of $\mathbf{R}^d$ where f is positive. We denote this set by $\mathbf{Z}_+(f) = \{\mathbf{v} : f(\mathbf{v}) > 0\}$, and state (1.1) as

$$\mathbf{v}(t) = \frac{\mathbf{v}_1(t) + \ldots + \mathbf{v}_n(t)}{n} \in \mathbf{Z}_+(f). \tag{1.2}$$

thus, the functional monitoring problem is transformed to the monitoring of a *geometric* condition. As a simple illustration, consider the case of three scalar functions $v_1(t)$, $v_2(t)$ and $v_3(t)$, and the identity function f (i.e. $f(x) = x$). We would like to monitor the inequality

$$v(t) = \frac{v_1(t) + v_2(t) + v_3(t)}{3} > 0$$

while keeping the nodes silent as long as possible. One strategy is to verify the initial inequality $v(t_0) = \frac{v_1(t_0) + v_2(t_0) + v_3(t_0)}{3} > 0$ and to keep the nodes silent while

$$|v_i(t) - v_i(t_0)| < \delta = v(t_0),\ t \geq t_0,\ i = 1,2,3.$$

The first time t when one of the functions, say $v_1(t)$, crosses the boundary of the local constraint, i.e. $|v_1(t) - v_1(t_0)| \geq \delta$ the nodes communicate, t_1 is set to be t, the mean $v(t_1)$ is computed, the local constraint δ is updated and made available to the nodes. The nodes are kept silent as long as the inequalities

$$|v_i(t) - v_i(t_1)| < \delta,\ t \geq t_1,\ i = 1,2,3$$

hold. This type of monitoring was suggested in [23] for a variety of vector norms. The numerical experiments conducted in [23] with the dataset described in Subsection 1.5.1 show that:

1. The number of time instances the mean violates (1.1) is a small fraction ($< 1\%$) of the number of time instances when the local constraint is violated at the nodes.
2. The lion's share of communications (about 75%) is required because of a single node violation of the local constraint δ.
3. The smallest number of communications is required when one uses the l_1 norm.

We note that if, for example, the local constraint is violated at $\mathbf{n}_1$, i.e. $|v_1(t) - v_1(t_0)| \geq \delta$, and at the same time

$$v_1(t) - v_1(t_0) = -[v_2(t) - v_2(t_0)],$$

while $|v_3(t) - v_3(t_0)| < \delta$ then $|v(t) - v(t_0)| < \delta$, $f(v(t)) > 0$, and update of the mean can be avoided. Separate monitoring of the two node cluster $\{\mathbf{n}_1, \mathbf{n}_2\}$ would require communication involving two nodes only, and could reduce communication load. We aim to extend this idea to the general case – involving many nodes, arbitrary functions, and high-dimensional data.

Clustering in general is a difficult problem, and many clustering problems are known to be NP-complete [8]. Unlike standard clustering that attempts to collect together similar data items [4], we are seeking clusters with *dissimilar data items*, which cancel out each other as much as possible. While sub-clusters of a "classical" good cluster are usually good, this may not be the case when a cluster contains dissimilar objects. These observations indicate that a straightforward application of common clustering methods to our problem is not possible.

A basic attempt to cluster nodes was suggested in [24] with results reported for the dataset presented in Subsection 1.5.1. Clustering together just two nodes reported in [24] reduces communication by about 10%.

In this chapter we advance clustering approach to monitoring. The main contribution of this work is twofold:

1. We suggest a specific clustering strategy applicable to a variety of vector norms, and report the communication reduction achieved when the proposed strategy is applied with l_1, l_2, and l_∞ norms.
2. We suggest an application of well known clustering algorithms to monitoring.

The chapter is organized as follows. In Section 1.2 we present a relevant Text Mining application. Section 1.3 provides motivation for node clustering. A specific implementation of node clustering is presented in Section 1.4. Experimental results are reported in Section 1.5. In Section 1.6 we discuss a possible application of classical clustering algorithms to monitoring. Section 1.7 concludes the chapter and indicates new research directions. Appendix 1 summarizes some useful properties of the first and second moments. Appendix 2 details the accounting of message transmission.

In the next section we provide a Text Mining related example that leads to a non linear threshold function f.

1.2 Text Mining application

Let $\mathbf{T}$ be a textual database (for example a collection of mail or news items). We denote the size of the set $\mathbf{T}$ by $|\mathbf{T}|$. We will be concerned with two subsets of $\mathbf{T}$:

1. $\mathbf{R}$–the set of "relevant" texts (e.g. texts not labeled as "spam"),
2. $\mathbf{F}$–the set of texts that contain a "feature" (word or term for example).

We denote complements of the sets by $\overline{\mathbf{R}}$, $\overline{\mathbf{F}}$ respectively (i.e. $\mathbf{R} \cup \overline{\mathbf{R}} = \mathbf{F} \cup \overline{\mathbf{F}} = \mathbf{T}$), and consider the relative size of the four sets $\mathbf{F} \cap \overline{\mathbf{R}}$, $\mathbf{F} \cap \mathbf{R}$, $\overline{\mathbf{F}} \cap \overline{\mathbf{R}}$, and $\overline{\mathbf{F}} \cap \mathbf{R}$ as follows:

$$\begin{aligned} x_{11}(\mathbf{T}) &= \frac{|\mathbf{F} \cap \overline{\mathbf{R}}|}{|\mathbf{T}|},\ x_{12}(\mathbf{T}) = \frac{|\mathbf{F} \cap \mathbf{R}|}{|\mathbf{T}|}, \\ x_{21}(\mathbf{T}) &= \frac{|\overline{\mathbf{F}} \cap \overline{\mathbf{R}}|}{|\mathbf{T}|},\ x_{22}(\mathbf{T}) = \frac{|\overline{\mathbf{F}} \cap \mathbf{R}|}{|\mathbf{T}|}. \end{aligned} \tag{1.3}$$

Note that

$$0 \le x_{ij} \le 1, \text{ and } x_{11} + x_{12} + x_{21} + x_{22} = 1. \tag{1.4}$$

The function f is defined on the simplex (i.e. $x_{ij} \ge 0$, $\sum x_{ij} = 1$), and given by

$$\sum_{i,j} x_{ij} \log \left(\frac{x_{ij}}{(x_{i1} + x_{i2})(x_{1j} + x_{2j})} \right), \tag{1.5}$$

where $\log x = \log_2 x$ throughout the chapter. It is well-known that (1.5) provides the *information gain* for the "feature" (see e.g. [2]).

As an example, we consider n agents installed on n different servers, and a stream of texts arriving at the servers. Let $\mathbf{T}_h = \{\mathbf{t}_{h1}, \ldots, \mathbf{t}_{hw}\}$ be the last w texts received at the h^{th} server, with $\mathbf{T} = \bigcup_{h=1}^{n} \mathbf{T}_h$. Note that

$$x_{ij}(\mathbf{T}) = \sum_{h=1}^{n} \frac{|\mathbf{T}_h|}{|\mathbf{T}|} x_{ij}(\mathbf{T}_h),$$

i.e., entries of the global contingency table $\{x_{ij}(\mathbf{T})\}$ are the weighted average of the local contingency tables $\{x_{ij}(\mathbf{T}_h)\}$, $h = 1, \ldots, n$.

To check that the given "feature" is sufficiently informative with respect to the target relevance label r, one may want to monitor the inequality

$$f\left(x_{11}(\mathbf{T}), x_{12}(\mathbf{T}), x_{21}(\mathbf{T}), x_{22}(\mathbf{T})\right) - r > 0 \tag{1.6}$$

with f given by (1.5) while minimizing communication between the servers.

We next provide arguments in support of clustering for monitoring distributed data streams.

1.3 Monitoring threshold functions through clustering: motivation

In what follows we denote a norm of a vector $\mathbf{v}$ by $\|\mathbf{v}\|$. While the experiments reported in this chapter have been conducted with l_1, l_2, and l_∞ norms, the proposed monitoring and node clustering procedures can be applied with any norm. We shall identify a specific norm used when needed. For a vector set $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \subset \mathbf{R}^d$ we denote the arithmetic mean $\dfrac{\mathbf{x}_1 + \ldots + \mathbf{x}_m}{m}$ of the set by $\boldsymbol{\mu}(\mathbf{X})$. With slight abuse of notations the central second moment $\sum_{i=1}^{m} (\mathbf{x}_i - \boldsymbol{\mu}(\mathbf{X}))^T (\mathbf{x}_i - \boldsymbol{\mu}(\mathbf{X}))$ is denoted by $\sigma^2(\mathbf{X})$. The zero set $\{\mathbf{v} : \mathbf{v} \in \mathbf{R}^d, f(\mathbf{v}) = 0\}$ of a function f is denoted by $\mathbf{Z}_f$.

Our aim is to monitor data streams with as little communication as possible over a sequence of discrete time instances that we shall denote by t. The time instances that require communication between nodes are denoted by t_i, $i = 0, 1, 2, \ldots$. The approach suggested in this chapter builds on the monitoring strategy proposed in [23] and briefly described as follows:

Algorithm 1.3.1 *Monitoring Threshold Function*

- *A node is designated as a root* $\mathbf{r}$.
- *The root sets* $i = 0$.
- *Until end of stream*

  1. *The root sends a request to each node* $\mathbf{n}$ *for the vectors* $\mathbf{v}_\mathbf{n}(t_i)$. *The nodes respond to the root. The root computes the distance* δ *between the mean* $\dfrac{1}{n} \sum_{\mathbf{n} \in \mathbf{N}} \mathbf{v}_n(t_i)$ *and the zero set* $\mathbf{Z}_f$ *of the function* f. *The root transmits* δ *to each node.*
  2. *do for each* $\mathbf{n} \in \mathbf{N}$
     If $||\mathbf{v}_\mathbf{n}(t) - \mathbf{v}_\mathbf{n}(t_i)|| < \delta$

the node **n** *is silent*
else
n *notifies the root about the violation of its local constraint* δ
the root sets $i = i + 1$
go to Step 1.
endif

- *Stop*

An application of the above procedure to data streams generated from the Reuters Corpus RCV1–V2 (see Section 1.5 for detailed description of the data and experiments) leads to 4006 time instances in which the local constraints are violated, and the root is updated. Results presented in Table 1.1 show that in 3034 out of 4006

Table 1.1 number of local constraint violations simultaneously by k nodes, $r = 0.0025$, l_2 norm, the feature is "bosnia"

# of nodes violators	1	2	3	4	5	6	7	8	9	10
# of violation instances	3034	620	162	70	38	26	34	17	5	0

time instances, communications with the root are triggered by constraint violations at exactly one node.

