MONITORING DISTRIBUTED DATA STREAMS

AVISHAY LIVNE
MONITORING DISTRIBUTED DATA STREAMS

RESEARCH THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SCIENCE

Avishay Livne

SUBMITTED TO THE SENATE OF THE TECHNION - ISRAEL INSTITUTE OF TECHNOLOGY

Sivan 5770 Haifa June 2010
ACKNOWLEDGMENT

I am grateful to Prof. Assaf Schuster, Prof. Daniel Keren and Guy Sagy for their support and guidance.

THE GENEROUS FINANCIAL HELP OF THE TECHNION IS GRATEFULLY ACKNOWLEDGED
Contents

1 Introduction 3

2 Related Work 7

3 Preliminaries 9
   3.1 Distributed Threshold Monitoring 9
   3.2 Minkowski Sum and Average 9

4 Safe Zone Monitoring 12

5 Algorithmic Framework 16
   5.1 Practical Computation of the Safe Zones 16
   5.2 Constructing the Optimization Problem 16
      5.2.1 Polygons as SZs 18
      5.2.2 Axis-aligned rectangles 18
      5.2.3 Rotated rectangle 19
      5.2.4 Union of rectangles 19
      5.2.5 Convex polygons 20
   5.3 Approximation Quality of Safe Zones 21
      5.3.1 Rectangles 21
      5.3.2 Convex polygonal SZs 22
6 Scalability

6.1 Number of Parameters in the Optimization Process 23
6.2 Iterative Clustering Algorithm 24
6.3 Algorithm Correctness 26
6.4 Number of Clusters 27
6.5 Number of Iterations vs. Optimality 27
6.6 Distance Measure and Cluster Representation 29
   6.6.1 Distance measure 29
   6.6.2 Cluster representation 30

7 Probabilistic Approach 31

8 Empirical Evaluation 33

8.1 Experimental Framework 33
8.2 Various Types of SZs 33
8.3 Complexity of the Safe Zone Type 37
8.4 Reduction in Local Computation 38
8.5 Clustering 38
   8.5.1 Flat clustering 39
   8.5.2 Number of iterations 39
   8.5.3 Scalability 41
   8.5.4 Loss of optimality 43
8.6 Probabilistic Approach 46
9 Future Work

9.1 Monitoring Higher Dimensions ............... 49
9.2 Recovery From Violations ................. 50
9.3 Monitoring Other Types of Functions ......... 51

10 Conclusions 52

References 53
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Illustration of the Minkowski sum operation</td>
</tr>
<tr>
<td>2</td>
<td>Illustration of the Minkowski average operation</td>
</tr>
<tr>
<td>3</td>
<td>Illustration of the Minkowski sum operation on two SZs</td>
</tr>
<tr>
<td>4</td>
<td>Illustration of the constraint on the Minkowski average of two SZs</td>
</tr>
<tr>
<td>5</td>
<td>Example of two axis-aligned SZs</td>
</tr>
<tr>
<td>6</td>
<td>Inscribing a rectangle within an ellipse</td>
</tr>
<tr>
<td>7</td>
<td>Example of SZ composed of union of two free rectangles</td>
</tr>
<tr>
<td>8</td>
<td>Different types of SZs and their number of parameters</td>
</tr>
<tr>
<td>9</td>
<td>Illustration of the iterative clustering algorithm</td>
</tr>
<tr>
<td>10</td>
<td>Similarity and dissimilarity of data distribution</td>
</tr>
<tr>
<td>11</td>
<td>Example of non-balanced clustering</td>
</tr>
<tr>
<td>12</td>
<td>The effect of number of iterations</td>
</tr>
<tr>
<td>13</td>
<td>Example of the probabilistic approach</td>
</tr>
<tr>
<td>14</td>
<td>Example of real-data distribution</td>
</tr>
<tr>
<td>15</td>
<td>Comparison of different SZ types</td>
</tr>
<tr>
<td>16</td>
<td>Performance of different SZ types</td>
</tr>
<tr>
<td>17</td>
<td>Cost of monitoring of various SZ types</td>
</tr>
<tr>
<td>18</td>
<td>Comparison of clustering algorithms</td>
</tr>
<tr>
<td>19</td>
<td>Performance and cost of hierarchical clustering</td>
</tr>
<tr>
<td>20</td>
<td>Scalability test - performance and cost with respect to number of nodes</td>
</tr>
<tr>
<td>21</td>
<td>Loss of optimality due to clustering</td>
</tr>
<tr>
<td>22</td>
<td>Distribution of nodes in the AirBase data</td>
</tr>
<tr>
<td>23</td>
<td>SZs of clustered nodes with different orientation</td>
</tr>
<tr>
<td>24</td>
<td>Clustering nodes with different orientation - performance and cost</td>
</tr>
<tr>
<td>25</td>
<td>Probabilistic Approach - the effect of error rate</td>
</tr>
<tr>
<td>26</td>
<td>Probabilistic approach - the effect of number of nodes</td>
</tr>
<tr>
<td>27</td>
<td>Resolving violations locally</td>
</tr>
</tbody>
</table>
Abstract

Monitoring distributed streams of data is a basic construct in many distributed systems. Examples include a wireless sensor network, where we would like to receive an alert every time the average of the temperature readings taken by the sensors exceeds a given threshold, or a distributed search engine, where we would like to determine the set of queries whose frequency of use exceeds a given threshold.

So far, research has focused on detecting global properties that are defined by simple aggregates (e.g., sum, average, or minimum) over the distributed data. For these cases, there are usually local conditions for the occurrence of the global property of interest. For example, if we wish our monitoring system to alert when the average temperature is over 50 degrees, we can use the observation that for this to occur, the temperature measurement in at least one node has to be over 50 degrees - hence configuring each node to remain silent as long as its reading is below 50 degrees.

In many cases, however, global properties of interest are expressed by more complex functions. Examples include detecting when the variance of the sensor readings in a sensor network exceeds a given threshold, determining the set of pairs of queries whose correlation exceeds a given threshold, and performing distributed classification tasks. Monitoring such complex functions is an expensive task in terms of communication, processing and storage access. Almost all the research to date has been restricted to linear scoring functions, to the exclusion of many other important functions, such as variance, mutual information and correlation coefficients.

We study global properties of the following form: we assume that each node holds a time varying two dimensional vector. In addition, we assume that we are
given an arbitrary scoring function that is calculated on the weighted average of the local vectors held by the nodes. We are interested in detecting when the value of this function violates a predetermined threshold value.

In the general case described above, it is far harder to devise local conditions such as the ones employed for simple aggregates. Even in the apparently trivial case of \( f(x) = x^2 \), we cannot make any inference on whether \( f\left(\frac{x+y}{2}\right) \) is above or below the threshold \( t \), when given the position of \( f(x) \) and \( f(y) \) vis-a-vis a threshold value \( t \); i.e., if we know that \( f(x) > t \) and \( f(y) > t \), we cannot tell whether \( f\left(\frac{x+y}{2}\right) > t \) or \( f\left(\frac{x+y}{2}\right) < t \). As this simple example illustrates, it is generally very difficult to find local indications for monitoring nonlinear functions. The problem is that it is not enough to know the function’s values at the local data vectors; we have to know something about the possible domain which contains the average of the local vectors.

Our proposed approach is based on the mathematical and geometric properties of the scoring functions, which have never been used by the data mining community. We plan to apply this approach for different data mining algorithms in different distributed setups, for which current solutions are either suboptimal or nonexistent.
1 Introduction

A basic construct in many distributed systems is the detection of global properties over distributed data. Consider, for example, a system designed to warn of forthcoming tsunamis. Such a system consists of water pressure sensors distributed over a wide area in the ocean. It can predict whether a tsunami is going to strike based on the combined readings of these sensors. The system must be able to issue correct warnings to save lives; however, false alarms are undesirable, as they may result in the needless evacuation of many thousands of people (as occurred in northern Japan in November 2006). If there were no power and computational resource limitations, the sensors could continuously broadcast their readings to a central monitoring station, and these could be processed there. Alas, that is not possible: wireless sensors have restricted power supply that can support a limited number of transmissions, and the central node usually cannot analyze all of the incoming data on the fly. In other distributed domains, additional restrictions such as huge data volumes come into play: for example, in a network-traffic monitoring scenario, a simple calculation shows that capturing only IP packet-header information over a single 1Gigabit/s Ethernet link can easily give rise to terabytes of data daily. Yet another example could be a search engine comprised of a set of distributed mirrors. Each mirror receives a stream of queries, each consisting of multiple search terms. Given a search term, we would like to generate an alert every time the global frequency of appearance of the term crosses a given threshold.

In these cases and others, it is not feasible to centralize and analyze all the dynamic information in real time. Nor will increased communication hardware abilities or CPU power solve this problem: physics places a lower bound on the amount of energy required to transmit information; the number of nodes in distributed information technology is growing constantly; increasingly sophisticated applications require sending and processing ever-increasing amounts of data; and centralizing the data is prohibited in many distributed applications due to privacy concerns.

However, knowing all the information is unnecessary for many applications; it suffices to know when, roughly speaking, something happens which requires attention. For instance, most sensor readings are not important because tsunamis are, fortunately, very rare. A distributed system to warn of a DDoS attack locally monitors numerous nodes, but usually these local measurements are unextraordinary. Since the current modus operandi of these systems is periodic collection of data, they not only fail to provide a timely alert, but also waste resources
on sending unimportant data. The question is therefore how to locally identify global system trends and transmit the local data only when such a global trend develops.

Most of the research on monitoring distributed data streams has focused mainly on monitoring simple aggregates, such as detecting when the sum of a distributed set of variables exceeds a predetermined threshold [18], or finding frequently occurring items in a set of distributed streams [5]. Monitoring more complex constructs over distributed streams, such as approximating quantiles [12], or detecting similar behavior among distributed streams [7], have been addressed in previous research, but the proposed solutions are tailored to the specific problem at hand. Monitoring in sensor networks has been addressed in the past as well [32, 33, 44, 9, 48]. A common approach to monitoring sensor networks is to periodically collect data from the sensors. The authors of [32] proposed a tree based algorithm for calculating aggregates over sensor networks. Algorithms for calculating quantiles over sensor networks include [44]. An alternative to the tree approach is the multi-path approach [9], which leverages the broadcast communication medium used by the sensors. In [33] an algorithm that combines the tree based and the multi-path approaches was proposed. An algorithm for continuously monitoring simple aggregated values over a sensor network, such as sums, counts and averages, has been presented in [48]. In contrast to the tree based and multi-path approaches, which require periodically sampling the network, communication generated by the algorithm presented in [48] is driven by changes in the data collected by the sensors.

