Noname manuscript No.
(will be inserted by the editor)

Data Fusion in Automotive Applications

Efficient Big Data Stream Computing Approach

Amir HAROUN · Ahmed MOSTEFAOUI · François DESSABLES

Received: date / Accepted: date

Abstract Connected vehicles are capable of collecting, through their embedded sensors, and transmitting huge amounts of data at very high frequencies. Leveraging this data can be valuable for many entities: automobile manufacturer, vehicles owners, third parties, etc. Indeed, this "big data" can be used in a large broad of services ranging from road safety services to after-market services (e.g., predictive and preventive maintenance). Nevertheless, processing and storing big data raised new scientific and technological challenges that traditional approaches cannot handle efficiently. In this paper, we address the issue of online (i.e., near real-time) data processing of automotive information. More precisely, we focus on the performance of data fusion to support several millions of connected vehicles. In order to face this performance challenge, we propose novel approaches, based on spatial indexation, to speed-up our automotive application. To validate the effectiveness of our proposal, we have implemented and conducted real experiments on PSA-Group[1] big data streaming platform. The experimental results have demonstrated the efficiency of our spatial indexing and querying techniques.

Keywords Big Data · Connected Vehicles · Stream Computing · Data Fusion

Amir HAROUN
PSA GROUP
Bessoncourt, France
E-mail: amir.haroun@mpsa.com

Ahmed MOSTEFAOUI
FEMTO-ST Institute/CNRS
Bourgognes-Franche-Comte University
Belfort, France
E-mail: ahmed.mostefaoui@univ-fcomte.fr

François DESSABLES
PSA GROUP
Bessoncourt, France
E-mail: francois.dessables@mpsa.com

[1] PSA Group is the second-largest automobile manufacturer in Europe with about 3 million sold vehicles in 2015.

1 Introduction

During the last few years, we have witnessed an great advance in communication technologies that led to the emergence of new concepts such as Connected Vehicles (CVs). By 2018, all new European cars must be equipped with communication capabilities to enhance the driver's safety[2]. Based on its *On Board Unit* (OBU), each connected vehicle can either communicate with other vehicles (Vehicle-To-Vehicle (V2V) communications) or with an infrastructure (Vehicle-To-Infrastructure (V2I) and Infrastructure-To-Vehicle (I2V) communications). Each vehicle is able to collect up to 170 information through several sensors deployed in all its components (i.e., engine, interior, exterior, etc.). The transmitted data, represented as frames, contains a wide range of information ranging from the Vehicle Identification Number (VIN), to the current GPS coordinates passing by the engine Rounds-Per-Minute (RPM), the vehicle speed, the external temperature, etc. This captured data could be of a paramount interest for several real applications as road safety, eco-driving, traffic regulation, environment monitoring, etc. Figure 1 illustrates the transmission flow from vehicles to end-users (e.g, vehicles owners, third-party partners, etc).

In some V2I applications (e.g., extensive Controller Area Network (CAN) monitoring), we are interested in

[2] `{http://sites.ieee.org/connected-vehicles/2015/04/28/ecall-in-all-new-cars-from-april-2018/}`

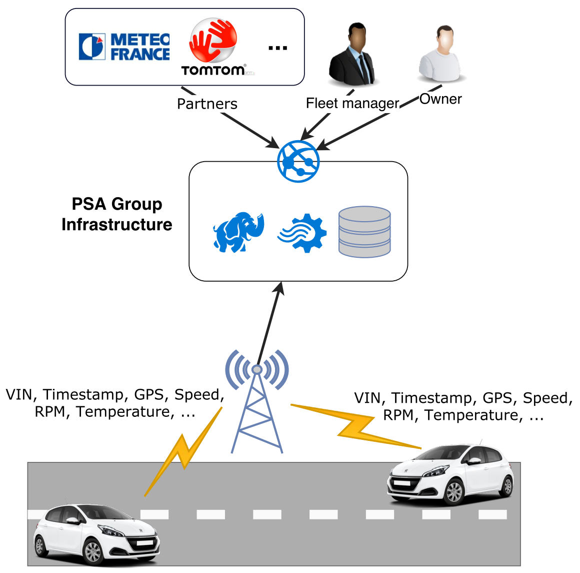

METEO FRANCE
TOMTOM
...
Partners
Fleet manager
Owner
PSA Group Infrastructure
VIN, Timestamp, GPS, Speed, RPM, Temperature, ...
VIN, Timestamp, GPS, Speed, RPM, Temperature, ...

Fig. 1 PSA Group's services

transmitting all the data captured from the CAN bus[3]. Connected vehicles are however able to generate a huge amount of data at a very high frequency. For instance, if we assume a car delivery of $1Mb$ per second per CV, the computing infrastructure has to support a workload of $5*10^6$ Mbit/sec[4]. This gives an amount of 5 Tbps (or 300 Tb per minute!), which has eventually to be stored for off-line analysis.

Processing and analyzing this big data represent a challenging research issue. Indeed, in automotive big data applications, we have to deal with the challenging *triple V*: Volume, Velocity, Variety [18] of big data.