The results immediately suggest to cluster nodes to further reduce communication load. Each cluster will be equipped with a "coordinator" **c** (one of the cluster's nodes). If a cluster node **n** violates its local constraint at time t, then the coordinator collects vectors $\mathbf{v_n}(t) - \mathbf{v_n}(t_i)$ from all the nodes in the cluster, computes the mean of the vectors, and checks whether the mean violates the coordinator constraint δ (at this point, node and coordinator constraints are identical, see Section 1.4 for discussion pertaining to constraints). We shall follow [20] and refer to this step as "the balancing process." If the coordinator constraint is violated, the coordinator alerts the root, and the mean of the entire dataset is recomputed by the root (for detailed description of the procedure see Section 1.4).

For the problem at hand we would like to partition the set of nodes **N** into k clusters $\Pi = \{\pi_1, \ldots, \pi_k\}$ so that

$$\mathbf{N} = \bigcup_{i=1}^{k} \pi_i, \text{ and } \pi_i \bigcap \pi_j = \emptyset \text{ if } i \neq j.$$

We denote the size of π_i by $|\pi_i|$. If for each cluster π_i one has

$$\frac{1}{|\pi_i|} \left\| \sum_{\mathbf{n} \in \pi_i} [\mathbf{v_n}(t) - \mathbf{v_n}(t_j)] \right\| < \delta,$$

then, due to convexity of any norm, one has

$$\left\| \frac{1}{n} \sum_{\mathbf{n}\in\mathbf{N}} \mathbf{v}_{\mathbf{n}}(t) - \frac{1}{n} \sum_{\mathbf{n}\in\mathbf{N}} \mathbf{v}_{\mathbf{n}}(t_j) \right\| \leq \sum_{i=1}^{k} \frac{|\pi_i|}{n} \left[\frac{1}{|\pi_i|} \left\| \sum_{\mathbf{n}\in\pi_i} [\mathbf{v}_{\mathbf{n}}(t) - \mathbf{v}_{\mathbf{n}}(t_j)] \right\| \right] < \delta.$$

Hence the "new" mean $\frac{1}{n} \sum_{\mathbf{n}\in\mathbf{N}} \mathbf{v}_{\mathbf{n}}(t)$ belongs to $\mathbf{Z}_+(f)$ if the "old" mean $\frac{1}{n} \sum_{\mathbf{n}\in\mathbf{N}} \mathbf{v}_{\mathbf{n}}(t_j)$ belongs to this set. We therefore may attempt to define the quality of a k cluster partition Π as

$$Q(\Pi) = \max_i \left\{ \frac{1}{|\pi_i|} \left\| \sum_{\mathbf{n}\in\pi_i} [\mathbf{v}_{\mathbf{n}}(t) - \mathbf{v}_{\mathbf{n}}(t_j)] \right\| \right\}, \; i = 1, \ldots, k. \tag{1.7}$$

Our aim is to identify k and a k cluster partition Π^o that **minimizes** (1.7). The monitoring problem requires to assign nodes $\{\mathbf{n}_{i_1}, \ldots, \mathbf{n}_{i_k}\}$ to the same cluster π so that the total average change within cluster π

$$\left\| \frac{1}{|\pi|} \sum_{\mathbf{n}\in\pi} [\mathbf{v}_{\mathbf{n}}(t) - \mathbf{v}_{\mathbf{n}}(t_j)] \right\| \text{ for } t > t_j$$

is minimized, i.e., nodes with **different** variations $\mathbf{v}_{\mathbf{n}}(t) - \mathbf{v}_{\mathbf{n}}(t_j)$ that cancel out each other as much as possible are assigned to the same cluster.

A standard clustering problem is often described as "... finding and describing cohesive or homogeneous chunks in data, the clusters" (see e.g. [4]). Unlike classical clustering procedures, this one needs to combine "dissimilar" nodes together.

The proposed partition quality $Q(\Pi)$ (see (1.7)) generates three immediate problems:

1. Since the arithmetic mean $\overline{a}$ of a finite set of real numbers $\{a_1, \ldots, a_k\}$ satisfies

   $$\min\{a_1, \ldots, a_k\} \leq \overline{a} \leq \max\{a_1, \ldots, a_k\}$$

   the single cluster partition always minimizes $Q(\Pi)$. Considering the entire set of nodes as a single cluster with its own coordinator that communicates with the root introduces an additional unnecessary "bureaucracy" layer that only increases communications. We seek a trade-off which yields clusters with "good" sizes (this is rigorously defined in Section 1.4).
2. Computation of $Q(\Pi)$ involves future values $\mathbf{v}_{\mathbf{n}}(t)$, which are not available at time t_j when the clustering is performed.
3. Since the communication overhead of the balancing process is proportional to the size of a cluster, the individual clusters' sizes should affect the clustering quality $q(\pi)$.

In the next section we address these problems.

1.4 Monitoring threshold functions through clustering: implementation

We argue that in addition to the average magnitude of the variations $\mathbf{v_n}(t) - \mathbf{v_n}(t_j)$ inside the cluster π, the cluster's size also affects the frequency of updates, and, as a result, the communication load. We therefore define the quality of the cluster π by

$$q(\pi) = \frac{1}{|\pi|}\left|\left|\sum_{\mathbf{n}\in\pi}[\mathbf{v_n}(t) - \mathbf{v_n}(t_j)]\right|\right| + \alpha|\pi|, \tag{1.8}$$

where α is a nonnegative scalar parameter. The quality of the partition $\Pi = \{\pi_1, \ldots, \pi_k\}$ is defined by

$$Q(\Pi) = \max_{i\in\{1,\ldots,k\}} q(\pi_i), \tag{1.9}$$

When $\alpha = 0$ the partition that minimizes $Q(\Pi)$ is a single cluster partition (that we would like to avoid). When $\max_{\mathbf{n}} \left|\left|\mathbf{v_n}(t) - \mathbf{v_n}(t_j)\right|\right| \leq \alpha$ the optimal partition is made up of n singleton clusters. In this chapter we focus on

$$0 < \alpha < \max_{\mathbf{n}\in\mathbf{N}} \left|\left|\mathbf{v_n}(t) - \mathbf{v_n}(t_j)\right|\right|. \tag{1.10}$$

The constant α depends on t and t_j, and below we show how to avoid this dependence.

Computation of $Q(\Pi)$ required for the clustering procedure is described below. In order to compute $Q(\Pi)$ at time t_j one needs to know $\mathbf{v_n}(t)$ at a future time $t > t_j$ which is not available. While the future behavior is not known, we shall use past values of $\mathbf{v_n}(t)$ for prediction. For each node $\mathbf{n}$ we build "history" vectors $\mathbf{h_n}(t_j)$ defined as follows:

1. $\mathbf{h_n}(t_0) = 0$
2. if ($\mathbf{h_n}(t_j)$ is already available)
   $\mathbf{h_n}(t_{j+1}) = \mathbf{h_n}(t_j)$
   for t increasing from t_j to t_{j+1} do
   $$\mathbf{h_n}(t_{j+1}) = \frac{1}{2}\mathbf{h_n}(t_{j+1}) + [\mathbf{v_n}(t) - \mathbf{v_n}(t_j)]$$

The vectors $\mathbf{h_n}(t_j)$ accumulate the history of changes, with older changes assigned smaller weights. We shall use the vectors $\{\mathbf{h_n}(t_j)\}$ to generate a node partition at time t_j. We note that normalization of the vector set that should be clustered does not change the induced optimal partitioning of the nodes. When the vector set is normalized by the magnitude of the longest vector in the set, the range for α conveniently shrinks to $[0,1]$. In what follows we set $h = \max_{\mathbf{n}\in\mathbf{N}} ||\mathbf{h_n}(t_j)||$, assume that $h > 0$, and describe a "greedy" clustering procedure for the normalized vector set

$$\{\mathbf{a}_1, \ldots, \mathbf{a}_n\},\ \mathbf{a}_i = \frac{1}{h}\mathbf{h}_{\mathbf{n}_i}(t_j),\ i = 1, \ldots, n.$$

We start with the n cluster partition Π^n (each cluster is a singleton). We set $k = n$ and loop the following procedure until the number of clusters reduces to $k = 2$.

Algorithm 1.4.1 *(Incremental Clustering)*

- *Set $k = n$.*
- *do until $k > 2$:*
  1. *in partition Π^k identify cluster π_j of maximal quality, i.e.,*

$$q(\pi_j) \geq q(\pi_i) \;\; i \neq j.$$

  2. *identify cluster π_i so that the merger of π_i with π_j produces a cluster of smallest possible quality, i.e.,*

$$q\left(\pi_j \bigcup \pi_i\right) \leq q\left(\pi_j \bigcup \pi_l\right), \; l \neq j,$$

     where cluster's quality is defined by (1.8).
  3. *Build partition Π^{k-1} by merging clusters π_j and π_i.*
  4. *Set $k = k - 1$.*
  5. *go to Step 1.*
- *Stop.*

The final partition is selected from the $n-1$ partitions $\{\Pi^2, \ldots, \Pi^n\}$ as the one that minimizes Q.