Distributed monitoring algorithms usually make use of local conditions for the occurrence of the global property of interest. For example, in the distributed search engine example given above, if we would like to determine that the global frequency of appearance of a search term $A$ is below a given threshold value, it is sufficient to verify that at any given time the local frequency of appearance of $A$ is below the threshold at each mirror. As long as all these local conditions are satisfied, the mirrors can determine that the global frequency of appearance of the search term is below the threshold without communicating.

In many cases, however, global properties of interest are expressed by more complex functions. Consider the following, more complex global property: given a pair of search terms, $A$ and $B$, we would like to generate an alert every time the global correlation between the appearance of $A$ and that of $B$ crosses a predetermined threshold. This global correlation is determined as follows: let $f_A$ denote the global frequency of appearance of $A$ (i.e., the number of queries that contain the term $A$ divided by the number of queries received by the mirrors). Similarly, let $f_B$ denote the global frequency of appearance of $B$. In addition, let $f_{AB}$ denote
the global frequency of queries that contained both $A$ and $B$. The global correlation between the appearance of $A$ and $B$ is denoted by $\rho_{AB}$ and is determined by the following function, known as Pearson’s correlation coefficient:

$$\rho_{AB}(f_A, f_B, f_{AB}) = \frac{f_{AB} - f_A f_B}{\sqrt{(f_A - f_A^2)(f_B - f_B^2)}}.$$ 

The correlation coefficient receives values in the range $[-1..1]$. A negative score indicates that the terms tend to exclude each other, a score of zero indicates that there is no correlation between the appearance of the terms, and a positive score indicates that the terms tend to appear in the same queries.

Note that for the search term correlation task it is far harder to devise local conditions such as the ones employed for simple aggregates. In fact, the global correlation coefficient for a pair of search terms can be above a given threshold even if the local correlation coefficient determined for the pair at every node is below the threshold. Consider, for example, two nodes, $p_1$ and $p_2$, holding the following statistics for the terms $A$ and $B$: let $f_{A,1} = 0.1$, $f_{B,1} = 0.1$ and $f_{AB,1} = 0.019$, $f_{A,2} = 0.4$, $f_{B,2} = 0.4$ and $f_{AB,2} = 0.184$. In addition, assume that both nodes received an equal number of queries, and therefore the global statistics for the pair are as follows: $f_A = 0.25$, $f_B = 0.25$ and $f_{AB} = 0.1015$. It is easy to verify that the local scores for this pair are 0.1 at both nodes. However, the global correlation coefficient for the pair is 0.208. Given a threshold value of 0.15, the local correlation coefficient is below the threshold at both nodes, while the global correlation coefficient is above it.

Previous work on threshold monitoring of general (not necessarily linear or monotonic) functions [42, 43] developed geometric compilation of local constraints that can be checked locally at each node. The methods proposed in these papers, however, are still deficient in several respects: there is no optimality criterion in the definition of the local constraints, the constraints are the same for all nodes and are nonadaptable (therefore these algorithms may perform poorly if the distributions of the data in distinct nodes are different), and the computational complexity of checking the constraints is high. Nonlinear functions in a peer-to-peer setup are also dealt with in [46], where an approximated local algorithm for monitoring global properties is proposed. However, this work has a few drawbacks such as a notable increase in communication when the monitored value is near the threshold and the fact that accuracy can only be improved with increased communication and computational demand. In [41] a distributed threshold detection algorithm is proposed, but for one-dimensional data and a restricted set of functions (polynomials with positive coefficients).
In this work we develop a novel theory for the compilation of local constraints that can be used to implement distributed threshold monitoring systems for complex scoring functions. In turn, a local constraint determines a local safe zone (SZ): data contained in the local SZ requires no special attention and no communication. Our approach uses the notion of Minkowski sum [40] to formulate the problem as a general optimization task. We define a general measure of optimality for the communication minimization problem and apply it in the development and evaluation of the proposed algorithm. The approach allows compiling the SZs while fitting them to the data distribution at each node, thus obtaining much better overall performance. Furthermore, the computational task performed at the nodes for testing the local constraints (i.e., testing that the data belong to the local SZ), which is very difficult in previous work, becomes straightforward and fast with the new method. This is crucial for a feasible implementation of the algorithm in battery-powered environments comprised of thin nodes, such as wireless sensor networks and mobile robots. Empirical evaluation using real-world data shows that the efficiency and flexibility of the new method reduces the communication volume by orders of magnitude over previous general methods [42, 43]. Furthermore, the improvement over previous work increases with the number of nodes.

This thesis is structured as follows. Section 2 reviews related work. Section 3 provides a formal description of the problem as well as the theoretical foundations of our solution. Section 4 presents our core algorithm and section 5 presents the algorithmic framework. Two important extensions to the original algorithm, the use of clustering and the probabilistic approach, are described in sections 6 and 7 respectively. Section 8 provides experimental results. Finally, section 9 describes possible directions for further research and future work, and section 10 presents our conclusions.
2 Related Work

Our work pertains to the task of monitoring distributed streams from a geometric point of view, as detailed in the following sections. Some of the theoretical foundations of our work were initially suggested in [42, 43], where a geometric approach to monitoring threshold functions was proposed. This geometric approach defines for each node a subset of the data tuples such that as long as the local vectors are inside their respective subsets, the function's value is guaranteed not to cross the threshold. However, this algorithm suffers from a few drawbacks. First, the definition of these subsets is heuristic, with no optimality criterion. Second, the shape of the subsets in the different nodes is identical; they are all translations of one another. That means that if the data in different nodes obey different distributions, the method will perform poorly. For example, if the distribution at the first node is elongated at the $x$-direction and at the second in the $y$-direction, it makes sense that the subsets at the first (second) node will be elongated along the $x$ ($y$) direction, thus allowing us to "capture more probability". Third, the subsets are defined not directly but by an implicit condition which is not straightforward to compute. As a consequence, the computational cost of the nodes is high. We address all these concerns by introducing an optimality criterion, defining flexible and different shapes of subsets in different nodes in order to better fit distinctive distributions, and assigning local constraints at the nodes which require low computational cost.

One of the few papers to deal with nonlinear functions is [46]: given a distributed system, with a data vector at each node, it presents a majority-voting algorithm for determining whether the norm of the average vector is below a certain threshold, or more generally, whether it lies inside a given convex set. However, this algorithm suffers from several drawbacks. The first is the lack of termination. While the algorithm guarantees convergence, there is no guarantee at each point that the system state represents the underlying data. Second, the algorithm is approximate, and accuracy can only be improved by increasing the computational and communicational cost. Finally, the algorithm introduces a drastic increase in communicational cost when the value of the monitored function is close to the threshold. Our solution has none of these disadvantages.

The authors of [27] stress the importance of developing distributed monitoring algorithms, and suggest assigning appropriate local thresholds at each node to allow efficiency. They also touch on the importance and difficulty of the problem of monitoring nonlinear functions: "Standard database languages offer other aggregates including AVERAGE, STDEV |standard
deviation], MAX and MIN. Given a constraint on one of these global aggregates (e.g., ‘ensure that the STDEV of latency is ≤ l second.’), it is not immediately clear what local ‘event’ should trigger global constraint checks.”

A general method used by traditional monitoring algorithms in order to monitor nonlinear functions in a distributed setup is polling. A coordinator periodically polls system variables and checks to see whether a global condition was violated. However, this method suffers from high communication overhead. Intelligent polling algorithms were proposed [49], but they may miss constraint violations unless the polling interval is set to be very short. In [47], a gossiping-based algorithm is presented, but it does not cover general functions.

In [25] a distributed paradigm to decide on the dimension of an approximating subspace for distributed data is suggested, with the application of detecting system anomalies such as a DDoS attack. A theoretical paper which discusses functional approximation in a distributed setting is [14] but it only deals with obtaining lower bounds for vector norm functions.

Other works on reducing communication are sketching [4, 10, 11], detection of “heavy hitters” [34], computing quantiles [12], and counting distinct elements [15]. Other topics in distributed computation include top- \(k\) problems [36, 5], set-expression cardinality estimation [17], clustering [16], and distributed verification of logical expressions [2].

In our work we make use of a probabilistic approach, exploiting the fact that in many applications it is acceptable for the monitoring algorithm to err once in a while. By allowing just a fraction of unreported violations, we manage to considerably reduce the overall communication. Some work along these directions is [3, 11, 13].
3 Preliminaries

3.1 Distributed Threshold Monitoring

Let us now introduce the formal definition of the distributed monitoring problem. Most such problems can be reduced to the question of threshold crossing. The formal definition is straightforward: given are a function \( f \), a threshold \( T \), and a distributed system with observations \( v_1(t), v_2(t), ..., v_n(t) \), where \( v_i(t) \) is a data tuple (vector) observed at the \( i^{th} \) node at time \( t \). New data is observed as \( t \) progresses, and the system needs to send an alert message whenever the value of \( f(v_1(t), v_2(t), ..., v_n(t)) \) crosses \( T \) (that is, either changes from a value larger than \( T \) to a value smaller than \( T \), or vice versa). In the tsunami warning system example given in section 1, \( v_i(t) \) are the measurements of the individual sensors, and \( f \) is a function that determines the level of danger from an imminent tsunami, given the entire collection of local measurements.

In our solution we will rely on the following assumptions in order to introduce a criterion for optimal monitoring. First, we assume the observations at node \( i \) are distributed according to some known distribution \( p_i \), or that it is possible to sample enough observations that will sufficiently represent node \( i \)'s distribution. Second, we assume the distributions of the nodes are independent of each other, i.e., knowing the distribution of node \( i \) provides no information regarding the distribution of node \( j \). Our general algorithm doesn't require this assumption, but it is crucial for our optimality criterion. It is possible to suggest a different optimality criterion if this assumption does not hold. Last, we assume the distributions are constant in time. In case there is a drift in the distributions it is possible to restart our algorithm. We suggest ideas for more efficient methods for recovery from distributions’ drift in section 9.2.

3.2 Minkowski Sum and Average

The Minkowski sum and Minkowski average operations, intrinsic to the theoretical foundations of our work, are briefly described here.