– Big Volume refers to data size that continues to increase exponentially. As we saw, when it comes to millions of CVs, the volume of data will rapidly explode. This issue makes the traditional storage solutions such as Centralized File Systems (CFS) and Database Management Systems (DBMS) inefficient in the face of this big volumes of data.
– Big Velocity stands for the speed of data generation and processing. Data is generated at high rates that makes processing techniques slower compared to the generation rates. However, some automotive applications require hard constraints of processing (i.e. (near) real-time response). The challenge of velocity is to design processing solutions that handles well the historical data as well as online data.
– Big Variety is related to the heterogeneity data types. In fact, CVs send various types of data such as time-based signals, scalar data, images, videos etc. This challenge must be handled for automotive applications to exploit the full potential of CVs.

However, big automotive data challenges include other factors: veracity and value. Data veracity concerns the provenance and the traceability of data and plays an important role in automotive applications in order to guarantee safety and security in connected services. Finally, value of big automotive data has a mean for a broad range of automotive services like aftermarket services (e.g., predictive and preventive maintenance), entertainment services (e.g., movies), etc. Another important issue which has also to be taken into account in big automotive data applications is the *scalability* of the infrastructure. This issue concerns the ability of the infrastructure to scale up with the heavy workload which it is supposed to support.

We illustrate these issues through the following real application, implemented in PSA GROUP's big data infrastructure, about the environmental monitoring. In fact, many private as well as public companies are interested to use CVs as *distributed mobiles sensors* to monitor the environment and getting up-to-date views of the monitored area. An example of such an application is the whole country temperature view, requested by the French National Meteorological Service [1] for their meteorological previsions. The objective is to get an updated map of the country temperature, including regions and departments, as precise as possible (see Figure 2). Hence, each CV sends the captured exterior temperature, tagged with its current location to the infrastructure. The latter is then responsible of correlating the received data in order to continuously updating the map. In this application, we focus on temperature, but any other data could be monitored; e.g., pollution, fog, rain, frost, etc.

This application has been implemented using the stream processing engine IBM STREAM [14]. The latter is based on the *Stream Computing* paradigm which exploits various kinds of parallelism (i.e., task, data and pipeline). Stream computing paradigm has been used to efficiently process real time unbounded data streams [9] as it is the case of our temperature application (see Section 3 for a detailed review).

The results of the real experiments we have conducted have however pointed out a poor performance of the application. In fact, the infrastructure was not able to handle in real-time more than 1500 simultaneous CV data streams, whereas its hardware platform is able

[3] The ISO 11898 standard specifies that the CAN physical layer allows transmission rates up to 1 Mbit/s for use within road vehicles. See `http://www.iso.org/iso/catalogue_detail.htm?csnumber=33423`
[4] PSA GROUP is planning for 2020 to handle data from nearly 5 millions cars around the country (e.g., France).

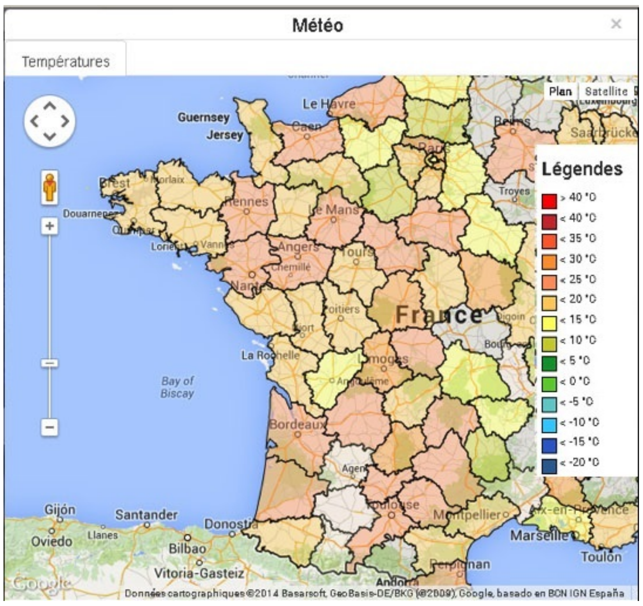


Fig. 2 "Temperature" use case.

to largely support this workload. Obviously, the poor performance comes from the software. As in Stream computing the processing chain is composed of various operators, organized in directed graphs, the overall processing time is then defined by the slowest processing operator (or component). We have then conducted a deep performance analysis of all components of our application in order to identify the bottleneck responsible of this poor performance. Our analysis revealed that the sub-routine IS-CONTAINED() was the slowest operator when the workload is heavy. We used this sub-routine, provided in IBM STREAM, GEOSPATIAL TOOLKIT, in order to correlate the temperature captured by a CV with the data on the map. More precisely, we used IS-CONTAINED() to locate a point, representing the temperature captured by a CV, within a polygon, representing the corresponding region (e.g., a town). We have then measured the performance of this sub-routine by varying the complexity of the region (i.e., number of edges in the polygon). We mainly measured the required time to determine to which polygon a point belongs. The results are plotted in Figure 3.

As shown in the figure, more complex the polygon, higher the processing time. Even though the processing time of one operation is very short (of order of 5 microseconds), the system is nevertheless not able to handle in real-time all the expected CVs data streams (of the order of millions) associated with a complex input map; i.e., polygons of regions and departments.