Note that node constraints δ do not have to be equal (see Algorithm 1.4.2, Step 2). Taking into account the distribution of the data streams at each node can further reduce communication. We illustrate this statement by a simple example involving two nodes. If, for example, there is reason to believe that the inequality

$$2\|\mathbf{v}_1(t) - \mathbf{v}_1(t_i)\| \leq \|\mathbf{v}_2(t) - \mathbf{v}_2(t_i)\| \tag{1.11}$$

always holds, then the number of node violations may be reduced by imposing node dependent constraints

$$\|\mathbf{v}_1(t) - \mathbf{v}_1(t_i)\| < \delta_1 = \frac{2}{3}\delta, \text{ and } \|\mathbf{v}_2(t) - \mathbf{v}_2(t_i)\| < \delta_2 = \frac{4}{3}\delta$$

so that the wider varying stream at the second node enjoys larger "freedom" of change, while the inequality

$$\left\| \frac{\mathbf{v}_1(t) + \mathbf{v}_2(t)}{2} - \frac{\mathbf{v}_1(t_i) + \mathbf{v}_2(t_i)}{2} \right\| < \frac{\delta_1 + \delta_2}{2} = \delta$$

holds true. Assigning "weighted" local constraints requires information provided by (1.11). With no additional assumptions about the stream data distribution this

information is not available. Unlike [22] we refrain from making assumptions regarding the underlying data distributions; instead, we estimate the weights through past values $\mathbf{v_n}(t)$, $\mathbf{n} \in \mathbf{N}$.

At the initial time t_0 all nodes report their vectors $\mathbf{v_n}(t_0)$ to the root, the root computes the average, and the the distance $\delta(\mathbf{r})$ from the average to the boundary of $\mathbf{Z}_+(f)$. At this point we define $\delta(\mathbf{n}) = \delta(\mathbf{r})$, for each $\mathbf{n} \in \mathbf{N}$.

We now focus on a particular node $\mathbf{n}$. Consider first m time instances $t^1, t^2, \ldots, t^m$ and the vector set

$$\mathbf{V}'_\mathbf{n} = \{\mathbf{v}'_\mathbf{n}(t^1), \ldots, \mathbf{v}'_\mathbf{n}(t^m)\},$$

where

$$\mathbf{v}'_\mathbf{n}(t^m) = \mathbf{v_n}(t^m),\ \mathbf{v}'_\mathbf{n}(t^{m-1}) = \frac{1}{2}\mathbf{v_n}(t^{m-1}),\ \ldots, \mathbf{v}'_\mathbf{n}(t^1) = \frac{1}{2^{m-1}}\mathbf{v_n}(t^1).$$

The node constraint $\delta(\mathbf{n})$ introduced below depends on the arithmetic mean $\boldsymbol{\mu}(\mathbf{V}'_\mathbf{n})$ and the central second moment

$$\sigma^2(\mathbf{V}'_\mathbf{n}) = \sum_{i=0}^{m} \left(\mathbf{v}'_\mathbf{n}(t_i) - \mu(\mathbf{V}'_\mathbf{n})\right)^T \left(\mathbf{v}'_\mathbf{n}(t_i) - \mu(\mathbf{V}'_\mathbf{n})\right)$$

of the node $\mathbf{n}$. We denote $\mu(\mathbf{V}'_\mathbf{n})$ by $\mu_\mathbf{n}$, and $\sigma^2(\mathbf{V}'_\mathbf{n})$ by $\sigma^2_\mathbf{n}$. Since $\left|\left|\mathbf{v}'_\mathbf{n}(t^i) - \mu_\mathbf{n}\right|\right| \leq \sqrt{\frac{m-1}{m}\sigma^2_\mathbf{n}}$, $i = 1, \ldots, m$ (see Appendix 1) we define

$$W_\mathbf{n}(t^m) = W_\mathbf{n} = ||\boldsymbol{\mu}_\mathbf{n}|| + \sqrt{\frac{m-1}{m}\sigma^2_\mathbf{n}}.$$

We note that although the bound $\sqrt{\frac{m-1}{m}\sigma^2_\mathbf{n}}$ may be very conservative, the same conservative criterion is applied uniformly to every node.

If at time t^m the root constraint $\delta(\mathbf{r})$ is updated, each node $\mathbf{n}$ broadcasts $W_\mathbf{n}(t^m) = W_\mathbf{n}$ to the root, the root computes $W = \sum_{\mathbf{n}\in\mathbf{N}} W_\mathbf{n}$, and transmits the updated $\delta(\mathbf{n}) = w_\mathbf{n}\delta(\mathbf{r})$ where $w_\mathbf{n} = n \times \frac{W_\mathbf{n}}{W}$ (so that $\sum_{\mathbf{n}\in\mathbf{N}} w_\mathbf{n} = n$) back to node $\mathbf{n}$. For a coordinator $\mathbf{c}$ of a node cluster π the constraint $\delta(\mathbf{c}) = \frac{1}{|\pi|}\sum_{\mathbf{n}\in\pi}\delta(\mathbf{n})$.

Algorithm 1.4.2 *Monitoring Threshold Function with Clustering*

- *A node is designated as a root* $\mathbf{r}$.
- *The root sets* $i = 0$.
- *Until end of stream*

  1. *The root sends a request to each node* $\mathbf{n}$ *for the vectors* $\mathbf{v_n}(t_i)$. *The nodes respond to the root. The root computes the distance* δ *between the mean*

$\frac{1}{n}\sum_{\mathbf{n}\in\mathbf{N}} \mathbf{v}_n(t_i)$ *and the zero set* $\mathbf{Z}_f$ *of the function* f*. The root transmits* δ *to each node.*

2. *do for each* $\mathbf{n} \in \mathbf{N}$
   If $||\mathbf{v}_\mathbf{n}(t) - \mathbf{v}_\mathbf{n}(t_i)|| < \delta$
   the node $\mathbf{n}$ *is silent*
   else
   $\mathbf{n}$ *notifies the root about the violation of its local constraint* δ
   endif
3. *The root requests vectors* $\mathbf{v}_\mathbf{n}(t_i)$ *and weights* $W_\mathbf{n}(t_i)$.
   The root forms a partition $\Pi = \{\pi_1, \ldots, \pi_k\}$ *and sends node and coordinator constraints* $\delta(\mathbf{n})$ *and* $\delta(\mathbf{c})$ *to nodes and coordinators.*
   The root sets $i = i + 1$.
4. *do for each* $\pi \in \Pi$
   do for each $\mathbf{n} \in \pi$
   If $\delta(\mathbf{n}) \leq ||\mathbf{v}_\mathbf{n}(t) - \mathbf{v}_\mathbf{n}(t_i)||$
   If $\delta(\mathbf{c}) \leq \frac{1}{|\pi|}\left|\left|\sum_{\mathbf{n}\in\pi} \mathbf{v}_\mathbf{n}(t) - \sum_{\mathbf{n}\in\pi} \mathbf{v}_\mathbf{n}(t_i)\right|\right|$
   notify root about coordinator violation
   goto Step 3
   endif
   endif

- *Stop*

Node constraints $\delta(\mathbf{n})$ based on the first moment only are introduced in [6]. In the next section we provide monitoring results for node constraints based on the first and second moments, and compare the results with those reported in [6] as well as monitoring with no clustering reported in [23].

1.5 Experimental Results

This section presents few experimental results of monitoring with clustering. Monitoring algorithms are always applied to the data considered in [20] which is described next.

1.5.1 Data

The data streams analyzed in this section are generated from the Reuters Corpus RCV1–V2[1]. The data consists of 781,265 tokenized documents with document ID

[1] `http://leon.bottou.org/projects/sgd`

ranging from 2651 to 810596. We simulate n streams by arranging the feature vectors in ascending order with respect to document ID, and selecting feature vectors for the stream in the round-robin fashion.

Each document in the Reuters Corpus RCV1–V2 is labeled as belonging to one or more categories. We label a vector as "relevant" if it belongs to the "CORPORATE/INDUSTRIAL" ("CCAT") category, and "spam" otherwise. Following [20] we focus on three features: "bosnia," "ipo," and "febru." Each experiment was performed with 10 nodes, where each node holds a sliding window containing the last 6,700 documents it received.

First we use $67,000$ documents to generate initial sliding windows. The remaining $714,265$ documents are used to generate datastreams, hence the selected feature information gain is computed $714,265$ times. Based on all the documents contained in the sliding window at each one of the $714,266$ time instances we compute and graph $714,266$ information gain values for the feature "bosnia" (see Figure 1.1). For the experiments described below, the threshold value r is predefined, and the goal is to monitor the inequality $f(\mathbf{v}) - r > 0$ while minimizing communication between the nodes. We also assume that new texts arrive simultaneously at each node. We define a broadcast as one time transmission of information between different nodes.