The Minkowski sum is a well-known construct in mathematics and computer science, for example in computational geometry and robotics. Its definition is the same as that of the sum of two vector subspaces, but it is applied to all subsets of the ambient Euclidean space, not necessarily subspaces. Let us assume we are working in the plane \( \mathbb{R}^2 \), and let \( A, B \) be any subsets. Their Minkowski sum is defined as the set consisting of all vector sums with one
3.2 Minkowski Sum and Average

Figure 1. Simple Minkowski sums (a) and an illustration of the Minkowski sums obtained by sliding a diamond shape over a triangle shape (b)

summand from $A$ and one from $B$: $A \oplus B = \{a + b | a \in A, b \in B\}$ (extension to more than two summands is straightforward). Figure 1(a) presents simple examples of the Minkowski sum operation. This operator is just what we need for the case in which the system’s global vector is the sum, or average, of the local vectors held at the nodes. Figure 1(b) demonstrates what the Minkowski sum looks like when the two summands are a diamond shape ($A$) and a triangle ($B$). To understand the figure, note that $A \oplus B = \bigcup_{a \in A} \{a\} \oplus B$, and that $\{a\} \oplus B$ is the translation of $B$ by $a$. Therefore, the Minkowski sum of two shapes can be visualized as the union of copies of the first shape, with one copy for each point in the second set, where each such copy is obtained by translating the first shape by the point of the second. It is easy to see that this union is obtained by placing the first shape on the second, “sliding” it over and taking the union of all the copies of the first shape thus obtained. The Minkowski sum behaves nicely under unions; a known property of Minkowski sums which we exploit in our work is $A \oplus (B \cup C) = (A \oplus B) \cup (A \oplus C)$. Therefore, two complicated shapes can be summed by expressing them as the union of simpler shapes, summing those, and unifying these partial sums.

The Minkowski average is a similar operation defined as $\frac{A \oplus B}{2} = \{\frac{a+b}{2} | a \in A, b \in B\}$ (an extension to more than two summands is straightforward, similarly to Minkowski sum). Figure 2(a) shows the Minkowski average (blue rectangle) and Minkowski sum (green rectangle) of the two black rectangles. Note that the Minkowski average of two axis-aligned rectangles is an axis-aligned rectangle whose corners are the averages of the respective corners of the two rectangles. Another important extension of the Minkowski average and Minkowski sum operations is to...
Figure 2. Illustrative comparison of Minkowski average (blue rectangle) and Minkowski sum (green rectangle) of two rectangles (a) and weighted Minkowski average (blue rectangle) of two rectangles, where the weight of the small rectangle is 3 times the weight of the big rectangle (b).

assign different weights to each of the summands. For \( N \) subsets \( A_1, A_2, \ldots, A_N \) each of them assigned with different weights \( \alpha_1, \alpha_2, \ldots, \alpha_N \), their weighted Minkowski average is defined as

\[
\alpha_1 A_1 \oplus \alpha_2 A_2 \oplus \cdots \oplus \alpha_N A_N = \left\{ \frac{\sum \alpha_i a_i}{N} | a_i \in A_i \right\}.
\]

Figure 2(b) demonstrates the weighted Minkowski average of two rectangles. It can be seen that the result (depicted as blue rectangle) is more similar to the rectangle with bigger weight (the right one), both in size and in location.
4 Safe Zone Monitoring

We suggest a rigorous solution to the monitoring problem, while introducing an optimality criterion. At the heart of our approach are subsets of the data space which will constitute optimal safe zones (SZs) at the nodes. As the name indicates, these SZs have the following property: as long as all nodes remain within their respective SZs, we are guaranteed that the function will not cross the threshold (thus no communication is required), and in addition, of all subsets satisfying this property, the SZs are optimal in the sense that the data will remain in them for the longest duration possible.

Assume then that we are monitoring a general function \( f \) that is evaluated on a weighted average of the local vectors, and we need to send an alert whenever its value increases above \( T \). Let us stick to the simple case in which two nodes are present and the function \( f() \) is evaluated at the average of the two local vectors. Defining \( S = \{ z \mid f(z) \leq T \} \), this means that if the average of the local vectors is not in \( S \) any more, at least one node must take some action (involving communication with other nodes) to decide whether \( T \) was crossed – that is, its vector must be outside its SZ. Denote the local vectors by \( x \) and \( y \) and their sought SZs by \( S_x, S_y \). The simple definition of the set \( S \) is the essence of the proposed framework: we do not impose conditions on the range values \( f(z) \) but on the domain vectors \( z \).

What are optimal SZs? They must be correct, so we must have \( (x \in S_x) \land (y \in S_y) \Rightarrow \frac{x+y}{2} \in S \) (if this condition does not hold, the global vector may wander outside \( S \) but no action will be taken). But the SZs also need to be large, since each time a local data vector wanders out of its safe zone, the corresponding node must initiate communication, and the goal is to minimize the number of these communication events. Assuming distributions \( p_x, p_y \) on the data at the nodes, we can formulate a constrained optimization problem as follows:

\[
\text{Maximize} \int_{S_x} p_x \, dx \int_{S_y} p_y \, dy \\
\text{Subject to:} \quad \frac{S_x \oplus S_y}{2} \subset S
\]

where \( \frac{S_x \oplus S_y}{2} = \{ \frac{x+y}{2} \mid x \in S_x, y \in S_y \} \), so it's just the Minkowski sum [40] of \( S_x, S_y \), in which every element is divided by 2 (or the Minkowski average). In the next subsection we further discuss Minkowski sums. For more nodes, it is still the Minkowski sum of the local SZs, divided by the number of nodes. (We mention in passing that when the global vector is the
concatenation of the local vectors, the Minkowski sum is replaced by the Cartesian product, but the solution methodology still applies.)

The practical meaning of the constraint $S_x \oplus S_y \subset S$ is straightforward: assume for example that the local vectors are the local contingency tables of a feature-category pair, built from the data at the respective nodes, and that $S$ is the set of contingency tables satisfying a certain condition (e.g., that the mutual information of the pair is above a certain threshold $T$). The solution then assigns to each node a certain subset of contingency tables, with the constraint that as long as the local contingency tables remain within their respective subsets (SZs), their average is a contingency table whose mutual information is greater than $T$. The maximization of the target function $\int_{S_x} p_x dx \int_{S_y} p_y dy$ means that, under this constraint, we expect the time it will take one of the local contingency tables to wander out of its SZ to be maximal; since communication is required only when the local table leaves its SZ, this is equivalent to minimizing communication.

Since an alert should be sent only if the global vector is not in $S$ anymore, then, as long as every local vector is inside its SZ and the Minkowski average of the SZs is contained in $S$, we can be certain that no alert needs to be sent. A schematic example is presented in figure 3.

The solution of the constrained maximization problem is optimal for reducing communication. The geometric constraint, expressed via the Minkowski average, guarantees correctness – as long as it holds, the threshold was not crossed. The probabilistic expression we propose to optimize – the product of the probabilities of the local data to remain inside their SZs – guarantees minimal communication.

The geometric interpretation of these conditions is that the SZs try to match the shape
of the p.d.f. (probability density functions) at each node, so as to maximize the probability of the local data to fall in its SZ. For example, a distribution which is wide along the x-axis and narrow along the y-axis should be assigned an SZ which is wide horizontally and narrow vertically. In addition, the Minkowski average of the SZs should tightly approximate $S$ (but remain bounded inside $S$). If it fills a relatively small part of $S$, this means that the SZs can be enlarged and the value of the optimized function increased. This point is further explained in figure 4.

We now extend the optimization problem that was formulated above in order to handle $n$ nodes:

$$\text{Maximize} \prod_{i=1}^{n} \int_{S_i} p_x \, dx$$

Subject to: $S_1 \oplus S_2 \oplus \ldots \oplus S_n \subset S$.

Note that both the objective function and the constraint can be seamlessly extended due to the independency in the distributions of the data and due to the properties of the Minkowski average operation as described in section 3.2. All the insights above remain applicable to the extended problem, with the number of optimization variables being the only difference for our purpose.
Figure 4. Top: left and right denote samples of two-dimensional data tuples at both nodes. The p.d.f.s at both nodes are Gaussian (normal) distributions. $S$, which must contain the Minkowski average of the two SZs, is the dotted ellipse in the middle. The point cloud in the middle is a sample of the global data tuple, obtained by averaging the data at both nodes. The depicted SZs fit the local data well, but alas are illegal, since their Minkowski average (continuous dark line in the middle) is not inside $S$. Bottom: two legal SZs are depicted. The SZ on the left tries to match the data, while satisfying the constraint. This figure demonstrates the trade-off between the fit of the SZs to the data and the necessity to maintain their Minkowski average inside $S$. Note that the Minkowski average "sticks" to $S$; this is because the SZs must be as large as possible (meaning that the integral of the p.d.f. over them is maximal). If the Minkowski average does not "stick" to $S$, it is possible to enlarge the SZs and hence the integral of the p.d.f. over them, meaning the original SZs were not optimal.
5 Algorithmic Framework

This section outlines the monitoring algorithm and covers different facets of the optimization problem that must be solved in order to compute the optimal SZs.

The optimization of the SZs is initially performed by a coordinator node. Then each node is assigned its SZ, and it remains silent as long as its local data vector is inside the SZ. Once the local vector wanders out of the SZ ("local violation"), the node notifies the coordinator. The coordinator can then issue an alert, or poll the other nodes in order to capture an exact snapshot of the system. Algorithm 1 outlines the version in which an alert is issued every time a local violation occurs.

5.1 Practical Computation of the Safe Zones

Given a real-life monitoring problem, we need to compute the optimal SZs. We start with the problem parameters; for the sake of simplicity, assume as before that two nodes are present, and that the data is two-dimensional, though the method described here can be applied in any dimension.

The problem’s input parameters are then:

- A function of two variables \( f() \) and a threshold \( T \).
- Two nodes, \( N_1, N_2 \), with initial data vectors \( v_0^1, v_0^2 \). At time \( t \) the \( i \)-th node holds a vector denoted \( v_t^i \). As discussed in Section 1, we assume that the global vector, which represents the entire system at time \( t \), equals \( v^t = \frac{v_t^1 + v_t^2}{2} \).
- Probability distributions \( p_1, p_2 \), at the nodes. \( p_i \) is the probability density function (p.d.f.) of the data at the \( i \)-th node. These can be, e.g., Gaussian [43], random walk [41], uniform, or any other p.d.f.
- Assume without loss of generality that \( f(\frac{v_t^1 + v_t^2}{2}) > T \), so we need to submit an alert when \( f(\frac{v_t^1 + v_t^2}{2}) \leq T \). Denoting as before \( S = \{ z | f(z) > T \} \), we need to submit an alert when \( \frac{v_t^1 + v_t^2}{2} \notin S \).

Given this input, the algorithm’s output are two subsets \( S_1, S_2 \), of the Euclidean plane \( \mathbb{R}^2 \), which maximize \( \int_{S_1} p_1(z) dz \int_{S_2} p_2(z) dz \), subject to the constraint \( \frac{S_1 \oplus S_2}{2} \subset S \).

5.2 Constructing the Optimization Problem

The first step in solving an optimization problem is determining the parameters to optimize over. Here, the space of “parameters” is huge— all subsets of the plane are candidates for
Algorithm 1 Safe Zone Monitoring Algorithm

**Input:** \( n \) nodes \( \{N_1, N_2, \ldots, N_n\} \) with data distribution \( p_i \) for each node \( i \), and a stream of observations \( v_i(t) \) for each node \( i \) and time step \( t \). A monitored function \( f() \), a threshold \( T \) and a domain \( S \) such that \( S = \{z \mid f(z) > T\} \).