The research issue we are facing is how to speed up this operation in order to meet the real-time target application requirements. More formally, the problem

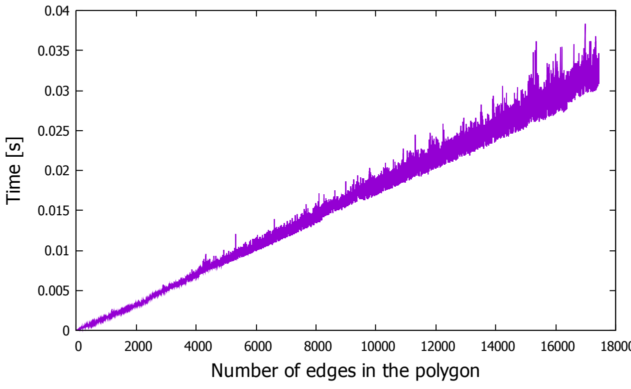


Fig. 3 Performance of sub-routine IS-CONTAINED() with regard to the complexity of the polygon (number of its edges)

we are facing is the general one called: *Point Location Problem (PLP)*, where a point has to be located inside or outside a given polygon.

In this paper, we study this issue and propose new approaches, based on the use of novel spatial indexing structures, that overcome PLP performance within stream computing. To the best of our knowledge, this is the first study addressing query performance of PLP within stream computing middleware.

The rest of the paper is organized as the following: In Section 2, we review the PLP related work. Section 3 introduces the concepts behind stream computing. In Section 4, we present the developed approaches to speed-up the target application. Section 5 highlights the experiments we conducted and the obtained results. Finally, Section 6 concludes this paper and highlights our future works.

2 Related work

This section presents the Point Location Problem (PLP) and some state-of-the-art studies that have been used to overcome it. Given a partition of the space divided into disjoint regions, PLP aims to determine efficiently the region where a query point lays (refer to Figure 4 for an illustration). This problem is present in different fields: computer vision, image processing, computer graphics, motion planning and geographic information systems (GIS).

Input description

– A planar subdivision formed by a set of disjoint regions R= $\{R_1, R_2, \ldots, R_n\}$, each region is defined by its bounding polygon P= $\{P_1, P_2, \ldots, P_n\}$, a polygon has m non intersecting line segments (or m vertices).

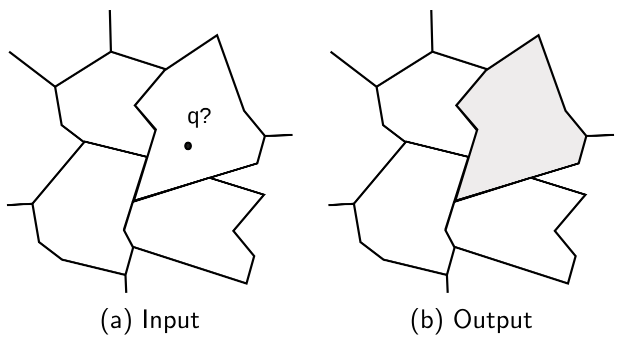


Fig. 4 Point Location Problem

– A query point q defined by its two dimensional coordinates.

Problem

Which polygon Pi contains the query point q?

2.1 Point In Polygon

Whereas PLP is the method that returns the containing polygon among a set of polygons, Point In Polygon (PIP) is used to check if a point is inside or outside a given polygon. One of the earliest PIP methods is the Crossing Test [26], we draw a ray starting from the query point and going to any direction then we count how many times this ray traverses the polygon's edges. If this number is odd then the point is inside the polygon, otherwise the point is outside. Unambiguously, the worst case computational complexity of this method is linear in the number of edges.
An improvement of this method was proposed by MacMartin [11], he filters the set of edges saving some calculations. Other methods are proposed to improve this algorithm (e.g., barycentric, angle summation test, triangle fan, etc) but the MacMartin method stays in the lead. Table 1 presents a comparison between the most known PIP methods, a full description of these methods can be found in [11].

Table 1 Comparison of general algorithms using random polygons [11]

Number of edges	3	4	10	100	1000
MacMartin	2.9	3.2	5.9	50.6	485
Crossings	3.1	3.4	6.8	60.0	624
edge sort	1.1	1.8	6.5	77.6	787
Triangle Fan	1.2	2.1	7.3	85.4	865
Barycentric	2.1	3.8	13.8	160.7	1665
Angle Summation	56.2	70.4	153.6	1403.8	14693

2.2 PLP approaches

In this part, we describe some state of the art approaches to overcome the PLP.

2.2.1 Slab decomposition

The slab decomposition of Dobkin and Lipton [20] is one of the earliest approaches to solve the PLP. A vertical line is drawn through each vertex, this divides the plane into at most $n + 1$ vertical slabs, each slab is defined by the region between two consecutive vertical lines (where n is the number of vertices) (see Figure 5). Let E_s be the set of all edges in the graph that cross a slab s. Note that the set E_s partitions the slab s into a set of regions (trapezoids and triangles).