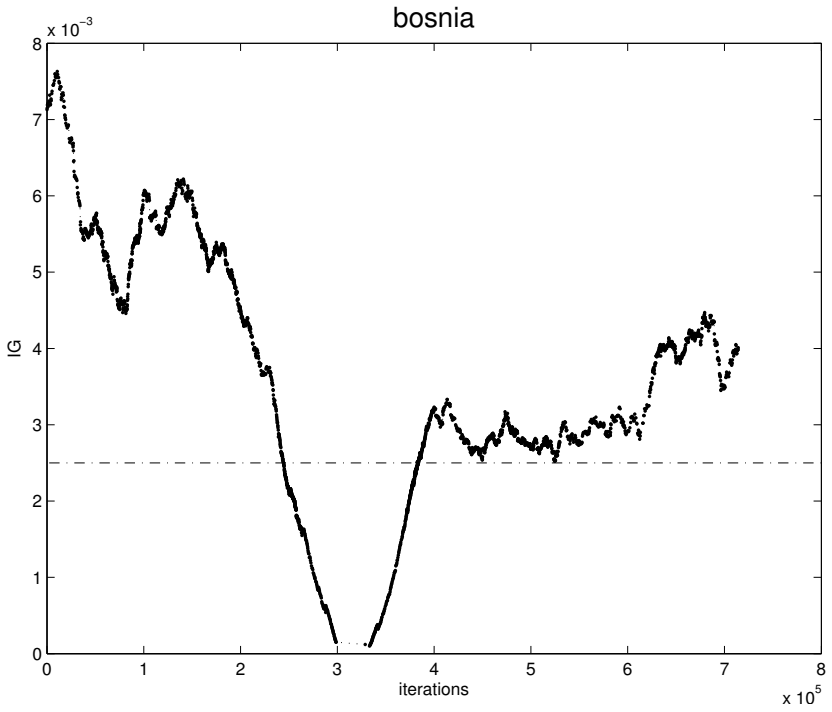

Fig. 1.1 information gain values for the feature "bosnia"

1.5.2 Monitoring with Incremental Clustering

The previous work [6] reported monitoring results obtained with the incremental clustering algorithm (Algorithm 1.4.1) with weights $W_{\mathbf{n}} = \|\boldsymbol{\mu}_{\mathbf{n}}\|$ only. We shall call this implementation of the algorithm "first moment incremental clustering" (FMIC). The clustering algorithm with weights $W_{\mathbf{n}} = ||\boldsymbol{\mu}_{\mathbf{n}}|| + \sqrt{\frac{m-1}{m}\sigma_{\mathbf{n}}^2}$ introduced in this chapter will be referred to as the "second moment incremental clustering" (SMIC). In this section we report and compare results generated by the algorithms for the threshold $r = 0.0025$ and $\alpha = 0.05, 0.10, \ldots, 0.95$.

The best result with respect to α obtained by an application of FMIC to the feature "febru," is presented in Table 1.2. The clustering approach in this case is particularly

Table 1.2 Number of root and coordinator mean computations, and total broadcasts for feature "febru" with threshold $r = 0.0025$ and the "first moment clustering"

norm	best α	root mean update	coordinator mean update	total broadcasts
l_1	0.70	1431	0	38665
l_2	0.80	1317	0	35597
l_∞	0.65	1409	0	38093

successful – coordinators' constraints are not violated, and the root mean updates are decreased significantly as compares, for example, to results reported in [23].

The corresponding results generated by SMIC are provided in Table 1.3. Results in

Table 1.3 Number of root and coordinator mean computations, and total broadcasts for feature "febru" with threshold $r = 0.0025$ and the "second moment clustering"

norm	best α	root mean update	coordinator mean update	total broadcasts
l_1	0.85	883	0	23859
l_2	0.75	833	0	22509
l_∞	0.75	854	0	23076

Table 1.3 show a significant decrease in the number of broadcasts as compared to results in Table 1.2.

Next we turn to the features "ipo" and "bosnia." In both cases we run monitoring with FMIC and SMIC, allowing $\alpha = 0.05, 0.10, \ldots, 0.95$, and report results with

the lowest number of broadcasts. The results obtained for "ipo" with FMIC are presented in Table 1.4. Application of SMIC leads to results provided in Table 1.4.

Table 1.4 Number of root and coordinator mean computations, and total broadcasts for feature "ipo" with threshold $r = 0.0025$ and the "first moment clustering"

norm	best α	root mean update	coordinator mean update	total broadcasts
l_1	0.15	5455	829	217925
l_2	0.10	7414	1782	296276
l_∞	0.10	9768	2346	366300

The tables demonstrates significant inside cluster activity, and a significant decrease

Table 1.5 Number of root and coordinator mean computations, and total broadcasts for feature "ipo" with threshold $r = 0.0025$ and the "second moment clustering"

norm	best α	root mean update	coordinator mean update	total broadcasts
l_1	0.50	4585	121	127345
l_2	0.35	6304	421	180536
l_∞	0.30	8405	842	240455

in broadcasts due to the second moment.

Finally we turn to the feature "bosnia." Monitoring procedure presented in [23] and involving no clustering produced results collected in Table 1.6. Application of FMIC to monitoring this feature information gain reported in [6] was described as the "less successful." FMIC leads to a slight decrease in the number of broadcasts in case of

Table 1.6 Number of mean computations, and broadcasts, for feature "bosnia" with threshold $r = 0.0025$, no clustering

norm	mean updates	broadcasts
l_1	3053	79378
l_2	4006	104156
l_∞	3801	98826

the l_2 and l_∞ norms (see Table 1.7). In case of the l_1 norm, the number of broadcasts

Table 1.7 Number of root and coordinator mean computations, and total broadcasts for feature "bosnia" with threshold $r = 0.0025$ and the "first moment clustering"

norm	best α	root mean update	coordinator mean update	total broadcasts
l_1	0.65	3290	2	89128
l_2	0.55	3502	7	97602
l_∞	0.60	3338	2	91306

increases. The second moment clustering further significantly reduces the number of broadcasts, see Table 1.8. In the next section we briefly recall a number of well

Table 1.8 Number of root and coordinator mean computations, and total broadcasts for feature "bosnia" with threshold $r = 0.0025$ and the "second moment clustering"

norm	best α	root mean update	coordinator mean update	total broadcasts
l_1	0.65	1749	8	47717
l_2	0.75	1940	4	52510
l_∞	0.65	1756	8	47958

known clustering algorithms, and indicate how those can be used for monitoring data streams.

1.6 Conventional Clustering Algorithms

This Section briefly reviews a number of classical clustering algorithms we propose to use for node clustering. The algorithms are:

1. Principal Direction Divisive Partitioning (PDDP), [7];
2. batch $k-$means (see e.g. [27], [16]),
3. incremental $k-$means [1],
4. the combination of batch and incremental $k-$means [15].

1.6.1 PDDP

A good partitioning of a vector set into a number of subsets is a difficult problem even in the case when the required number of subsets is only two. There is, how-

ever, an exception. When the dimension of the vector space is one, i.e. one has to deal with a scalar set, the problem is relatively easy. While real–life data is rarely one dimensional, a least squares one dimensional approximation can be constructed and used to cluster a multidimensional vector set. The Principal Direction Divisive Partitioning algorithm briefly recalled below does just that. In the reminder of this section we demote by $\|\mathbf{a}\|$ the l_2 norm of a vector $\mathbf{a}$.

For a vector $\mathbf{a}$ and a line $\mathbf{l}$ in $\mathbf{R}^d$ denote by $\mathrm{P}_{\mathbf{l}}(\mathbf{a})$ the orthogonal projection of $\mathbf{a}$ on $\mathbf{l}$. For a set of vectors $\mathscr{A} = \{\mathbf{a}_1, \ldots, \mathbf{a}_m\}$ and a line $\mathbf{l}$ in $\mathbf{R}^d$ denote by $\mathrm{P}_{\mathbf{l}}(\mathscr{A})$ the set of projections $\{\mathrm{P}_{\mathbf{l}}(\mathbf{a}_1), \ldots, \mathrm{P}_{\mathbf{l}}(\mathbf{a}_m)\}$. For a fixed vector set $\mathscr{A}$ the quantity

$$\sum_{i=1}^{m} ||\mathbf{a}_i - \mathrm{P}_{\mathbf{l}}(\mathbf{a}_i)||^2$$

depends on the line $\mathbf{l}$. A line that minimizes this quantity (and provides the best least squares fit for the set $\mathscr{A}$) defines a principal direction. This line passes through the arithmetic mean $\boldsymbol{\mu} = \boldsymbol{\mu}(\mathscr{A})$ of the vector set $\mathscr{A}$, and its direction vector is an eigenvector of the matrix BB^T that corresponds to the maximal eigenvalue. Here $B = [\mathbf{a}_1, \ldots, \mathbf{a}_m] - \mu\mathbf{e}^T$, and $\mathbf{e}$ is a vector of ones (for details see e.g. [3]).

A basic step of the Principal Direction Divisive Partitioning algorithm (PDDP) is the following:

1. Given a set of vectors $\mathscr{A}$ in $\mathbf{R}^d$ determine the one dimensional line $\mathbf{l}$ that provides the "best" approximation to $\mathscr{A}$.
2. Project $\mathscr{A}$ onto $\mathbf{l}$, and denote the projection of the set $\mathscr{A}$ by $\mathscr{P}$ (note that $\mathscr{P}$ is just a set of scalars). Denote the projection of a vector $\mathbf{a}$ by p.
3. Partition $\mathscr{P}$ into two subsets $\mathscr{P}_1$ and $\mathscr{P}_2$.
4. Generate the induced partition $\{\mathscr{A}_1, \mathscr{A}_2\}$ of $\mathscr{A}$ as follows:

$$\mathscr{A}_1 = \{\mathbf{a} \,:\, p \in \mathscr{P}_1\}, \text{ and } \mathscr{A}_2 = \{\mathbf{a} \,:\, p \in \mathscr{P}_2\} \tag{1.12}$$

The algorithm divides the entire collection into two clusters by using the principal direction. Each of these two clusters will be divided into two sub-clusters using the same process recursively. The subdivision of a cluster is stopped when the cluster satisfies a certain "quality" criterion (such as, for example, cluster size, number of clusters, or cluster quality).