**Initialization:**

1. Solve the following optimization problem:

   \[
   \text{Maximize} \prod_{i=1}^{n} \int_{S_i} p_x dx \\
   \text{Subject to:} \quad \frac{S_1 \oplus S_1 \oplus \ldots \oplus S_n}{n} \subset S
   \]

2. Assign each node \( N_i \) with its appropriate SZ \( S_i \).

**Main:** Repeat the following forever

1. For each new observation in each node \( v_i(t) \), test whether it falls inside the node’s SZ \( S_i \).
2. If \( v_i(t) \not\in S_i \) the system alerts.

\( S_1, S_2 \). This is an infinite dimensional, nonlinear space. While such an optimization task may be feasible in certain cases, generally speaking it is impossible to find the exact solution. Thus we suggest restricting the optimization to a parametric family of shapes, denoted by \( P \). This family, from which the SZs will be chosen, should satisfy the following requirements. First, \( P \) should not be too restricted, so that its members can reasonably approximate every subset of the plane which is a viable candidate for a SZ. If this does not hold, the solution may be grossly suboptimal. Second, the SZs should not be too complicated: it should be possible to define them using relatively few variables. Third, it should not be too difficult to compute the integral of the p.d.f. over members of \( P \); otherwise, the calculation of the objective function will be too costly. Last, it should be relatively easy to compute the Minkowski average of any members of \( P \), as this is an integral part of the calculation of the constraints.

The last three conditions are imposed for efficiency. Note that every node has to test whether its local data vector is inside its SZ; if the definition of the SZ is complicated, the test will consume time and energy. If computing the integrals of the p.d.f. or the Minkowski average is very time consuming, the optimization process may be lengthy, rendering the algorithm
impractical. The following subsections describe various types of possible SZs.

5.2.1 Polygons as SZs

Polygons are natural candidates for SZs, since they are flexible enough to approximate any domain in Euclidean space. Testing the local conditions at the nodes is easy, since very fast algorithms exist to test whether a point is inside a polygon; for a thorough survey, see [22]. It is also relatively easy to compute the integral of any reasonable p.d.f. on a polygon. Last but not least, there is extensive literature on computing the Minkowski sum of polygons. Hereafter, we will therefore assume that $P$, the family of SZs we optimize over, is a certain class of polygons. Nevertheless, we need to find an appropriate subclass of this family. Arbitrary polygons cannot be allowed because they have an unbounded number of degrees of freedom. The most direct solution is to fix some $n$ and define $P$ as the set of all polygons with no more than $n$ vertices (e.g., if $n$ is four, the SZs will consist of all quadrangles and all triangles). However, while the Minkowski average of polygons is also a polygon, it can have more vertices than each of the summands – as demonstrated in Figure 1(b). As a matter of fact, the Minkowski average of two general polygons, with $m$ and $n$ vertices, may have $O(m^2n^2)$ vertices [45]. So if many nodes are present, even if each has an SZ which is a polygon with $n$ vertices, the number of vertices in the Minkowski average may be very high, and so will be the complexity of computing it. In the next subsections we review various classes of polygonal SZs and discuss the properties of each.

5.2.2 Axis-aligned rectangles

Computing the Minkowski average of axis-aligned rectangles is particularly easy. As seen in section 3.2, the Minkowski average of axis-aligned rectangles is an axis-aligned rectangle whose corners are the averages of the respective corners of the summands.

Therefore, if the SZs are axis-aligned rectangles, the computation of their Minkowski average is very simple, as demonstrated in Figure 5. Computing the integral of a p.d.f. on a rectangular region is usually easy as well. The problem is that such SZs may be too simple and provide only a relatively poor fit to $S$. However, for data whose distribution is not too eccentric, even these simple SZs yielded results much better than those in [42, 43] (Section 8).
5.2 Constructing the Optimization Problem

5.2.3 Rotated rectangle

Limiting the SZs to axis-aligned rectangles implies that their Minkowski average is an axis-aligned rectangle as well, and as such may result in a rather poor fit to $S$. This can be improved by using free rectangles instead. By allowing rectangles to be rotated, i.e., not aligned to the axes, the SZs can be much more accurate. Figure 6 illustrates such a case. However, when different SZs are rotated in different orientations, calculating their Minkowski average is no longer trivial.

![Figure 6](image_url)

Figure 6. Approximating $S$ in the shape of an ellipse with inscribed rectangles. Left: poor approximation when restricted to axes-aligned rectangle. Right: better approximation with a general rectangle.

5.2.4 Union of rectangles

A more complex class of SZs we investigated is a union of rectangles, particularly SZs which are a union of two rectangles. In this case, the Minkowski average can be more complicated than
Constructing the Optimization Problem

5.2 Constructing the Optimization Problem

of the summands and allow a tight approximation to $S$. The calculation of the Minkowski average uses the identity $(\bigcup A_i) \oplus (\bigcup B_j) = \bigcup (A_i \oplus B_j)$, and since each element is easy to compute, the entire sum is straightforward to compute as well (figure 7). The union of more than two rectangles can be used, but the marginal improvement in the quality of the SZs might not justify the extra computational cost. While computing the integral of a p.d.f. on a rectangular region is easy, integrating over a union of rectangles is more expensive as the overlapping areas must be counted only once. This requirement introduces the problem of finding intersections of rectangles, a problem which gets more complicated as the number of rectangles increases.

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{figure7.png}
\caption{Two nodes $A,B$ with non-Gaussian distributions (green dots mark sample data tuples). The SZs at both nodes are a union of two rectangles. $S$ is depicted by a circle, blue dots are samples of the global data tuples (average of the data at the nodes), and the Minkowski average of the two SZs is in green. Note that, unlike the case in which both SZs are single rectangles, the Minkowski average is more complicated than both SZs, and can thus achieve a better fit to $S$.}
\end{figure}

5.2.5 Convex polygons

As noted in Section 5.2.1, the Minkowski average of general polygons might be complicated. For convex polygons the situation is better: it is known that the Minkowski average of two convex polygons with $m$ and $n$ vertices is also convex and has at most $m + n$ vertices [45]. Moreover, efficient algorithms for computing the Minkowski average in this case have been developed in [45]. Therefore, it is possible to use convex polygons with a bounded number of vertices as SZs. Nodes with complicated data distributions can be assigned polygons with more vertices.
5.3 Approximation Quality of Safe Zones

When using restricted families of polygons as SZs, the question is how far these SZs are from the optimal ones. An important measure is the ability of the Minkowski average to occupy a relatively large portion of the set $S$. The SZs should also be able to capture a substantial part of the area of the optimal SZs. Theoretically speaking, and considering the most general scenario, it is impossible to compute such bounds; given any family $P$ of candidates for SZs with a certain (finite) number of parameters, and given a constant $\epsilon > 0$, it is trivial to construct a subset $L$ of the plane which is pathological enough so that every member of $P$ inscribed in $L$ covers at most a portion $\epsilon$ of $L$'s area. However, our experience with typical functions which need to be monitored is that the set $S$ is "nicely behaved." For example, it is often convex, and if not, it contains a large convex subset. If that is the case, we can bound from below the measure of "tightness" of the Minkowski average inscribed in $S$. We start with the following lemma:

If $S$ is convex, the optimal SZs are also convex.

**Proof** For the sake of simplicity, we'll prove for two SZs. Assuming that $\frac{S_1 \oplus S_2}{2} \subset S$ where $S$ is convex, we'll prove $C(S_1) \oplus C(S_2) \subset S$, where for every set $Q$, $C(Q)$ denotes the convex hull of $Q$. Let $x_1, y_1 \in S_1$; $x_2, y_2 \in S_2$, and $0 \leq \lambda_1, \lambda_2 \leq 1$. We need to prove that $v \equiv \frac{\lambda_1 x_1 + (1 - \lambda_1) y_1 + \lambda_2 x_2 + (1 - \lambda_2) y_2}{2} \in S$. Assume without loss of generality that $\lambda_2 \leq \lambda_1$. Since $\frac{S_1 \oplus S_2}{2} \subset S$, it follows that $\frac{x_1 + x_2}{2}, \frac{x_1 + y_2}{2}, \frac{y_1 + y_2}{2} \in S$. But since $S$ is convex, it holds that $v \equiv \lambda_1 \frac{x_1 + x_2}{2} + (1 - \lambda_1) \frac{x_1 + y_2}{2} \in S$, as the last expression is a convex combination of elements of $S$ (note that all the coefficients are positive and their sum equals 1). Therefore, if one of $S_1, S_2$ was not convex, its convex hull would have been a viable candidate for the respective SZ which is strictly larger than $S_1$ or $S_2$, thus contradicting their optimality.  

5.3.1 Rectangles

As mentioned in subsection 5.2.3, free rectangles can cover better the p.d.f. and achieve a tighter approximation of $S$. In [39] it was shown that every convex subset of the plane, $C$, has an inscribed rectangle $R$ whose area is at least half of $C$'s area. This rectangle may not be axis-aligned, thus making it harder to calculate the Minkowski average of the SZs. However, defining the SZs to be rectangles with similar orientation will keep this calculation simple.
5.3.2 Convex polygonal SZs

In Lemma 5.3 it was noted that if $S$ is convex, the optimal SZs are convex as well. Assume we are using SZs which are convex polygons with $k$ vertices, as discussed in Section 5.2.5. We are interested in whether these SZs can provide a reasonable approximation to the optimal SZs. The following result provides the answer.

From [20]: let $C$ be a convex subset of the plane, and let $k$ be a fixed integer. Denote by $C_k$ the maximal value of the ratio $\frac{A(P_k)}{A(C)}$, where $A$ stands for area and $P_k$ is any convex polygon with $k$ vertices inscribed in $C$. Then the minimal value of $C_k$ is obtained when $C$ is a disk. Intuitively, we can see that disks (and spheres in higher dimensions) are the convex sets which are the hardest to approximate by inscribed polyhedra. Using this result, it is straightforward to see that as $k$ increases, $C_k$ behaves as $1 - \frac{\alpha}{k^2}$ for some small constant $\alpha$. Similar results hold for higher dimensions.

This theorem guarantees that the family of convex polygons with a bounded number of vertices provides a good approximation to general convex sets. Hence, using such polygons as SZs will provide a good solution when the optimal SZs are known to be convex. For nonconvex sets, we can use the very rich body of work on decomposing or approximating general sets by a small number of convex sets, e.g., [8].
6 Scalability

In the previous section we introduced the algorithm for determining the local constraints at the nodes. This section starts by describing the major drawback of the algorithm – its lack of scalability, which stems from its reliance on solving a difficult optimization problem with many parameters. We then describe how to use clustering algorithms in order to reduce the number of parameters and overcome this hurdle.