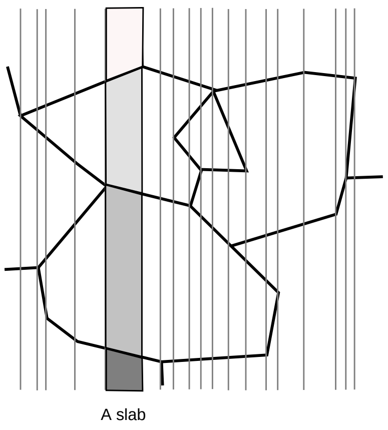


Fig. 5 Slab decomposition

The PLP is reduced into two simpler problems:

– which slab contains the query point.
– which region of the slab contains the query point.

Both of these two sub-problems can be answered by a binary search. First, a binary search on the x coordinate of the vertical line is performed in order to finding the slab that contains the query point. Then, a second binary search is done on the y coordinates of the edges crossing the slab to find the region of the slab containing the query point.

2.2.2 Triangulation refinement

The basic idea of triangulation refinement [17] is to build a tree hierarchy (i.e. directed acyclic graph) of triangles. As its name suggests, each level of the hierarchy must be a triangulation, it's however not always the case. So the first step is to transform the initial

set of polygons into a set of triangles, there are several algorithms to do this [7, 25]. The data structure is then built starting from the triangulated subdivision S_1, we remove some vertices to simplify the data structure then we re-triangulate the subdivision to get a less complex level S_2 and iterate until obtaining the root triangle S_h.
The query is performed iteratively starting from the triangle of the level S_h that contains the query point. In each iteration, we try to find triangle of the level $Si - 1$ that contains the query point. Each triangle of the level S_i has pointers for the triangles it intersects of the level S_{i-1}. We iterate until reaching a leaf from the level S_1. Point location method is described in Algorithm 1

Algorithm 1: Triangulation

```
   Input: A sequence of triangulations: $S_1, S_2, \ldots, S_h$
          organized as a tree hierarchy,
          A query point $q$
1  if $q$ is not included in the root triangle $S_h$ then
2      return null
3  else
4      current ← root;
5      while current has descendants do
6          for $i$ ← each descendant of current do
7              if $q$ is included in descendent $i$ then
8                  current ← $i$;
9                  break;
10 return current
```

2.2.3 Trapezoidal decomposition

The trapezoidal decomposition [29] is obtained by drawing a vertical line from each vertex that stops when hitting an edge (see Figure 6). This way, we obtain only a subset of the slabs compared to the slabs decomposition. The PLP query is however answered by building a trapezoidal hierarchy resembling that of the former method (i.e. Triangulation refinement). Starting from the bounding box as a root rectangle, we add edges one at a time from the trapezoidal decomposition to create a trapezoidal data structure.

The methods described in this section have not been used directly in our implementations due to fact that they are exact methods that keep the precision of polygons and hence they do complex operations. In automotive applications and more precisely in stream processing, computations must however be very quick (i.e., near real-time). Moreover, in real applications, the desired precision variates, some applications require more or less precision than others. Nevertheless, the inferred

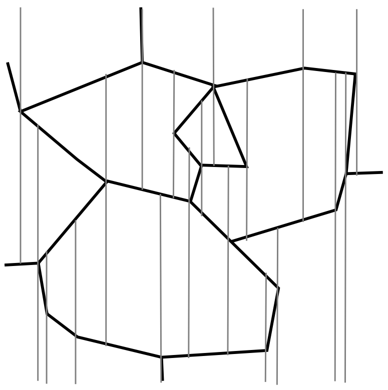


Fig. 6 Trapezoidal decomposition

knowledge has been considered in the proposition of adapted approaches for automotive context.

3 Stream Computing Paradigm

Big data applications pose new challenges that traditional processing and storage solutions are not able to sustain [16], especially in term of scalability [12]. Many solutions have been studied and proposed to respond to big data challenges, among them Hadoop framework [3]. In fact, due to its distributed nature, Hadoop fits well to some big data requirements, but not to all. In particular, because its processing component (i.e. MapReduce) operates in batch mode [30], Hadoop is not able to respond efficiently to the automotive requirements (i.e., online processing, low latency queries, etc).

Stream processing[5] has been introduced as a new programming model for efficient and parallel processing of continuous data streams [13], with hard constraints (i.e., real-time or near real-time computing).
To understand the concept of stream processing and distinguish it from the classical form of processing, here are the main stream computing's characteristics:

- Each and every data item is processed as it arrives (i.e. online).
- Events are time-based, every record is typically timestamped on creation.
- Operations are done in a data flow design (see Figure 7).
- Every operation is done on one data element (or a small window of recent data).
- An operation computes something relatively simple.
- Each computation needs to complete in (near) realtime to avoid congestions.

[5] The terms processing and computing are interchangeable in the rest of the paper.

Streaming applications can be modeled through a Directed Graph (DG) in which vertices are *streams* (i.e., conceptually an infinite sequence of data items) and edges are *operators* (see Figure 7).