Implementation of the algorithm requires computation of the largest eigenvalue of the symmetric matrix BB^T. In many cases this task may not be performed analytically. While in the text mining application described in Section 1.2 the space dimension $d = 4$ one of the coordinates is an affine function of three others (see (1.4)). We now consider the case when each d dimensional data vector $\mathbf{a}$ can be written as

$$\mathbf{a} = \begin{bmatrix} \mathbf{b} \\ C\mathbf{b} + \mathbf{d} \end{bmatrix},$$

where $\mathbf{b} \in \mathbf{R}^{d_1}$, $\mathbf{d} \in \mathbf{R}^{d_2}$, $d_1 + d_2 = d$, and C is an $d_2 \times d_1$ matrix. For a vector set $\mathscr{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_m\}$ one has $\boldsymbol{\mu}(\mathscr{A}) = \begin{bmatrix} \boldsymbol{\mu}(\mathscr{B}) \\ C\boldsymbol{\mu}(\mathscr{B}) + \mathbf{d} \end{bmatrix}$. We now turn to the matrix

BB^T. Denoting $\mathbf{b}_i - \boldsymbol{\mu}(\mathscr{B})$ by $\mathbf{v}_i$ we obtain

$$B = \begin{bmatrix} \mathbf{v}_1 \dots \mathbf{v}_m \\ C\mathbf{v}_1 \dots C\mathbf{v}_m \end{bmatrix} = \begin{bmatrix} I \\ C \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 \dots \mathbf{v}_m \end{bmatrix}, \text{ where } I \text{ is the } d_1 \times d_1 \text{ identity matrix.}$$

Since rank $B \leq d_1$ one has rank $BB^T \leq d_1$, and the number of nonzero eigenvectors of BB^T does not exceed d_1. For our text mining application $d_1 = 3$, and the nonzero eigenvalues can be obtained by solving a cubic equation, i.e., the eigenvector for BB^T corresponding to the largest eigenvalue can be obtained just by solving a system of linear equations.

PDDP by itself generates good clustering results. Those could be further improved by applying $k-$means clustering to partitions generated by PDDP (see e.g. [3]). Next we briefly recall a number of versions of $k-$means.

1.6.2 Batch $k-$means

A work horse of clustering mentioned already in [27] and most often attributed to [16] batch $k-$means is by far most popular clustering algorithm. The algorithm is scalable, and easy to implement. The algorithm is centered around the concept of "centroid"–the best vector representative for a vector set introduced below (see [12], [13]).

For a set of vectors $\mathscr{A} = \{\mathbf{a}_1, \dots, \mathbf{a}_m\} \subset \mathbf{R}^n$, and a "distance" function $d(\mathbf{x}, \mathbf{a})$ define a centroid $\mathbf{c} = \mathbf{c}(\mathscr{A})$ of the set $\mathscr{A}$ as a solution of the minimization problem

$$\mathbf{c} = \arg\min \left\{ \sum_{\mathbf{a} \in \mathscr{A}} d(\mathbf{x}, \mathbf{a}),\ \mathbf{x} \in \mathscr{C} \right\}. \tag{1.13}$$

We call d a "distance" function because even in the classical implementation of $k-$means $d(\mathbf{x}, \mathbf{a}) = \|\mathbf{x} - \mathbf{a}\|_2^2$, the square of the l_2 norm, that fails to be a distance function (the triangle inequality does not hold). Further, $k-$means works with a wide class of functions called Bregman divergences and failing to be distances (Kullback–Leibler divergence is one of them, see [5]).

The quality of the set $\mathscr{A}$ is denoted by $q(\mathscr{A})$ and is defined by

$$q(\mathscr{A}) = \sum_{\mathbf{a} \in \mathscr{A}} d(\mathbf{c}, \mathbf{a}), \text{ where } \mathbf{c} = \mathbf{c}(\mathscr{A}) \tag{1.14}$$

(we set $q(\emptyset) = 0$ for convenience). Let $\Pi = \{\pi_1, \dots, \pi_k\}$ be a partition of $\mathscr{A}$, i.e.

$$\bigcup_i \pi_i = \mathscr{A}, \text{ and } \pi_i \cap \pi_j = \emptyset \text{ if } i \neq j.$$

We define the quality of the partition Π by

$$Q(\Pi) = q(\pi_1) + \ldots + q(\pi_k). \tag{1.15}$$

We aim to find a partition $\Pi^{\min} = \{\pi_1^{\min}, \ldots, \pi_k^{\min}\}$ that *minimizes* the value of the objective function Q. The problem is known to be NP–hard, and we are looking for algorithms that generate "reasonable" solutions. It is easy to see that centroids and partitions are associated as follows:

1. Given a partition $\Pi = \{\pi_1, \ldots, \pi_k\}$ of the set $\mathscr{A}$ one can define the corresponding centroids $\{\mathbf{c}(\pi_1), \ldots, \mathbf{c}(\pi_k)\}$ by:

$$\mathbf{c}(\pi_i) = \arg\min \left\{ \sum_{\mathbf{a} \in \pi_i} d(\mathbf{x}, \mathbf{a}),\ \mathbf{x} \in \mathscr{C} \right\}. \tag{1.16}$$

2. For a set of k "centroids" $\{\mathbf{c}_1, \ldots, \mathbf{c}_k\}$ one can define a partition $\Pi = \{\pi_1, \ldots, \pi_k\}$ of the set $\mathscr{A}$ by:

$$\pi_i = \{\mathbf{a} \,:\, \mathbf{a} \in \mathscr{A},\ d(\mathbf{c}_i, \mathbf{a}) \le d(\mathbf{c}_l, \mathbf{a}) \text{ for each } l = 1, \ldots, k\} \tag{1.17}$$

   (we break ties arbitrarily). Note that, in general, $\mathbf{c}(\pi_i) \neq \mathbf{c}_i$.

The classical batch k–means algorithm is a procedure that iterates between the two steps described above to generate a partition Π' from a partition Π.

While $0 \le Q(\Pi') \le Q(\Pi)$ and the process described above converges, it rarely converges to the global minimum. Even in a simple scalar case $\mathscr{A} = \{0, 2, 3\}$, and the initial partition $\Pi^{(0)} = \left\{\pi_1^{(0)}, \pi_2^{(0)}\right\}$ where $\pi_1^{(0)} = \{0, 2\}$, and $\pi_2^{(0)} = \{3\}$ an application of batch $k-$means to $\Pi^{(0)}$ does not change the partition, and misses a better partition $\Pi^{(1)} = \left\{\pi_1^{(1)}, \pi_2^{(1)}\right\}$ with $\pi_1^{(1)} = \{0\}$, and $\pi_2^{(1)} = \{2, 3\}$. The reason for this phenomenon along with a possible remedy is suggested in [10]. Before the relevant material is briefly recalled in the next section we remark that an application of PDDP to the scalar dataset $\mathscr{A}$ generates partition $\Pi^{(1)}$. This observation suggests to use PDDP to generate initial partitions to $k-$means like algorithms.

1.6.3 Incremental $k-$means

The failure of batch $k-$means to discover a better partition $\Pi^{(1)}$ stems from a simple fact that Step 2 of the procedure ignores change of centroids due to data-vectors' movement governed by (1.17). A way to accurately account for the centroid change is to allow a single data-vector movement during one iteration of the algorithm. This version of $k-$means is described, for example, in the classical manuscript [1].

While more accurate, incremental $k-$means changes cluster affiliation of only one vector per iteration. As compared to batch $k-$means the algorithm requires many more iterations to converge, hence is time consuming. We next discuss a "merger" of two algorithms.

1.6.4 Batch $k-$means followed by incremental $k-$means

While more accurate incremental $k-$means is not as fast as the batch algorithm. To benefit from speed of the batch algorithm and accuracy of the incremental $k-$means a number of contributions suggested to "merge" both algorithms as follows:

1. run batch $k-$means until it stops.
2. run one iteration of incremental $k-$means
3. if the iteration incremental $k-$means changed the partition
   goto Step 1
   else
   Stop.

All numerical computations associated with Step 2 of the algorithm have been already performed in Step 1. The improvement over batch $k-$means comes, therefore, at virtually no additional computational expense [11]. This "merger" was first introduced, perhaps, in [15], and later rediscovered by others[2]. Sequential application

$$\text{PDDP} \longrightarrow \text{batch } k-\text{means} \longrightarrow \text{incremental } k-\text{means}$$

generates good tight clusters. Next we describe how these tight clusters can be used for node clustering.

1.6.5 Node Clustering with Classical Clustering Algorithms

To simplify the exposition we first consider a two cluster $\{\pi_1, \pi_2\}$ partition problem for a given set of n vectors $\mathscr{A} = \{\mathbf{a}_1, \ldots, \mathbf{a}_n\} \subset \mathbf{R}^d$. We are seeking a partition $\Pi = \{\pi_1, \pi_2\}$ so that

$$\mathscr{A} = \pi_1 \bigcup \pi_2,\ \pi_1 \bigcap \pi_2 = \emptyset$$

and the partition Π quality $Q(\Pi)$ given by

$$Q(\Pi) = \max \left\{ \left|\left| \frac{1}{|\pi_1|} \sum_{\mathbf{a} \in \pi_1} \mathbf{a} \right|\right|, \left|\left| \frac{1}{|\pi_2|} \sum_{\mathbf{a} \in \pi_2} \mathbf{a} \right|\right| \right\}$$

is minimized. Due to convexity of any norm one has

$$\begin{aligned} \left|\left| \frac{1}{|\mathscr{A}|} \sum_{\mathbf{a} \in \mathscr{A}} \mathbf{a} \right|\right| &= \left|\left| \frac{|\pi_1|}{|\mathscr{A}|} \frac{1}{|\pi_1|} \sum_{\mathbf{a} \in \pi_1} \mathbf{a} + \frac{|\pi_2|}{|\mathscr{A}|} \frac{1}{|\pi_2|} \sum_{\mathbf{a} \in \pi_2} \mathbf{a} \right|\right| \\ &\leq \frac{|\pi_1|}{|\mathscr{A}|} \left|\left| \frac{1}{|\pi_1|} \sum_{\mathbf{a} \in \pi_1} \mathbf{a} \right|\right| + \frac{|\pi_2|}{|\mathscr{A}|} \left|\left| \frac{1}{|\pi_2|} \sum_{\mathbf{a} \in \pi_2} \mathbf{a} \right|\right| \\ &\leq \frac{|\pi_1|}{|\mathscr{A}|} Q(\Pi) + \frac{|\pi_2|}{|\mathscr{A}|} Q(\Pi) = Q(\Pi). \end{aligned}$$

[2] confirming the old adage that "success has many parents while failure is an orphan."