6.1 Number of Parameters in the Optimization Process

The difficulty of the optimization problem is in the number of parameters. The search space exponentially increases with the number of parameters, as each SZ depends on the other SZs: enlarging one will necessitate reducing the others due to the constraint on their Minkowski average.

The number of parameters which should be optimized over depends on the number of nodes and the type of SZ at each node. For example, an axis-aligned rectangle is defined by four parameters, and a general rectangle by five. A quadrangle is defined by eight parameters, and so forth (Figure 8). If each SZ is defined by \( k \) parameters and \( n \) nodes are present, there are \( kn \) parameters to optimize over. However, if this number is too large, we can determine some of the SZs using other methods (such as in [42, 43]) and optimize only over some of the nodes – for example, those having data distributions which are wide or complicated relative to the other nodes.

![Figure 8. Four parameters \((x_1, y_1, x_2, y_2)\) are required to define an axis-aligned rectangle, five \((x_1, y_1, x_2, y_2, l)\) for a general rectangle, and eight \((x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4)\) for a quadrangle.](image-url)
In previous sections, we presented the optimization problem of constructing local SZs and showed that the exact solution is hard to find because of the infinite space of SZ definitions. Nonetheless, using TOMLAB’s SNOPT package [24], we were able to compute an approximated solution, but only for a few nodes \((n)\) and simple SZ shapes \((k)\). In this section we describe clustering methods for reducing the number of optimization parameters \((kn)\), in order to support large scale systems.

### 6.2 Iterative Clustering Algorithm

The common goal of the presented iterative clustering algorithms is to ease the optimization problem by reducing the number of optimization parameters. We trade one difficult optimization task for many, relatively easy, optimization tasks which produce an approximated solution. The underlying idea is to split the nodes into \(c\) different groups, and assign a local SZ for each group, where all the nodes in one group will be assigned an identical SZ. By selecting \(c \ll n\), the number of optimization parameters \(kc\) is much smaller than the original \(kn\) parameters, greatly reducing the computational cost.

A simple solution to the SZ optimization problem is to perform the clustering operation once. We denote this straightforward solution as the flat algorithm. However, by exploiting the characteristics of the Minkowski average operation, we developed an iterative clustering algorithm, denote the hierarchical clustering algorithm, which further repeats the clustering process to obtain more accurate SZs. After the flat clustering algorithm is invoked, we expand each group of nodes into a new set of sub-clusters and solve \(c\) new independent optimization problems. In each such problem, the new \(S\) (the range of the given function) will be the SZ that was assigned to this group of nodes in the previous optimization iteration. We formally prove below how doing so maintains the correctness of the solution. This process of spawning new optimization problems for sub-clusters of nodes can be repeated until a group of nodes consists of a single node or a certain stopping condition is met. In the experiments section we show how repeating this process over a few iterations can improve the quality of the SZs within a reasonable computational cost.

We now formally describe the iterative clustering algorithm. The algorithm is initiated by clustering the given \(n\) nodes to \(c\) clusters, where the distance measure between nodes (which is used for the clustering) is based on the distance between the data distributions in them (e.g., Kullback-Leibler divergence [31] or Kolmogorov–Smirnov test [19]). Each cluster is defined by
(\(C_i^0, n_i^0, p_i^0\)), where \(C_i^0\) is the \(i\) cluster in step 0, \(n_i^0\) is the number of nodes in this cluster, and \(p_i^0\) represents the data distribution among all the nodes in the cluster. After the clustering phase, a Minkowski optimization problem is defined, using the data distribution of the clusters \( (p_i^0) \) and the given function range value \((S)\). Since the number of nodes in each cluster can vary, we use a weighted Minkowski average constraint, i.e., 
\[
\prod \frac{n_i^0SZ_i^0\oplus\cdots\oplus n_{c-1}^0SZ_{c-1}^0}{n_i^0SZ_i^0\oplus\cdots\oplus n_{c-1}^0SZ_{c-1}^0} \subset S.
\]

The solution of this optimization problem defines a local \(SZ_i^0\) for each cluster \(C_i^0\). In iteration \(t\), the nodes in cluster \(C_i^{t-1}\) are clustered to \(c\) sub-clusters \( (C_i^t, n_i^t, p_i^t)\), and new \(S\) are computed recursively for each. In each recursive step, the Minkowski average of the \(S\)s must be contained in the cluster’s \(SZ\) \((SZ_i^{t-1})\) instead of in \(S\), i.e., 
\[
\prod \frac{n_i^0SZ_i^0\oplus\cdots\oplus n_{c-1}^0SZ_{c-1}^0}{n_i^0SZ_i^0\oplus\cdots\oplus n_{c-1}^0SZ_{c-1}^0} \subset SZ_i^{t-1}.
\]

The algorithm may expand the complete tree of clusters until reaching the leaves (where each cluster consists of a single node). The algorithm is presented in Algorithm 2.

**Algorithm 2 Iterative Clustering Algorithm**

*Input*: \( n \) nodes with data distribution \( p_i \) for each node \(i\), a monitoring function \( f \) and a clustering parameter \(c\).

1. Let \( t = 0 \) be the iteration counter.
2. In step \( t\), cluster the \( n \) nodes into \(c\) clusters, using the distance between the data distributions as a distance measure. Each cluster \(C_i^t\) is defined by its number of nodes \(n_i^t\) and its data distribution \(p_i^t\).
3. Compute local \(SZ_i^t\) for each cluster \(C_i^t\), using a weighted Minkowski optimization problem:

\[
\text{Maximize } \prod \int p_i dx dy, \text{ s.t.: } \prod \frac{n_i^0SZ_i^0\oplus\cdots\oplus n_{c-1}^0SZ_{c-1}^0}{n_i^0SZ_i^0\oplus\cdots\oplus n_{c-1}^0SZ_{c-1}^0} \subset S, \text{ where } S \text{ is the range value of } f.
\]

4. For each cluster \(C_i\), if \(C_i\) contains more than \(c\) nodes, repeat steps 2-3 where:
   - (a) Replace the global \( n \) nodes with each cluster \(n_i^t\) nodes.
   - (b) Replace \( S \) with the cluster’s safe zone \((SZ_i^t)\).
   - (c) \( t = t + 1 \).
5. Otherwise, compute the local \(SZ\) for each node in \(C_i\) and terminate.

Figure 9 presents a simple example of the iterative clustering algorithm over 8 nodes. Given a clustering parameter \(c = 2\), the first iteration splits the nodes into two groups (4 nodes, 4 nodes), as presented in figure 9(a). In the next iterations both clusters are re-split until each node is assigned its \(SZ\); see figure 9(b).
6.3 Algorithm Correctness

In order to prove the correctness of the algorithm, a simple property of the Minkowski sum operation is exploited:

(Minkowski Property) Given three shapes \( A, B, C \) such that \( C \subset A \), then \( C \oplus B \subset A \oplus B \).

According to the definition of Minkowski sum for each \( x \in C \oplus B \), exist \( c \in C \) and \( b \in B \) such that \( x = c + b \). Since \( C \subset A \), then \( c \in A \) and \( x \in A \oplus B \). Therefore \( C \oplus B \subset A \oplus B \).

The iterative clustering algorithm defines valid SZs for each node, i.e.,

\[
\frac{S_{Z_0}^{t_1} \oplus S_{Z_1}^{t_1} \oplus ... \oplus S_{Z_{n-1}}^{t_1}}{n} \subset S.
\]

We prove by induction over the number of iterations \( t \). For \( t = 0 \),

\[
\frac{n_0^{t-1} S_{Z_0}^{t-1} \oplus ... \oplus n_c^{t-1} S_{Z_c}^{t-1}}{n} \subset S.
\]

Let assume that after \( t - 1 \) iterations it holds that

\[
\frac{n_0^{t-1} S_{Z_0}^{t-1} \oplus ... \oplus n_c^{t-1} S_{Z_c}^{t-1}}{n} \subset S.
\]

In iteration \( t \), a local SZ is defined for each node. According to the algorithm, the safe zone \((SZ_i^t)\) of each of \( n_i^{t-1} \) nodes in each cluster \( C_i^{t-1} \) maintains the following property:

\[
\frac{S_{Z_j}^{t} \oplus ... \oplus S_{Z_i}^{t}}{n_i^{t-1}} \subset S_{Z_0}^{t-1}.
\]

Using the Minkowski property, we can replace each \( SZ_i^{t-1} \) in equation 1 with the SZ’s definition in equation 2, which proves our theorem that

\[
\frac{S_{Z_0}^{t} \oplus S_{Z_1}^{t} \oplus ... \oplus S_{Z_{n-1}}^{t}}{n} \subset S.
\]
6.4 Number of Clusters

The computational cost of the algorithm, as we have mentioned, is subject to the number of clusters \((c)\). By selecting a small value of \(c\), the optimization problem might be solved very fast (as the number of optimization parameters is drastically reduced) but the clusters might be nonhomogeneous and the resulting SZs will probably be of low quality. However, when a large value of \(c\) is used, the algorithm might produce accurate SZs but at high computational cost. This tradeoff introduces the necessity of a method for wisely selecting the values of \(c\).

Given the clustering algorithm, a simple solution is to execute the algorithm for different values of \(c\) and determine which value yields the best SZ. However, this method requires solving the optimization problem for each value of \(c\), which can be very expensive. Therefore, we suggest using the homogeneity of the clusters as a quality test. The quality of the clusters is evaluated using the distance between the nodes. We wish nodes with similar p.d.f. to have similar SZs, and therefore the distance between nodes should be small for nodes with similar p.d.f.s. Figure 10 demonstrates the intuition behind this by showing the optimal SZs in two simple systems, each consisting of two nodes. In figure 10(a) the nodes have similar data distributions, and the two SZs are similar as a result. In figure 10(b) the data distributions are different, which results in different SZs. Our goal is to choose the smallest value of \(c\) that yields a good clustering homogeneity.

6.5 Number of Iterations vs. Optimality

The computational demand of the iterative clustering algorithm (Algorithm 2) is also derived from the number of iterations \((t)\). If the clustering creates a balanced tree of clusters, the number of iterations is bounded by \(\log_n n\). However, if the tree is not balanced (a phenomenon we found to be common in real data), \(t\)'s value can greatly increase (as presented in figure 11), up to \(O(n)\) in the worst case, thus increasing the computational demand. In a case like this, choosing a small value of \(t\) might yield bad SZs (as depicted in figure 12). Therefore, we need to define a method to choose the number of iterations.