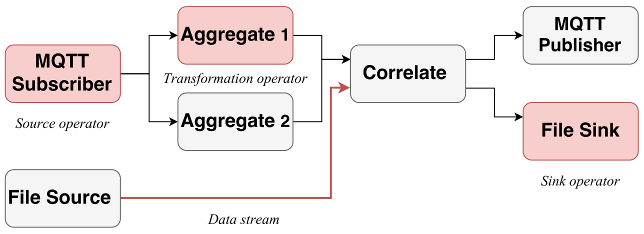


Fig. 7 A streaming application

There are three categories of operators:

- *Source operators:* sources of the input data streams. They are connected to an external source of data so they transform the external data into data streams to send them via their output ports. Example: operators reading from a message queue, a broker, a data base, etc.
- *Transformation operators:* carrying out the computation, they transforms input data streams to output data streams. Example: aggregation, transformation, correlation, etc.
- *Sink operators:* usually placed at the end of the streaming application. Example: operators writing to a message queue, a file system, etc.

In order to process unbounded data streams efficiently, stream computing uses various kinds of parallelisms (i.e. task, data, and pipeline parallelism) [9].

Streaming applications are published as *jobs* on a *Stream Runtime Environment* (SRE) also called instance. SREs can be deployed on one or multiple hosts. Hence, the stream computing scales horizontally by adding more nodes to the SRE. A job is an application instance running on a SRE, it executes a set of Processing Elements (PEs)[6]. In stream applications, the overall processing time is defined by the slowest processing part. In other words, slowing a component in the processing graph involves slowing down the whole application.

4 Proposed approaches

The ineffectiveness of the first implemented approach, called *Exhaustive approach*, is due primarily to the fact that all the polygons are tested until the correct polygon is found (hence the name). Then, the corresponding temperature is added to the target region. This approach has been shown to be ineffective since it is not able to support the (near) real-time processing requirement (see its experimental evaluation in Section 5).

Two main facts contribute in the poor performance issues of exhaustive approach:

(a) The high complexity of the target polygons as confirmed in Figure 3. Indeed, more complex the polygons, higher the processing time.
(b) The large set of polygons to test. In fact, all the polygons are candidates and are hence tested, which consumes more processing time.

In this section, we present three approaches that alleviate the weakness of the exhaustive approach.

4.1 Two Steps Approach

In this approach, in order to improve the exhaustive method, we focus on reducing the large number of polygons. In order to reduce the set of tested polygon, we studied the approach called two steps approach [22]. The basic idea behind this approach is to filter the initial set of polygons, keeping only few potential candidates. This filtering is performed on *approximations* of the input polygons in order to speed-up the process. Once a smaller subset of polygons has been identified, in a second step, only retained polygons are tested in depth. Figure 8 illustrates the two steps approach.

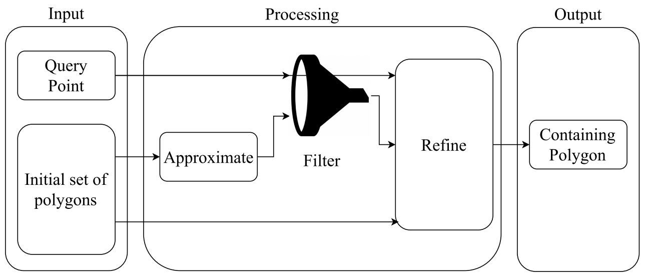


Fig. 8 Approximation-based query processing.

As shown in the description of the approach, it operates in two steps:

- **Filtering**: This step is based on a simple fact, if a query point q is not contained in a simpler form A_i that covers a polygon P_i, then it will be never contained in the polygon P_i. By doing so, a large subset of unnecessary candidates can be avoided and hence saving processing time. The filtering must be quick

[6] A processing element is a thread executing executes a set of operators instances.

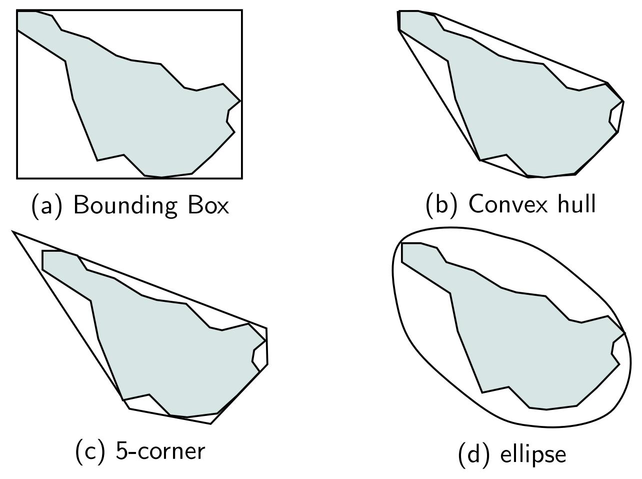


Fig. 9 Various conservative approximations.

and efficient to maintain good overall performance. This first step aims to simplify and approximate the initial shape of the polygon to get a less complex objects with fewer verticies. The fundamental condition of the simplification is that its result should be simple and contain completely the initial shape. Several approximation techniques (illustrated in Figure 9) exist in the literature (e.g. minimal bounding box[23], minimal bounding circle, convex hull[24], 4-corner, 5-corner, ellipse, etc.). A comparison of these methods could be found in [5]. The quality of an approximation is defined by the ratio of the area of the initial spatial object to the area of the approximation. The closer the ratio to 1, the better the quality of approximation. The latter is very important because it will define the quality of filtering.
A first exhaustive search (i.e. filtering) is done on the polygon's approximations then the full test on the original polygon is performed. If the query point is not contained in the polygon's approximation, there is no need to test on the full polygon. (refer to Section 5 for the practical results).