This inequality shows that the norm of the mean is a lower bound for $Q(\Pi)$. We next show how to build an optimal partition for a special particular case of the data set.

Assume that $n = 2m$, and the vector set $\mathscr{A}$ consists of two identical copies of m vectors, i.e.

$$\mathscr{A} = \{\mathbf{a}_1, \ldots, \mathbf{a}_m, \mathbf{a}_1, \ldots, \mathbf{a}_m\}.$$

If $\pi_1^o = \pi_2^o = \{\mathbf{a}_1, \ldots, \mathbf{a}_m\}$, then $|\pi_1^o| = |\pi_2^o| = \frac{1}{2}|\mathscr{A}|$, and

$$\max\left\{\frac{1}{|\pi_1^o|}\left\|\sum_{\mathbf{a}\in\pi_1^o}\mathbf{a}\right\|, \frac{1}{|\pi_2^o|}\left\|\sum_{\mathbf{a}\in\pi_2^o}\mathbf{a}\right\|\right\} = \left\|\frac{1}{|\mathscr{A}|}\sum_{\mathbf{a}\in\mathscr{A}}\mathbf{a}\right\|,$$

i.e., $\{\pi_1^o, \pi_2^o\}$ is an optimal partition.

This observation motivates the following two cluster $\Pi = \{\pi_1, \pi_2\}$ partition strategy:

1. Apply any clustering algorithm to the dataset $\mathscr{A}$ to generate clusters of size 2.
2. Select one vector from each cluster generated and assign selected vectors to cluster π_1.
3. Assign remaining vectors to cluster π_2.

Generalization of this strategy to a k cluster partition and full description of the algorithm is beyond the scope of this chapter and will be provided elsewhere.

1.7 Conclusions

In this chapter we consider application of clustering to monitoring data streams in a distributed system. Unlike standard clustering algorithms that aiming at collections of similar data items into same clusters, monitoring requires clusters with *dissimilar* vectors canceling each other as much as possible. A straightforward application of a standard clustering algorithm is, therefore, not possible.

We devise a specific clustering strategy that yields a reduction in communication load. The proposed clustering depends on a scalar parameter α, and may be too slow for applications involving systems with large number of nodes. Dependence of the number of broadcasts on α is not understood at this point and should be further investigated. Figure 1.2 shows the number of broadcasts for 18 values $\alpha = 0.05, 0.10, \ldots, 0.95$. The smallest number of broadcasts corresponds to $\alpha = 0.30$ (as reported in Table 1.5). Next we run monitoring for 98 values of $\alpha = 0.01, 0.02, \ldots, 0.99$ (see Figure 1.3). This time the smallest number of broadcasts corresponds to $\alpha = 0.16$. Zooming in does not indicate any particularly useful property of the function.

Each recomputation of δ (the distance from the mean $\frac{1}{n}\sum_{\mathbf{n}\in\mathbf{N}}\mathbf{v}_n(t_i)$ and the zero set $\mathbf{Z}_f$ of the function f) triggers recomputations of node constraints $\delta(\mathbf{n})$. This chapter uses the first and second moments to recompute node constraints. We plan to consider additional statistical metrics such as, for example, median for node constraints computations.

A possible applications of classical clustering algorithms is an additional research direction that may lead to scalable clustering procedures. While the experimental results demonstrate than communication savings may depend on the choice of a norm or a "distance" function many of the proposed clustering algorithms can be applied with Bregman divergences [26].

Appendix 1: First and Second Moments

In what follows we consider the auxiliary problem: "Let $\mathbf{X}$ be a vector set of size m with mean $\boldsymbol{\mu}$ and variance $\gamma > 0$. How far away from $\boldsymbol{\mu}$ a vector $\mathbf{x} \in \mathbf{X}$ can get?"

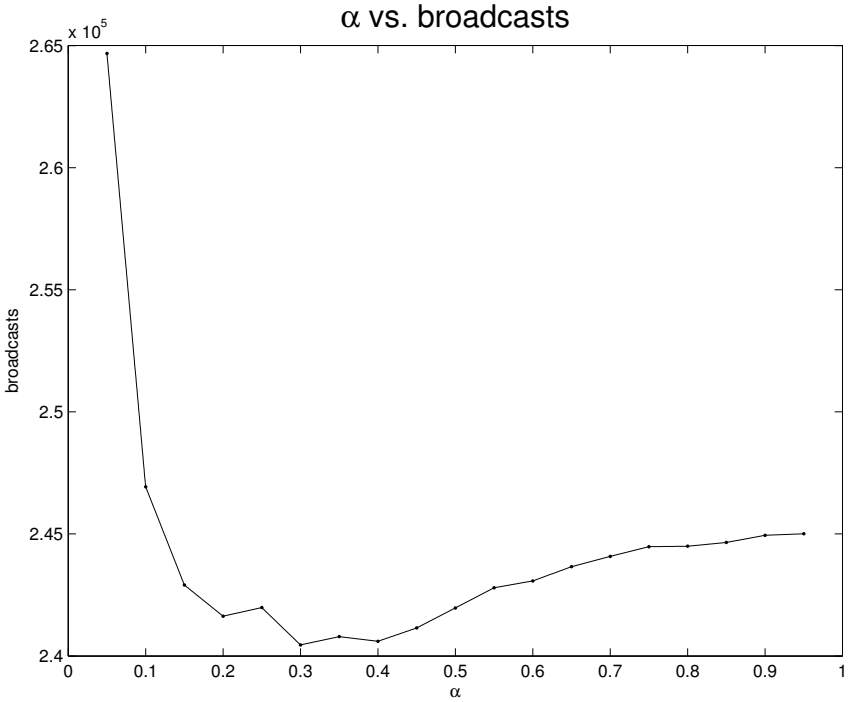

Fig. 1.2 l_∞ norm, $\alpha = 0.05, 0.10, \ldots, 0.95$ vs. broadcasts, for feature "ipo"

The answer to this question is provided below. To simplify the exposition we first assume $\boldsymbol{\mu} = 0$.

Let $\mathscr{X}(m,\gamma)$ be a family of sets $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \subset \mathbf{R}^d$ with $\boldsymbol{\mu}(\mathbf{X}) = 0$, and $\sigma^2(\mathbf{X}) = \sum_{i=1}^{m} (\mathbf{x}_i - \boldsymbol{\mu}(\mathbf{X}))^T (\mathbf{x}_i - \boldsymbol{\mu}(\mathbf{X})) = \gamma \geq 0$. In this section $||\mathbf{x}||$ stands for $||\mathbf{x}||_2$. For each $\mathbf{X} \in \mathscr{X}(m,\gamma)$ define $r(\mathbf{X})$ and $R(\gamma)$ as follows:

$$r(\mathbf{X}) = \max_{\mathbf{x}\in\mathbf{X}} \|\mathbf{x}\|, \; R(\gamma) = \sup_{\mathbf{X}\in\mathscr{X}(m,\gamma)} r(\mathbf{X}).$$

In what follows we describe sets $\overline{\mathbf{X}}_\gamma \in \mathscr{X}(m,\gamma)$ that maximize r, and the function $R(\gamma)$.

Lemma 1.1. *The function $R(\gamma)$ is a homogeneous function of degree $\frac{1}{2}$. For each positive scalar c one has $R(c\gamma) = c^{\frac{1}{2}} R(\gamma)$.*

Proof. Note that for a positive scalars t and s one has

$$tR(\gamma) = tr\left(\overline{\mathbf{X}}_\gamma\right) = r\left(t\overline{\mathbf{X}}_\gamma\right) \leq r\left(\overline{\mathbf{X}}_{\gamma t^2}\right) = R(\gamma t^2),$$

and

$$sR(\gamma t^2) = sr(\overline{\mathbf{X}}_{\gamma t^2}) = r(s\overline{\mathbf{X}}_{\gamma t^2}) \leq r(\overline{\mathbf{X}}_{\gamma t^2 s^2}) = R(\gamma t^2 s^2).$$

In particular when $ts = 1$ one has

$$R(\gamma) \leq t^{-1} R(\gamma t^2), \text{ and } R(\gamma t^2) \leq s^{-1} R(\gamma).$$

This shows that for positive t one has $tR(\gamma) = R(\gamma t^2)$ and completes the proof. □

Lemma 1.2. *Let $\mathbf{u} \in \overline{\mathbf{X}}_\gamma$ be such that $\|\mathbf{u}\| = r(\overline{\mathbf{X}}_\gamma) = R(\gamma)$. For each $\mathbf{x} \in \overline{\mathbf{X}}_\gamma$ there is a scalar c such that $\mathbf{x} = c\mathbf{u}$.*

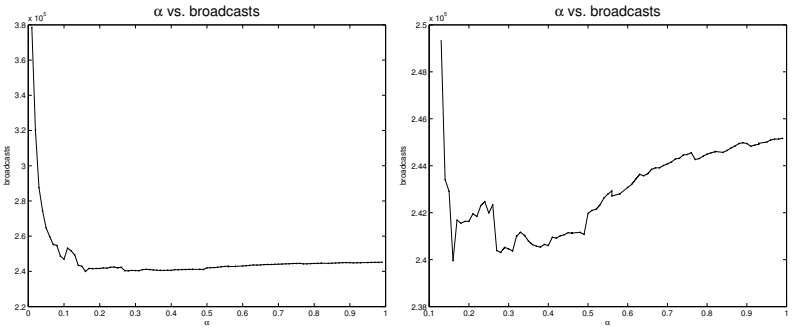

Fig. 1.3 feature "ipo", l_∞ norm, $\alpha = 0.01, 0.02, \ldots, 0.99$ vs. broadcasts, (left) and zoom in for $\alpha = 0.14, 0.15, \ldots, 0.99$ (right)