In order to intelligently limit the number of iterations, we present an alternative stopping criterion. In the basic version (Algorithm 2), the algorithm terminates when there is a single node in each cluster. We suggest using the clustering homogeneity as a stopping criteria. If the nodes in a certain cluster are similar enough to each other in terms of their data distribution (i.e., they are highly homogeneous), the cluster's SZ can be assigned to all the nodes in that cluster,
Figure 10. Example of SZ similarity for (a) highly homogeneous and (b) not highly homogeneous clusters. It can be seen that the optimal SZs are similar for nodes with similar distributions but different for nodes with different distributions.

Figure 11. An example of non-balanced clustering execution of the iterative algorithm
6.6 Distance Measure and Cluster Representation

In order to cluster nodes according to their data distribution, we must define a distance measure between two nodes and a representation for a group of nodes.

6.6.1 Distance measure

Measuring the distance between two entities (nodes) in the system is a basic construct in every clustering algorithm, as these algorithms group together nodes based on their distance. For two nodes, the Kullback–Leibler (KL) divergence [31] can be used if the p.d.f. of the data is known. The KL divergence is a widely used tool in statistics which measures the difference between two probability distributions. Alternatively, when the p.d.f. describing the data is unknown, the distance can be determined using samples of the observation. The Kolmogorov–Smirnov (KS)
test [19] and Earth mover's distance (EMD) [38] are popular distance measures for discrete distributions (or two sets of samples). In addition to discrete distance measures such as KS-test and EMD, it is possible to use GMM (Gaussian Mixture Model) in order to approximate the discrete data at the nodes and then measure the KL divergence between the GMMs. The authors of [23] review several techniques for computing the KL divergence between GMMs. Other statistical distance measures such as Bhattacharyya distance [29] might be applied, depends on the data distribution in the nodes.

6.6.2 Cluster representation

As described in algorithm 2, we assume each cluster has its own data distribution $p_i$. Therefore, once the nodes were clustered into groups we need to define the p.d.f. of each group as if it was a single node. If the data distribution of each node is known, it might be possible to analytically calculate the expected distribution of the group of nodes. If such analytic calculation is not feasible, typical samples can be produced for each node, combined and approximated with GMM. If no good approximation to the p.d.f. of each node exists, the samples of all the nodes in the group can be joined together and serve as input for GMM.
7 Probabilistic Approach

The algorithm described so far is rigid in the sense that it demands absolute accuracy: *every* threshold crossing must be reported. While this is desirable for some applications, in many cases we may allow a small error probability. In detection theory, it is common practice to allow for *false negatives*, in which the sought event occurs but the detection algorithm errs. Allowing a rather low probability for false negatives often reduces the running time considerably. Furthermore, the noisy nature of real data often makes it too strict a requirement to report all threshold crossings, since random noise may cause the function to increase slightly beyond the threshold.

In order to apply this paradigm here, we replaced the condition $\frac{S_1 \oplus S_2}{2} \subset S$ (and its generalization to more than two nodes), which guarantees that every threshold crossing will be reported, with the following “softer” condition:

$$\Pr \left[ \forall_i (v_i \in S_i) \left| \frac{\sum v_i}{N} \notin S \right. \right] < \epsilon \ (N = \text{number of nodes}),$$

where $v_i, S_i$ are the local vector and SZ at the $i$-th node, and $\epsilon$ is a small positive constant representing the error probability. This condition guarantees that if the global system vector (which equals the average of the local vectors) wanders out of $S$, then with a probability of at least $1 - \epsilon$ this will be reported by at least one of the nodes. Our experiments (Section 8.6) demonstrate that this relaxation of the threshold crossing constraint, even with a very small error probability, reduced communication considerably.

Our work on the probabilistic approach is merely a preliminary effort required for the more important task of estimating the possible performance improvement. It is crucial to formalize a method for defining a new domain $S'$ such that $\Pr \left[ \forall_i (v_i \in S_i) \left| \frac{\sum v_i}{N} \in S' \right. \right] \geq 1 - \epsilon$. In our experiments we used a naive heuristic in which we stretched $S$ (i.e., increased the ellipse’s radii or rectangle’s edges) symmetrically, without considering the distribution of the data. However, we did consider the number of nodes in the system. Note that the more nodes there are in the system, the smaller the variance in the global distribution will be. Consider, for example, a system with $n$ nodes in which the data in the nodes is *i.i.d.* and is distributed according to a normal distribution with the following parameters $X_i \sim N(\mu, \Sigma)$. The sum of these distributions is distributed as $\sum X_i \sim N(n\mu, \sqrt{n}\Sigma^2)$, and the global data, or the average of the distributions in the nodes, is distributed according to $\frac{1}{n} \sum X_i \sim N \left( \mu, \frac{\sqrt{n}^2}{n} \right) = N \left( \mu, \frac{\Sigma}{\sqrt{n}} \right)$. The variance of the global data thus decreases in a square root proportion with respect to the
Figure 13. Illustration of a system applying the probabilistic approach. The system consists of 8 nodes, with green and red dots denoting non-violating and violating local observations respectively. The turquoise dots in the center depict the global observations. The smaller black circle portrays the domain of the threshold $S$, where the bigger dashed black circle portrays the extended domain $S'$. The blue hexagons describe the SZs and their Minkowski sum is the red line tightly bounded by $S'$. It is easy to see that the Minkowski sum is much larger than would have been possible given the original $S$, whose magnification results in much bigger SZs that cover many more observations.

number of nodes. As a result, for systems with many nodes, we could stretch $S$ significantly in order to achieve some desired $\epsilon$, thus allowing a significant increase in the SZs’ size. Section 8.6 describes our empirical evaluation of the probabilistic approach. The setup of a system applying the probabilistic approach is presented in figure 13.
8 Empirical Evaluation

In order to test the algorithms, and their extensions, we ran them on both synthetic and real data. We tested various families of SZs, and compared the results with those of [42, 43]. The SZs were optimized using TOMLAB’s SNOPT package [24]. As in [42, 43], performance was measured by the number of messages sent by the nodes to the coordinator; every time the local data at a node wanders out of its SZ, it notifies the coordinator. In the real data we used (Section 8.1), each run consisted of 1,500 timestamps, with each timestamp corresponding to a simultaneous measurement at the distinct nodes.

8.1 Experimental Framework

The experimental data we used consist of air pollutant measurements, taken from "AirBase – The European Air Quality Database" [1]. We used sulphur dioxide (SO2), corresponding to the x-axis of the scatter diagrams in figure 14, and nitrogen monoxide (NO), corresponding to the y-axis. Both were measured in micrograms per cubic meter. Nodes correspond to sensor clusters at different geographical locations. The data at different nodes greatly vary in size and shape, as can be seen in figure 14(a,b). The data were approximated by a GMM (Gaussian Mixture Model), using a Matlab routine [28], as depicted in figure 14(c), in order to compute the integral of the p.d.f. on the respective SZs.

As a threshold function we use a quadratic function \( \frac{x^2}{20} + \frac{y^2}{100} < 1 \), which can be viewed, as is common in statistics, as a Gaussian distribution over the concentrations of the pollutants. Defining such a threshold is a rather common operation in statistics. The range of this function \( S \) is defined by an axis-aligned ellipse with the x-axis of length 20 and y-axis length of 100, centered at the origin.

8.2 Various Types of SZs

In order to examine the representational power of various types of SZs, we tested the following SZs for the AirBase data: squares, rectangles, union of two rectangles, and polygons with up to 10 vertices. This means that the number of parameters to optimize over (Section VII-D) ranged from 3 (for squares) to 20 (for decagons). Typical results (for a subset of two nodes) are
8.2 Various Types of SZs

Figure 14. Air pollutant measurements: (a,b) two scatter diagrams of the measurements at different nodes; (c) two of the GMM elements superimposed on the data. Note the very different data distributions in the scatter diagrams.
depicted in figure 15, for rectangles, quadrangles, and octagons. Each of the plots in figure 15 depicts the global data (blue dots) and the set $S$ (ellipse, pale blue). Left and right figures show the local data vectors: the green dots denote local data for which no violation was reported, or for which a violation was correctly reported (that is, both a local and global violation occurred). The red dots denote local data vectors for which a local violation took place, but no global violation; this means that the node in which the local violation occurred unnecessarily initiated a communication operation. The shapes outlined on the left and the right depict the SZs of the respective nodes (since the left one is much smaller, a close-up is depicted as well). Note that as the family of candidates SZs increases, the algorithm performs better: geometrically, this is reflected in the fact that the Minkowski average is a tighter fit to $S$ and the SZs fit the data better.
Figure 15. Comparison of different SZ types: (a) rectangles, (b) quadrangles, (c) octagons
8.3 Complexity of the Safe Zone Type

In figure 16, the different SZ models are compared in terms of performance and running time. The overall results are depicted, as well as the marginal improvement of models of increasing complexity. These results are for 10 nodes with a total of 1,500 timesteps.

![Violations and Cost vs. Free Optimization Parameters](image1)

(a) Performance and Cost of Different Types of Safe Zones

![Marginal Improvement of Violations vs. Free Optimization Parameters](image2)

(b) Marginal Improvement in Performance

Figure 16. The performance and cost of different SZ types: (a) performance and cost for various SZ models; (b) marginal improvement of every SZ model over the previous model (e.g., 10 parameters improve by about 7% over 8 parameters). GM (“geometric method”) stands for the work in [43, 42], and the other values on the X-axis represent the number of parameters of the SZ model being tested (3 for square, 4 for rectangle, 4+4 for union of two rectangles, 8 for quadrangles etc.).
8.4 Reduction in Local Computation

In addition to communication cost, local power consumption and overall system latency are also affected by the computational complexity of checking the local conditions. The algorithm proposed here enabled a huge improvement over [43],[42] in this regard, since in these works testing the local condition requires constructing a sphere and testing whether all of it is inside $S$, while here one need only test whether the local vector is inside a simple geometric shape. Figure 17 shows an improvement by three orders of magnitude over the geometric method. Clearly this advantage will become even more pronounced in higher dimensions. Moreover, the different types of SZs (hexagons and axis-aligned rectangles) show similar cost, thus justifying the use of free polygons as SZs.

8.5 Clustering

In this set of experiments we evaluated different aspects of the clustering algorithm described in subsection 6.1. Following the discussion in subsection 6.6, we approximated the data distribution of each node using a single Gaussian. This enabled us to use the correlation coefficient between two normal distributions as the distance measure between two nodes. Finally, we
approximated the data distribution of a group of nodes with a GMM.

8.5.1 Flat clustering

The problem of clustering has been widely researched, creating a large number of clustering algorithms. In this section we verified that the clustering method does not significantly affect the algorithm results. We have chosen two algorithms from different sets of clustering classes: agglomerative clustering [37] (from the family of hierarchical clustering algorithms) and K-means [21] (a non-hierarchical clustering algorithm).