– **Refinement**: In this second step, only a reduced subset of polygons is selected for the refinement, which consists in tests on the candidates original polygons.

Algorithm 2 presents the two steps approach.

Algorithm 2: The two steps method

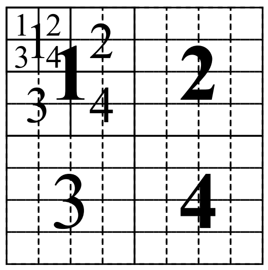
```
Input: A finite set P= {$P_1, P_2, \dots, P_n$} of polygons,
       A query point $q$
Output: The polygon that contains the query point $q$
/* Offline computation,                               */
1  for $i \leftarrow 1$ to $n$ do
2      $A_i \leftarrow$ approximate($P_i$);
/* Online Computations                                */
/* Filtering                                          */
3  $Candidates \leftarrow \emptyset$
4  for $j \leftarrow 1$ to $n$ do
5      if PIP($A_j, q$) then
6          $Candidates \leftarrow Candidates \cup j$
/* Refinement                                         */
7  for $k \leftarrow 1$ to size($Candidates$) do
8      if PIP($P_{Candidates[k]}$,$q$) then
9          return $P_{Candidates[k]}$
10 return null
```

4.2 Decomposition Approach

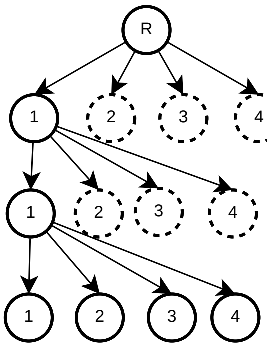
Whereas in the two step approach we focus on reducing the large number of candidates, in this approach, we treat the problem of complexity of polygons. The key idea behind this approach, more precisely a family of approaches, is to reduce the complexity of the considered polygons by decomposing them to small and less complex objects. In other words, constructing a spatial indexing structure on which the considered polygons are mapped. In a second step, this spatial indexing structure is used to decide which polygon contains the query point. Of course, the objective is to speed-up this operation. To this end, the space is first divided to produce a structure (e.g. Grid [27], R-Tree [10], X-Tree [4], Quadtrees [6]). Within this paper, we used a Quadtree structure [28] because of its relative implementation simplicity and its coherence with spatial indexation. The Quadtree is a tree data structure where every node has exactly four children. To create a spatial Quadtree, we start with the entire space as a single partition, then it is recursively divided into four equal cells until a stop condition is validated. The stop condition, which depends of the required target application precision, can lead to the creation of balanced Quadtree or unbalanced Quadtree.

4.2.1 Balanced Quadtree

A balanced Quadtree is a tree where all the leaves are at the same level. Hence, all cells have the same area size. Obviously, lower the cell's size, better the precision. So, cells can well capture the mapped polygons (see Example of Figure 10), but at the expense of a large indexing Quadtree. Oppositely, largest cells lead to a reduced indexing Quadtree, which however can lack precision; i.e., one cell can contain several polygons. Consequently, there is a need for a trade-off between the size of the indexing structure and the processing time. Whilst balanced Quadtree presents the advantage to be simple to



(a) Balanced decomposition

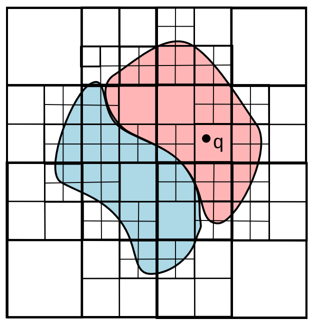


(b) Quadtree

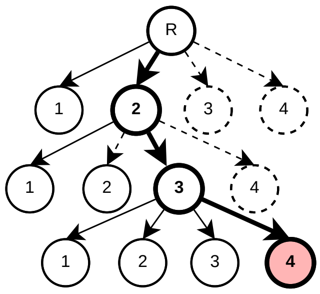
Fig. 10 A balanced Quadtree with four levels

implement, it does not however well capture the complexity of the considered map, as it is the case of our considered temperature use case. Indeed, all the polygons, representing regions or departments, are not of the same shape complexity. For instance, areas located on regions or department boundaries are more complex and consequently need more indexing cells than areas located in the middle. To capture this difference, the Quadtree needs to be expended in some region more than in others, leading to unbalanced Quadtree.

4.2.2 Unbalanced Quadtree



(a) Unbalanced decomposition



(b) Explored nodes

Fig. 11 Unbalanced decomposition

The main problem with the balanced Quadtree is that all the leaves have the same level of detail (i.e. they are on the same level of the Quadtree). However, in real applications, we want variable precision depending on the position of the Quadtree cell compared to the polygon's boundary. The unbalanced Quadtree is an improvement of the balanced Quadtree. It handles better precision and hence improves the processing time. But it still requires, as for the balanced Quadtree, a trade-off between the precision and the processing time. In fact, higher the precision, longer the processing time and *vice versa*.
For instance, using an unbalanced quadtree (in Figure 11), we explored only three levels in order to find the leaf containing the query point. If we use a balanced quadree, we would have to explore all the levels in order to find the leaf containing the query point.