Proof. We assume now that the claim is false. Without any loss of generality we assume that $\|\mathbf{u}\| = 1$. Let $\{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ be all nonzero vectors in $\overline{\mathbf{X}}_\gamma$ so that $\mathbf{u}(\mathbf{u}^T\mathbf{x}_i) \neq \mathbf{x}_i$, $i = 1, \ldots, k$. The vectors $\mathbf{x}_i - \mathbf{u}(\mathbf{u}^T\mathbf{x}_i) \neq 0$, $\sum_{i=1}^{k} \left[\mathbf{x}_i - \mathbf{u}(\mathbf{u}^T\mathbf{x}_i)\right] = 0$, and $\sum_{i=1}^{m} \mathbf{u}(\mathbf{u}^T\mathbf{x}_i) = 0$. Consider now the vector set $\mathbf{X}' = \{\mathbf{x}'_1, \ldots, \mathbf{x}'_m\}$ where

$$\mathbf{x}'_i = \frac{1}{2}[\mathbf{x}_i - \mathbf{u}(\mathbf{u}^T\mathbf{x}_i)] + \mathbf{u}(\mathbf{u}^T\mathbf{x}_i),\ i = 1, \ldots, k, \text{ and } \mathbf{x}'_i = \mathbf{x}_i,\ i = k+1, \ldots, m.$$

We note that $\boldsymbol{\mu}(\mathbf{X}') = 0$, and $\sigma^2(\mathbf{X}') = \gamma' < \gamma$. Due to Lemma 1.1 one has

$$1 = r(\mathbf{X}') \leq R(\gamma') < R(\gamma) = 1.$$

This contradiction completes the proof.

Lemma 1.3. *Let* $\mathbf{u} \in \overline{\mathbf{X}}_\gamma$ *be such that* $\|\mathbf{u}\| = r(\overline{\mathbf{X}}_\gamma) = R(\gamma)$. *For each* $\mathbf{x} \in \overline{\mathbf{X}}_\gamma$, $\mathbf{x} \neq \mathbf{u}$ *there is a scalar* $c \leq 0$ *such that* $\mathbf{x} = c\mathbf{u}$.

Proof. First note that there is at least one $\mathbf{x} \in \overline{\mathbf{X}}_\gamma$ such that $\mathbf{x} = c\mathbf{u}$ with $c < 0$. We denote this c by c_-. Assume that the statement of the lemma is false. Then there is $0 < c_+ \leq 1$ such that $c_+\mathbf{u} \in \overline{\mathbf{X}}_\gamma$. Let $\varepsilon > 0$ be so small that $c_+ - \varepsilon > 0$, and $c_- + \varepsilon > 0$. Define $\mathbf{X}'$ by substituting the vectors $c_+\mathbf{u}$ and $c_-\mathbf{u}$ by $(c_+ - \varepsilon)\mathbf{u}$ and $(c_- + \varepsilon)\mathbf{u}$ correspondingly, and keeping the other $m-2$ vectors unchanged. We note that $\boldsymbol{\mu}(\mathbf{X}') = 0$, and $\sigma^2(\mathbf{X}') = \gamma' < \gamma$. Due to Lemma 1.1 one has

$$\|\mathbf{u}\| = r(\mathbf{X}') \leq R(\gamma') < R(\gamma) = \|\mathbf{u}\|.$$

This contradiction completes the proof.

Lemma 1.4. *Let* $\mathbf{u} \in \overline{\mathbf{X}}_\gamma$ *be such that* $\|\mathbf{u}\| = r(\overline{\mathbf{X}}_\gamma) = R(\gamma)$. *If* $\mathbf{x} \in \overline{\mathbf{X}}_\gamma$ *and* $\mathbf{x} \neq \mathbf{u}$, *then* $\mathbf{x} = -\dfrac{1}{m-1}\mathbf{u}$.

Proof. Assume the opposite, i.e., there are $\mathbf{x}_1 = c_1\mathbf{u}$, and $\mathbf{x}_2 = c_2\mathbf{u}$ such that $c_1 < c_2 \leq 0$. Let $\mathbf{X}'$ be a vector set obtained from $\overline{\mathbf{X}}_\gamma$ by substituting $c_1\mathbf{u}$ by $(c_1 - \varepsilon)\mathbf{u}$, $c_2\mathbf{u}$ by $(c_2 + \varepsilon)\mathbf{u}$, and keeping the other vectors unchanged. Note that $\boldsymbol{\mu}(\mathbf{X}') = 0$, and

$$\sigma^2(\overline{\mathbf{X}}_\gamma) - \sigma^2(\mathbf{X}') = 2\varepsilon(c_2 - c_1 - \varepsilon).$$

We note that for a small positive ε one has $\sigma^2(\overline{\mathbf{X}}_\gamma) > \sigma^2(\mathbf{X}')$. Due to Lemma 1.1 one has

$$\|\mathbf{u}\| = r(\mathbf{X}') \leq R(\gamma') < R(\gamma) = \|\mathbf{u}\|.$$

This contradiction completes the proof.

The next statement summarizes the above results.

Theorem 1.1. *If* $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \in \mathbf{R}^d$ *with* $\boldsymbol{\mu}(\mathbf{X}) = \boldsymbol{\mu}$, *and* $\sigma^2(\mathbf{X}) = \gamma \geq 0$, *then*

$$\|\mathbf{x}_i - \boldsymbol{\mu}\|^2 \leq \frac{m-1}{m}\gamma.$$

Further, $\|\mathbf{x}_m - \boldsymbol{\mu}\|^2 = \dfrac{m-1}{m}\gamma$ *if and only if*

$$\mathbf{x}_1 = \ldots = \mathbf{x}_{m-1} = \boldsymbol{\mu} - \frac{1}{m-1}[\mathbf{x}_m - \boldsymbol{\mu}].$$

We now establish connection between first and second moments for two sets of d dimensional vectors $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ and $\mathbf{X}' = \{\mathbf{x}'_1, \ldots, \mathbf{x}'_m, \mathbf{x}'_{m+1}\}$ where $2\mathbf{x}'_i = \mathbf{x}_i$, $i = 1, \ldots, m$.

$$\mu(\mathbf{X}') = \frac{m}{m+1}\left(\frac{1}{2}\mu(\mathbf{X})\right) + \frac{1}{m+1}\mathbf{x}'_{m+1}$$

$$\sigma^2(\mathbf{X}') = \frac{1}{4}\sigma^2(\mathbf{X}) + \frac{m}{m+1}\left(\mathbf{x}'_{m+1} - \frac{1}{2}\mu(\mathbf{X})\right)^T\left(\mathbf{x}'_{m+1} - \frac{1}{2}\mu(\mathbf{X})\right).$$

Appendix 2: Broadcast Count

Transmission of a double precision real number is defined as a message in [23]. In this chapter, in addition to real numbers typically representing vector coordinates, integer values such as node ID and node "reporting order" should also be transmitted. Transmission of node IDs is needed, for example, to allow the root to cluster nodes. To minimize communication load nodes in smaller clusters report violations of node constraints first, and the reporting order is assigned and communicated to nodes by the root that knows all cluster sizes.

Since every vector $\mathbf{v}$ associated with a node belongs to a simplex, it is represented by a real number not exceeding 1. We may use the integer part of these real numbers for transmission of integers. There is a variety of coding and compression techniques that can be used to transmit a set of real numbers as a single real. The discussion of these methods is beyond the scope of this chapter. In order to be able to compare different monitoring techniques we shall count a number of broadcasts, where by a broadcast we mean a single communication between two nodes. As an illustration, below we compute the number of broadcasts needed for one iteration of Algorithm 1.3.1 triggered by violation of a node constraint. We first assume that the violator node $\mathbf{n}$ is different from the root.

1. The violator node $\mathbf{n}$ notifies all other nodes (except the root) about the violation ($n-2$ broadcasts).
2. Each node $\mathbf{n}$ broadcasts its vector $\mathbf{v_n}$ to the root ($n-1$ broadcasts).
3. The root recomputes $\delta(\mathbf{r})$ and sends it to each node ($n-1$ broadcasts).

This leads to $3(n-1)-1$ broadcasts. If the violator node $\mathbf{n}$ is the root itself, the number of broadcasts becomes $3(n-1)$ (at step 1 above the root has to make $n-1$ broadcasts).

Next we turn to monitoring with clustering. The monitoring procedure starts with each node $\mathbf{n}$ sending its initial vector $\mathbf{v_n}(t_0)$ to the root $\mathbf{r}$ (that requires $n-1$ broadcasts). The root computes the mean $\frac{1}{n}\sum_{\mathbf{n}} \mathbf{v_n}(t_0)$ of the initial vectors, computes $\delta(\mathbf{r})$, and broadcasts $\delta(\mathbf{r})$ to each node ($n-1$ broadcasts). After exchanging

$$2(n-1) \tag{1.18}$$

broadcasts the monitoring proceeds with each node being a singleton cluster.