We performed one iteration of the iterative clustering algorithm (Figure 18) and compared the performance and the cost of the K-means algorithm and agglomerative clustering for different numbers of clusters. The results show no significant difference between the algorithms in communication (Figure 18a) or running time (Figure 18b). Following these results, we used the K-means algorithm in the rest of the experiments.

This experiment gives us an important insight regarding the desired numbers of clusters. While the optimization time significantly increases with the number of clusters, the improvement in performance becomes negligible beyond a certain point. Therefore, for the dataset at hand, setting the number of clusters \( c \) in the range of \([4..10]\) achieves most of the boost in performance.

8.5.2 Number of iterations

In the following experiment we tested the degree of clustering homogeneity as a stopping criteria for the iterative clustering algorithm. We executed the hierarchical algorithm with \( c = \{2, 3\} \) and depth \( t \) in the range \([1..4]\). As a baseline comparison we show the flat clustering algorithm \( (t = 1) \) with \( c = 16 \).

Figure 19(a) shows that as the value of \( c \) increases \( \{2, 3, 16\} \), fewer iterations (\( \{4, 3, 1\} \) respectively) are required to achieve similar performance. This result is not surprising, as splitting the nodes into more groups results in groups that are more homogeneously clustered and in Szs which are better fitted to the distributions of the nodes in each group.

Figure 19(b) shows the cost of the various clustering algorithms and reveals some important insights. First, the optimization time increases exponentially with \( t \), where \( c \) serves as the base of the exponent. The continuous lines represent the overall optimization time (summation of the optimization time of all the optimization problems). For balanced trees, the number of
Figure 18. Comparing the performance (a) and cost (b) of the K-means algorithm vs. the agglomerative clustering algorithm.

optimization problems is equal to \( c^t \) and each optimization problem takes a similar amount of time as the number of variables is identical.

As shown in this figure, another benefit of the hierarchical clustering algorithm is the speedup gained by parallelizing the execution. Once the first optimization problem is solved, the next optimization problems are independent of each other and can be solved by different machines simultaneously. The dashed lines in figure 19(b) present the running time of such a parallel execution, which corresponds to the hierarchical clustering execution with \( c = \{2, 3\} \). As can be seen, the gain from parallelization is substantial and increases as \( d \) increases.

Finally, figure 19(b) shows an interesting comparison between hierarchical clustering with \( c = 2 \) and flat clustering with \( c = 16 \). With \([c = 2, t = 4]\), the eventual number of different clusters in the system is 16, similar to the scenario with \([c = 16, t = 1]\). Figure 19(a) shows the results are indeed similar. However, figure 19(b) shows the running time of \([c = 2, t = 4]\) is
better (even for the serialized execution). While it might not hold for every dataset, this comparison serves as a strong indication that it might be beneficial to split the difficult optimization problem into many relatively easy problems.

![Hierarchical Clustering - Performance vs. Depth](image1)

(a)

![Hierarchical Clustering - Cost vs. Depth](image2)

(b)

Figure 19. The marginal improvement of deepening the tree decreases (a) while the overall optimization time exponentially increases (b)

8.5.3 Scalability

We next tested the scalability of the dynamic hierarchical clustering version of the algorithm over the simulated data. Figure 20 presents the performance and computational cost required
for large scale systems (from 100 to 5000 nodes). Figure 20 shows how the parameter $c$ influences the results. As expected, using more clusters leads to better results, as shown in figure 20(a). On the other hand, the results of the optimization time are a bit surprising. We expected that using more clusters would necessarily increase the running time but found an abnormal behavior for systems with more than 1000 nodes. It turns out the overhead of the clustering process increases for smaller values of $c$, and naturally this overhead further increases with the number of nodes. Therefore one should keep in mind the cost of the clustering process when deciding on the value of $c$. 
8.5 Clustering

Figure 20. The iterative clustering algorithm supports large scale systems. The algorithm efficiently reduces the number of local violations when choosing the relevant $c$ value (a), and it requires low computational resources even for large number of nodes (b).

8.5.4 Loss of optimality

One might claim that while using clustering does reduce the optimization time, the decrease in performance might be too great. In this experiment we intend to refute this assertion by comparing the performance of optimal SZs (i.e., where each node is assigned a unique SZ) and sub-optimal SZs as produced using the clustering algorithm. For this experiment we used 40 nodes of the AirBase data described in subsection 8.1, so that we were able to solve the complete
optimization problem (without clustering). Figure 21 presents the performance (number of local violations) and the cost (optimization time) as function of $c$. The optimal solution is achieved when each of the nodes is assigned a unique SZ, i.e., when $c = n = 40$. It can be seen that while the optimization time is greatly reduced, the decrease in performance is not substantial. In the worst case, i.e., $c = 2$, performance decreases by 10%, while the optimization is more than 1600 times faster.

Figure 21. The performance of cost as function of $c$. When $c = 50$, each node is assigned a unique SZ; for lower values of $c$, some nodes share similar SZs. In the least accurate case, where $c = 2$, the number of local violations is only 10% higher than in the optimal case. This finding indicates that the loss of accuracy as a result of clustering is not crucial while the improvement in optimization time is considerable.

The explanation for this negligible decrease in performance for small values of $c$ is the distribution of the data. Figure 23 shows the distribution of nodes in the AirBase data according to their size. We measured the “size” of a node by approximating its data distribution with a Gaussian and summing the two eigenvalues of the Gaussian’s covariance matrix. It is evident that the vast majority of the nodes have dense distribution while only a few nodes have sparse distribution. The clustering algorithm groups all the small nodes together and assign unique SZs only to the big nodes. Increasing $c$ in this setup will lead to a better separation between the big nodes, but the improvement in the performance is small because these few nodes were responsible only for a small portion of the violations in the system. For higher values of $c$ the clustering algorithm separates between small nodes which contributes nothing to the performance.
8.5 Clustering

Figure 22. Histogram of the nodes from the AirBase data according to their size. Each node is approximated with a single Gaussian, and its size is the sum of the eigenvalues of that Gaussian.

Figure 23. The SZs for $c = 2$ (a) and for $c = 4$ (b) on synthetic data where each node belongs to one of four orientation classes.
8.6 Probabilistic Approach

In this subsection we evaluate two important aspects of the probabilistic approach (Section 7). We used synthetic data in these experiments in order to isolate the effect of the investigated parameters.

The data distribution was identical in all the nodes and distributed as a bivariate normal distribution with the following parameters: \( N(\mu = \begin{bmatrix} 0 & 0 \end{bmatrix}, \Sigma = \begin{bmatrix} 50 & 0 \\ 0 & 50 \end{bmatrix}) \). The threshold domain \( S \) was the unit circle.

Following the results of the experiments on the AirBase data we conclude that for typical data where most of the nodes are of similar size with few exceptional large nodes the loss in performance caused by the clustering algorithm is not meaningful. In the following experiment we show a scenario where \( c \) has a great impact on the performance. We used synthetic data in which each node belongs to one of four groups, as shown in Figure 23.

Figure 24 shows the performance and cost of the clustering algorithm in this experiment. We can see increasing \( c \) from 2 to 3 and 4 has great impact on the performance, while further increase makes no difference.

The reason \( c \) plays such a significant role in the last experiment, as oppose to the rest of our experiments, is that in this experiment the improvement in SZs is beneficial to all the nodes while in previous experiments only the big nodes benefited from the improved SZ.

Figure 24. The performance and cost of the clustering algorithm on nodes with different orientation.
In the first experiment we investigated the effect of $\epsilon$, the error probability (or the allowed rate of false negatives), on the algorithm’s performance. We extended the domain $S'$ by enlarging its radius and measured both the error rate and the number of messages for various radii. Figure 25 shows the number of local violations (i.e., the number of messages sent from the nodes to the coordinator) against the error rate in the system. It is evident from the graph that allowing some false alarms can considerably reduce communication. Moreover, most of the reduction is achieved within a small probability of error. About 75% of the communication is saved with an error probability of just 0.1%.

![Figure 25. Reduction in communication when using the probabilistic algorithm with error probability of up to 14%](image)

The next experiment investigates the effect $n$, the number of nodes in the system, on the probabilistic approach.

![Figure 26. Number of alerts as a function of number of nodes](image)
Figure 26 presents the number of messages sent from the nodes to the coordinator with respect to the number of nodes in the system. We compare the deterministic algorithm with the probabilistic implementation with $\epsilon = 1\%$. It can be seen that while the number of messages increases in the deterministic algorithm, the opposite trend is reflected in the probabilistic approach. Thus, increasing the number of nodes further emphasizes the benefits of the probabilistic approach. Since our algorithm is designed for large-scale systems with thousands of nodes, this property is very desirable.
9 Future Work

Our work is merely an initial attempt to present optimal local constraints as a solution to the task of distributed monitoring. While we started to investigate various extensions such as the probabilistic approach and clustering, further extensions in various directions are possible.

The probabilistic approach could also be further extended and be formalized. We used a basic heuristic of extending the threshold and measuring the false-negative rate post factum. Further research is required in order to enable the extraction of an extended threshold from a desired limit on the false-negative rate.

Greater effort can also be placed on clustering. We primarily investigated the clustering of bivariate normal distributions. While this distribution is common, other, more general types of distributions should be investigated as well.

Three additional directions are handling more dimensions, monitoring more complex functions and designing an algorithm to efficiently recover from violations. The following subsections outline these directions.

9.1 Monitoring Higher Dimensions

Our algorithm is applicable to higher dimensions, but a naïve adaptation would suffer from a drastic increase in the cost of solving the optimization problem. We started looking for methods to tackle this problem but we made only the first steps and there is room for more work in order to look for more methods and to evaluate their performance. In the following passage we briefly describe some developments in our ongoing research.

Let $N, D$ be the number of nodes and dimensions respectively and assume our SZs are convex polytopes with $m$ vertices. The number of free variables in the optimization problem is $NDm$. We can ensure the Minkowski average of the SZs is contained in $G$ by calculating the average of every possible combination of the SZs vertices and checking whether it resides in $G$ or not - which sum up to $Dm^k$ constraints. This method is extravagant as most of the averages fall inside the convex hull of Minkowski average rather than on its facets.

We can greatly reduce the number of constraints if we define $G$ as the intersection of $f$ half hyperplanes, where $f$ is the number of $G$'s facets. By testing the Minkowski average of the SZs against one half-hyperplane at a time we can be satisfied with just $fDmN$ constraints. This is a significant improvement, but one must keep in mind that the number of facets of $D$-dimensional
polytope with $m$ vertices is bounded by $f \leq m \left\lfloor \frac{D}{2} \right\rfloor$, so the curse of dimensionality is relieved but not resolved.