4.3 GeoHash Approach

Based on the GeoHash geocoding system [21], we used a grid decomposition in order to create a spatial indexing structure. The result is a hierarchical spatial data structure that subdivides space into grid shapes (see Figure 13). Similarly as in the previous approach, the polygons are mapped on this grid and the GeoHash coding is used after to locate a point on the corresponding polygon. Geohashes offer properties like arbitrary precision and the possibility of gradually removing characters from the end of the code to reduce its size (and gradually lose precision). GeoHash is very light, it is composed of two main steps: binary code generation and base 32 encoding. First, it uses recursively series of divisions to generate a binary code, Figure 12 illustrates the first step of binary code generation. Then, it uses a base 32

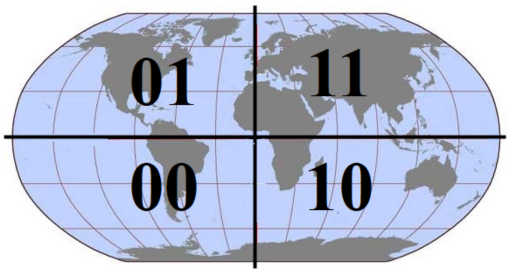


Fig. 12 Binary coding of GeoHash

map to encode the binary result in a alpha-numerical string format. For instance, to get a 4 characters length GeoHash from the coordinates (48.870321, 2.305931), only 20 binary operations are needed:
Binary code : 11010 00000 01001 11100
Decimal code: 26 0 9 28
GeoHash: u09w.
Unlike the previous decomposition method where we store the whole data structure (i.e. the Quadtree), the only stored data in this method is the cell's GeoHash as a unique key and the corresponding region or department as the value.

GeoHash is used mainly as a Universal Unique Identifier (UUID) to represent coordinates or as GeoTags.

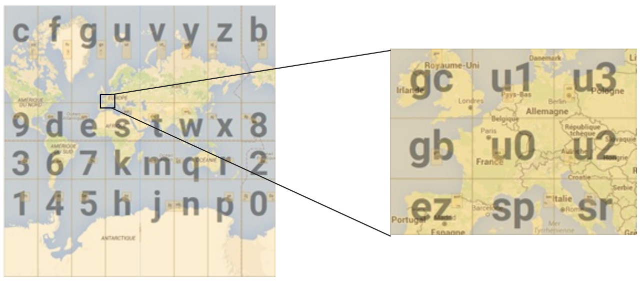


Fig. 13 Geohash

Table 2 Error margin for a GeoHash length

Length	Lat. bits	Error(km)	Long. bits	Error(km)
1	2	±2487.1	3	±2504.7
2	5	±310.9	5	±626.1
3	7	±77.7	8	±78.2
4	10	±9.7	10	±19.5
5	12	±2.4	13	±2.4
6	15	±0.3	15	±0.61
7	17	±0.076	18	±0.076
8	20	±0.009	20	±0.019
9	22	±0.002	23	±0.002
10	25	±0.0005	25	±0.0005

We have however used some interesting GeoHash properties in order to index the grid:

– Depending on the desired string's length, the generated GeoHash has a margin error in longitude and latitude (refer to TABLE 2). So in fact, GeoHash produces a rectangle rather than a point. The grid's cells (i.e rectangles) will be indexed using this property.
  $encode(latitude, longitude, length)$: is the function that produces the hash string.
  $decode(string)$: is the function that restores GPS coordinates from the hash string.
  $decode(encode(x, y, length)) = (x', y')$,
  where: (x', y') is the center of the rectangle that contains every (x, y) such as:

$$x \in [x' - Lon_{err}, x' + Lon_{err}]$$
$$\text{and } y \in [y' - Lat_{err}, y' + Lat_{err}]$$

– GeoHash loses in precision by removing characters from the end of the hash string. So depending on the desired grid's cells size, we can variate (increase or decrease) the GeoHash length.
– Any two points belonging to the same cell will have the same GeoHash. We will use this property for the online query, if the query point has the same GeoHash as a grid's cell index, then it is contained in this cell.

For the grid structure, we have also the choice between a uniform grid and a non uniform grid.

4.3.1 Uniform grid

A uniform grid is a grid where all its cells have the same size. If we want a precision corresponding to 10 characters (refer to Table 2), all the grid cell will have a GeoHash length of 10 characters. To find if a query point is contained in one of the grid's cells, we generate a GeoHash of 10 characters for the query point. If the query point has the same GeoHash as one of the grid's cells, then it is contained in that cell.

4.3.2 Non uniform grid

A non uniform grid is a grid where the cells have not the same size (i.e. a grid with a variable precision), and hence they have not the same GeoHash length. To transform a uniform grid into a non uniform grid, we have just to fuse the cells of a polygon into a meta-cell if the polygon contains all the sub-cells of the meta-cell. For instance, if a polygon Pi contains all the sub-cells of the meta-cell u $(u0, u1, \ldots, uz)$, we can fuse them and replace them with a single cell u. This will improve the memory requirement (because we will store less cells) but it will add more complexity to the query. Since the grid cells have not a fixed GeoHash length, we must variate the GeoHash length of the query point and see if it exists in the grid indexes, see the example bellow.