1. As long as the inequality

   $$|\mathbf{v_n}(t) - \mathbf{v_n}(t_0)| < \delta(\mathbf{r}) \text{ holds true for each node } \mathbf{n}$$

   the nodes are silent. At the first time instance t when the inequality is violated for at least one node $\mathbf{n}$, the following actions are triggered:

   a. the node $\mathbf{n}$ (if the node itself is not the root) broadcasts its ID and vector $\mathbf{v_n}(t)$ to the root (1 broadcast),
   b. the root issues $n-2$ requests for ID and $\mathbf{v_n}(t)$ to the other nodes ($n-2$ broadcasts),
   c. $n-2$ nodes report their IDs and $\mathbf{v_n}(t)$ vectors to the root ($n-2$) broadcasts),

   This brings the number of broadcasts to $2n-3$. If the violating node is the root, then this number is $2n-2$. To simplify the computations we select the largest number $2n-2$.
   At this step, and keeping in mind (1.18), the total number of broadcasts needed to be exchanged is

   $$2(n-1) + 2n - 2 = 4(n-1). \tag{1.19}$$

2. Next the root recomputes $\delta(\mathbf{r})$, clusters nodes, and broadcasts to each node ($n-1$ broadcasts) its updated local constraint $\delta(\mathbf{n})$, the ID of its coordinator, and the reporting order. If a node is also a coordinator, then IDs of its nodes, and coordinator reporting order are provided to the coordinator by the root. Keeping in mind (1.19), the total number of broadcasts right after the first root mean update and first clustering is

   $$5(n-1). \tag{1.20}$$

Clusters are now formed, and we shall count the number of broadcasts needed to be exchanged for each of the three types of possible violations:

1. A node constraint is violated in a singleton cluster.

   a. the violating node $\mathbf{n}$ reports its ID, $\mathbf{v_n}(t)$, $W_{\mathbf{n}}$, and the history vector $\mathbf{h_n}$ to the root (1 broadcasts),
   b. the root requests all other $n-2$ nodes to provide their input (ID's, $\mathbf{v_n}(t)$ vectors, $W_{\mathbf{n}}$ weights, and history vectors $\mathbf{h}$, total of $n-2$ broadcasts),
   c. the $n-2$ nodes report ID's, $\mathbf{v_n}(t)$ vectors, $W_{\mathbf{n}}$ weights, and history vectors $\mathbf{h}$ to the root ($(n-2)$ broadcasts),

d. the root recomputes the constraint $\delta(\mathbf{r})$, node constraints $\delta(\mathbf{n})$, and reports to each node its coordinator ID, $\delta(\mathbf{n})$, and the node "reporting order." Cluster coordinators also receive IDs of the nodes in their respective clusters ($n-1$ broadcasts).

This leads to $3(n-1)-1$ broadcasts if the violating node is not the root, and $3(n-1)$ broadcasts if the violation is at the root. To compute the broadcasts we use the larger number

$$3(n-1). \tag{1.21}$$

2. A node constraint is violated in a non singleton cluster π with coordinator $\mathbf{c}$.

a. the violator $\mathbf{n}$ reports its ID, $\Delta_{\mathbf{n}}$, and $\delta_{\mathbf{n}}$ to the coordinator $\mathbf{c}$ (1 broadcast),
b. the coordinator $\mathbf{c}$ sends request for $\Delta_{\mathbf{n}}$ vectors and node constraints $\delta_{\mathbf{n}}$ for all nodes in its cluster π other then $\mathbf{n}$ and itself ($|\pi|-2$ broadcasts)
c. the nodes broadcast their vectors Δ and constraints δ to the coordinator (total of $(|\pi|-2)$ broadcasts). The total comes to $2|\pi|-3$, and this number is $2|\pi|-2$ when the violating node is the coordinator.

The total of broadcasts needed is:

$$2|\pi|-2. \tag{1.22}$$

3. A coordinator constraint is violated. First we assume the coordinator $\mathbf{c}$ is not the root:

a. the coordinator $\mathbf{c}$ of cluster π broadcasts requests to all nodes (except itself and the root) to provide the root with their IDs, vectors $\mathbf{v}_{\mathbf{n}}(t)$, weights W, and history vectors $\mathbf{h}$ ($n-2$ broadcasts).
b. $n-1$ nodes ($n-2$ nodes requested by the coordinator and the coordinator itself) send the requested information to the root ($n-1$ broadcasts).
c. the root recomputes $\delta(\mathbf{r})$, clusters nodes and provides each node with updated local constraint $\delta(\mathbf{n})$, the new cluster affiliation (i.e. ID of a new coordinator), and the node "reporting order." Coordinators are also provided with the IDs of their nodes (total of $n-1$ broadcasts).

This brings the number of broadcasts to $3(n-1)-1$. If $\mathbf{c}$ is the root, then this number is $3(n-1)$, and this is the number we use to compute broadcasts

$$3(n-1). \tag{1.23}$$

References

1. Duda, R. O. and Hart, P. E. and Stork, D. G.: Pattern Classification. John Wiley & Sons, (2000)
2. Gray, R.M.: Entropy and Information Theory. Springer–Verlag, New York, (1990)

3. Kogan, J.: Introduction to Clustering Large and High–Dimensional Data. Cambridge University Press, New York, (2007)
4. Mirkin, B.: Clustering for Data Mining: A Data Recovery Approach. Chapman & Hall/CRC, Boca Raton, (2005)
5. Banerjee, A. and Merugu, S. and Dhillon, I. S. and Ghosh, J.: Clustering with Bregman divergences. Proceedings of the 2004 SIAM International Conference on Data Mining. 234–245, (2004)
6. Barouti, M., Keren, D., Kogan, J., Malinovsky, Y.: Monitoring Distributed Data Streams Through Node Clustering. (2014)
7. Boley, D. L.: Principal Direction Divisive Partitioning. Data Mining and Knowledge Discovery. **2**, 325–344 (1998)
8. Brucker, P.: On the complexity of clustering problems. In Lecture Notes in Economics and Mathematical Systems, **157**, 45–54, Springer-Verlag, Berlin, (1978)
9. Burdakis, S. and Deligiannakis A.: Detecting outliers in sensor networks using the geometric approach. In ICDE, 1108-1119, (2012)
10. Dhillon, I. S. and Guan, Y. and Kogan, J.: Iterative Clustering of High Dimensional Text Data Augmented by Local Search. Proceedings of the 2002 IEEE International Conference on Data Mining, 131–138, (2002)
11. Dhillon, I. S. and Kogan, J. and C. Nicholas: Feature Selection and Document Clustering. Survey of Text Mining, M.W. Berry (ed.), Springer-Verlag, 73-100, (2003)
12. Diday, E.: The dynamic cluster method in non–hierarhical clustering. J. Comput. Inf. Sci., VOLUME = 2, 62-88, (1973)
13. Diday, E.: Some recent advances in clustering. Recent Developments in Clustering and Data Analysis. Proceedings of the Japanise–French Scientific Seminar, 119-136, (1987)
14. Dilman, M and Raz, D.: Efficient reactive monitoring. Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communication Societies, 1012-1019, (2001)
15. Hansen, P. and Mladenovic N.: J–Means: a new local search heuristic for minimum sum of squares clustering. Patern Recognition. **34**, 405-413, (2001)
16. Forgy, E.: Cluster Analysis of Multivariate Data: Efficiency vs. Interpretability of Classifications. Biometrics. **21**, 768, (1965)
17. Gabel, M. and Schuster, A. and Keren, D.: Communication-efficient outlier detection for scale-out systems. In BD3@VLDB, 19-24, (2013)
18. Garofalakis, M. and Keren, D. and Samoladas, V.: Sketch-based geometric monitoring of distributed stream queries. In PVLDB, 2013.
19. Madden, S. and Franklin, M.J.: An architecture for queries over streaming sensor data. ICDE 02, 555, (2002)
20. I. Sharfman and A. Schuster and D. Keren: A Geometric Approach to Monitoring Threshold Functions over Distributed Data Streams, ACM Transactions on Database Systems, **32**, 23:1-23:29, (2007)
21. I. Sharfman and A. Schuster and D. Keren: A Geometric Approach to Monitoring Threshold Functions over Distributed Data Streams. In Ubiquitous Knowledge Discovery, M. May and L. Saitta (eds.), 163-186, Springer–Verlag, (2010)
22. D. Keren and I. Sharfman and A. Schuster and A. Livne: Shape Sensitive Geometric Monitoring. IEEE Transactions on Knowledge and Data Engineering, **24**, 1520-1535, (2012)
23. Kogan, J.: Feature Selection over Distributed Data Streams through Convex Optimization. Proceedings of the Twelfth SIAM International Conference on Data Mining (SDM 2012), SIAM, 475-484, (2012)
24. Kogan, J. and Malinovsky, Y.: Monitoring Threshold Functions over Distributed Data Streams with Clustering. In Proceedings of the Workshop on Data Mining for Service and Maintenance (held in conjunction with the 2013 SIAM International Conference on Data Mining), 5-13, (2013)
25. Manjhi, A. and Shkapenyuk, V. and Dhamdhere, K. and Olston, C.: Finding (recently) frequent items in distributed data streams. ICDE 05, 767-778, (2005)

26. Teboulle, M. and Berkhin, P. and Dhillon, I. and Guan, Y. and Kogan, J.: Clustering with Entropy–like *k*–means Algorithms. In Grouping Multidimensional Data: Recent Advances in Clustering. J. Kogan and C. Nicholas and M. Teboulle (eds.), Springer–Verlag, 127–160, (2006)
27. Steinhaus, H.: Sur la division des corps matèriels en parties. Bulletin De L'Acadēmie Polonaise Des Sciences Classe III Mathematique, Astronomie, Physique, Chimie Geologie, et Geographie. **4**, 801-804, (1956)
28. Yi, B.-K. and Sidiropoulos N. and Johnson T. and Jagadish, H.V. and Faloutsos, C. and Biliris A.: Online datamining for co–evolving time sequences. ICDE 00, 13, (2000)
29. Zhu, Y. and Shasha, D.: Statestream: Statistical monitoring of thousands of data streams in real time. VLDB, 358-369, (2002)