Calculating the objective function becomes expensive as well in higher dimensions as Green’s theorem can no longer be applied. A reasonable solution is to use discrete integration, in which we use a training set $S$ of $|S|$ observations for constructing the SZs. The objective function will be equal to the number of training observations that reside inside the SZ, i.e., $\sum_{s \in S} I(s \in SZ)$, where $I(s \in SZ) = \begin{cases} 1 & s \in SZ \\ 0 & s \notin SZ \end{cases}$. In practice we found it is desirable to have a continuous objective function, therefore we use $\sum_{s \in S} \min(1, e^{-d})$, where $d$ is the distance of $s$ from the SZ. This way slightly adjustments to SZs will not cause the objective function to jump discontinuously. The time complexity of calculating this objective function is $\Theta(|S|D_f)$.

If we use union of axes aligned hyperrectangle SZs rather than polytopes we can further decrease the cost of calculation the constraints and the objective function. The number of constraints will remain the same but the constants are smaller. More importantly, the shift from convex polytopes to union of hyperrectangles makes it possible to define non convex SZs, which in turn allows us to monitor a non convex $G$.

Another issue we need to consider when moving to higher dimensions is the cost of the clustering algorithm. The problem of calculating the distance between each pair of nodes becomes more crucial as the number of dimensions increases, because more training observations are required in order to reliably describe the data distribution.

9.2 Recovery From Violations

The event in which a node wanders outside its SZ (violation) necessitates communication, the amount of which depends on the extent to which the violation can be resolved locally, in collaboration with as few nodes as possible. If $k$ nodes take part in handling the violation, assume w.l.o.g. that these are nodes 1 to $k$ having SZs $S_1...S_k$, and that the violation occurred at node 1. Then a local procedure for resolving the violation will optimize not over all nodes, but only over 1 to $k$. The result of the optimization will assign nodes 1 to $k$ new SZs $S'_1...S'_k$ so that:

- a) $x_1 \in S'_1$
- b) $S'_1 \oplus ... \oplus S'_k \subset S_1 \oplus ... \oplus S_k$ (this implies that the global Minkowski average is still inside
9.3 Monitoring Other Types of Functions

So far we have discussed functions defined by $f\left(\frac{x_1 + \ldots + x_n}{n}\right)$, where $x_i$ are the local vectors. The idea can easily be extended to functions defined by $f(\alpha_1 x_1 + \ldots + \alpha_n x_n)$, for any scalars $\alpha_i$ (positive or negative). If a function is equal to the sum of a few such functions, one can define a threshold on it by choosing appropriate thresholds on these functions. For example, the inner product $(x_1, x_2)$, which is an important measure of correlation, cannot be expressed as $f\left(\frac{x_1 + x_2}{2}\right)$. However it equals $\frac{\|x_1 + x_2\|^2 - \|x_1 - x_2\|^2}{4}$, that is, the difference of two functions which can be monitored by the method described here. Therefore, the inner product can be thresholded by thresholding $\|x_1 + x_2\|^2$ (from below) and $\|x_1 - x_2\|^2$ (from above). It would be interesting to find how different types of functions can be expressed as a combination of $f\left(\frac{\alpha_1 x_1 + \ldots + \alpha_n x_n}{n}\right)$, so they can be monitored with our algorithm.
10 Conclusions

This work proposes a very general approach to monitoring arbitrary functions (not necessarily linear or monotonic) over dynamic, aggregated data distributed between nodes. In the deterministic version, an alert is sent when the function's value crosses a certain threshold. In the general case, it is impossible to infer anything about the value of the function on the aggregated data, given its value on the individual nodes; one has to track the domain over which the function is defined. Viewing this domain in purely geometric terms, we define "safe zones" (SZs) at each node, such that as long as the local data tuples remain in them, the value of the function at the global data vector will not cross a given threshold. A rigorous optimality criterion is provided for computing the SZs, which considers the distribution of the data at the nodes and can handle even radically different distributions in distinct nodes. A few parametric families of SZs are considered, and some lower bounds are provided on their geometric distance from the optimal SZs. Two extensions are studied as well. By clustering similar nodes, we achieve scalability and allow monitoring thousands of nodes within reasonable cost. By applying the probabilistic approach, we improve performance significantly while rarely producing false alarms. We evaluated our algorithm empirically and tested the effect of various parameters. Our solution improved the performance of previous work by orders of magnitude and reduced the computational overhead of testing the local conditions at the nodes by more than three orders of magnitude.
References


REFERENCES


54


REFERENCES


The algorithm we propose requires dealing with systems having thousands of sensors and even more.

We achieve this goal by clustering sensors according to their call distribution. We base on the observation that sensors with similar distribution will be allocated similar local conditions and will significantly improve the algorithm's capability to deal with large numbers of nodes.

In this work, we include a description of several experiments we conducted to examine the system's performance and the effects of various parameters.

The experiments conducted involved both synthetic data and data collected from sensors distributed throughout Europe and monitoring the emission of various pollutants.

The structure of the work is as follows:

Part 2: We review various works in the field.

Part 3: We provide a formal description of the problem and present the theoretical foundations of the work.

Part 4: We present the algorithm we developed, and Part 5 covers aspects of the system and the algorithm.

Two important extensions to the basic algorithm, sensor clustering and the probabilistic approach, are described in Parts 6 and 7, respectively.

Part 8: We describe the results of the experiments we conducted.

Finally, Part 9: We discuss various directions in which the research work can be expanded, and Part 0: We summarize our conclusions.
The task is to minimize the cost of checking the functions at the global field and the values of each sensor. One simple solution is to require each sensor to send its samples to a central processor, which calculates the global vector and the global function value. However, this solution is problematic because the communication cost is very high.

One of the goals of our research is to propose an algorithm that reduces the communication cost as much as possible. This problem can be addressed using a geometric approach that defines an optimal criterion for solving the problem. The algorithm we propose is an optimal for each sensor, as long as the value it measures is within the sensor's field. If the sample is outside the sensor's boundaries, the sensor is not required to send it.

The geometric approach proposed in previous works suffered from several limitations. First, the algorithm used local heuristics, while we define an optimal criterion and rely on it to define local heuristics. Second, the previous works did not exploit the distribution of the samples in the sensors to propose local heuristics that are better. The algorithm we propose is based on the distribution of the samples at the boundaries and defines optimized local heuristics.

Finally, the local heuristics offered in previous works were very expensive to calculate and not intuitive for human observers. It is crucial to ensure that the local heuristics are simple and require no calculation. Moreover, in many systems, the sensors are based on a battery with a finite lifetime, which limits the operations they can perform.

In our algorithm, the local heuristics are the simplest and have no calculation cost. Additionally, the algorithm we propose allows the local heuristics to be described in a simple and intuitive way for human observers.

The geometric approach was proposed in previous works but suffered from several limitations. First, the algorithm used local heuristics, while we define an optimal criterion and rely on it to define local heuristics. Second, the previous works did not exploit the distribution of the samples in the sensors to propose local heuristics that are better. The algorithm we propose is based on the distribution of the samples at the boundaries and defines optimized local heuristics.

Finally, the local heuristics offered in previous works were very expensive to calculate and not intuitive for human observers. It is crucial to ensure that the local heuristics are simple and require no calculation. Moreover, in many systems, the sensors are based on a battery with a finite lifetime, which limits the operations they can perform.

In our algorithm, the local heuristics are the simplest and have no calculation cost. Additionally, the algorithm we propose allows the local heuristics to be described in a simple and intuitive way for human observers.
תקציר

אלגוריתמי כריית מידע הם כלים מרכזיים ביישומים רבים. דוגמה לאלגוריתמים אלה היא ניטור זרמים מידע בזירים. כך, ניתן להשתמש באלגוריתמים כדיリンזיות מזג אוויריות או מערכות חיפוש ב🇮🇱. בבית לחם, יש להתריע על תופעות שונות כגון לחץ המים והטמפרטורה. כאשר מתאימים, הערכה של נתונים מתאימים לערכים שונים, ניתן להשתמש באלגוריתמים כדיリンзовית זרמי מידע.

עד ל.postValueות המודרניים, המודדים בתוכניות בזירות זרמי מידע, ניתן להגדיר תוצאות מקומיות באמצעות פעולות אגרגציה כגון ממוצע, סכום או מינימום. בעריכת פונקציות פוטנציאליות, ניתן למדוד את התוצאה של פונקציה גלובלית. לדוגמה, תוך מתן התמדה בשתי מערכות, אחד משתי מערכות התדות לריפוע של שאילתות מופ/member התנהר של 1% ריפוע, ודבר שתרות התופשות.

הסתברות של שירת בודדתtol המועבר סך זה.

ב�ומני בריכים הפונקציה imposition at נתןضغוד של מרחבי הלוך והשוב, כש yal של פונקציה גלובלית. בדוגמהvasion,мя נטילון תמרון משולש עד מפגש בין שניים, שיטת הקירור ב giữa המרובה במודולאסר, ויבוש של EFFECTS. לדוגמה, נטילון תמרון משולש עד מפגש בין שניים, שיטת הקירור ב giữa המרובה במודולאסר, ויבוש של EFFECTS.

השימוש באלגוריתמים&mמך בתנהל של רכיבים,ント ריכים, ויבוש של EFFECTS. לדוגמה, נטילון תמרון משולש עד מפגש בין שניים, שיטת הקירור ב giữa המרובה במודולאסר, ויבוש של EFFECTS.

לשם התנהל השיפור של השיפור המופך במודולאסר, ויבוש של EFFECTS. לדוגמה, נטילון תמרון משולש עד מפגש בין שניים, שיטת הקירור ב בין המרובה במודולאסר, ויבוש של EFFECTS.
הباحث מתאר את הבנתו של פרופ' אסף שווסטר ופרופ' דניאל קרן במפעל המחבר.
כרכר בפקולטה למדעי המחשב.

הכרת תודה
אני מודה לפרופ' אסף שווסטר, פרופ' דניאל קרן וגיא שגיא על תמיכתם והנחייתם.
אני מודה לטכניון על התמיכה הנדרשת.

אני מודה לטכניון על התמיכה המקצועית והנהלית.
בעשתלמותי.
ניסוח ורמי מידע מבוזרים

חיבור על מחקר
לשם מילוי חלקי של הדרישות לקבלת התואר
מגיסטר למדעי컴ью במדעי המחשב

אבישי ליבנה

הוגש לסנט הטכניון – מכון טכנולוגי לישראל
וני 2010

ıyor ט"ש

Technion - Computer Science Department - M.Sc. Thesis  MSC-2010-17 - 2010
גיוסור זרמי מידע מבוזרים

אביישי ליבנה