Example

We illustrate our proposed approach through the following example: assuming a CV is located in (48.87, 2.3059). The first step is to generate the GeoHash of this CV's location with the desired precision. If we use a uniform grid, it is known that all the cells have the same GeoHash length. However, if we use a non uniform grid, we have to generate a GeoHash of with the maximum precision (e.g., a precision corresponding to 10 characters length, see Table 2)

$$encode(48.870321, 2.305931, 10) = u09wh2x4k0.$$

The second step is to find to which cell corresponds this GeoHash. Using a uniform grid, this step is straight forward, all the cells have the same GeoHash length so we have just to check if a cell with this index exists. In the other hand, the non uniform grid's cells have not the same GeoHash length, so we will start by checking if the 10 characters length GeoHash ($u09wh2x4k0$) exists in the grid index. If it does not exist, we will reduce the precision to 9 characters length ($u09wh2x4k$) and so on until we find the corresponding cell. In our example, we found that the 5 characters length index ($u09wh$) exists and its corresponding department is Île-de-France (see Figure 14).

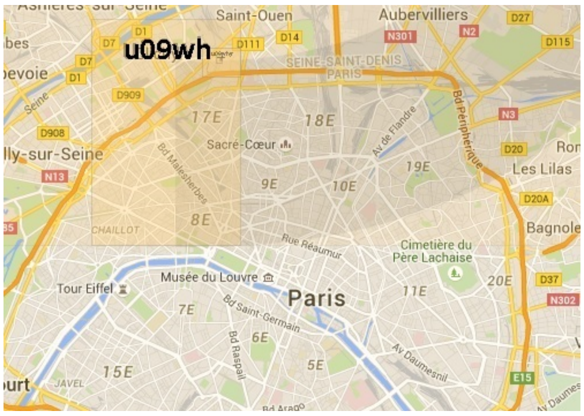


Fig. 14 A 5 characters length Geohash of the GPS coordintes (48.870321, 2.305931).

5 Experimental Evaluation

To evaluate the effectiveness of the studied approaches, we have implemented them within an instance of the stream processing engine IBM Streams v3.2.1 deployed on a test node consisting of 64 vCPU with a frequency of 2.4 Ghz and 40 GB of RAM. For our tests on the temperature use case, we re-injected stored CVs data in order to have a real coordinates distribution. We used the map of France available at the OpenStreetMap [2] as *Keyhole Markup Language* (KML) [8] files. This map contains 96 metropolitan departments, represented as polygons with a mean complexity of 480 vertices per polygon. To check if a polygon contains a query point, we used the *Point In Polygon* (PIP) function isContained() [15] implemented in the geospatial toolkit 1.1.1. This function takes two geometric objects (polygon, line, point) as an input and returns whether an object contains the other. In our case, it takes a polygon and a point and returns whether the polygon contains the point or not.

To measure the performance of each of the studied approaches, we constructed the following scenarios: each CV is sending one frame per second. The number of CVs is gradually increased (i.e., increasing the data rate) and we measured the corresponding processing time of the stream platform. The number of concurrent CV that the method can handle is defined by the number of processed event within one second (because a vehicle send generally one frame per second). In order to meet the real time requirement, the whole processing time has to remain under one second, otherwise congestion in the stream chain occurs.

5.1 Exhaustive Approach

As expected, the Exhaustive approach presents the poor performance among all the studied approaches. In fact, as highlighted in Figure 15, the rate of concurrent CVs has not exceeded 21 CVs, where the processing time remains under 1 second. Beyond, the real-time constraint is no more ensured. This clearly shows the poor performance of this approach and motivate the need for more effective approaches.

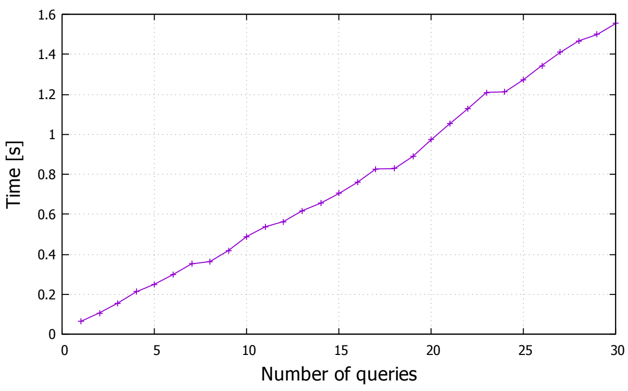


Fig. 15 Exhaustive search

5.2 Two Steps Approach

Within this approach, we have chosen the *bounding box* as an approximation for the filtering phase. The latter is known to be rough and inaccurate but profits from simplicity among other approximation methods [5]. to have the better ratio simplicity/quality among the other approximations. To evaluate the effectiveness of the filtering operation, we applied bounding box on the set of the French polygons and counted the number of candidates after the filtering operation. We recall that the number of polygons in input is 96, the results are showed in Table 3.

Table 3 Filtering quality of bounding box

# of candidates	0	1	2	3	4	5
%	1.3	40.9	45.9	11.2	0.6	0.0

As showed in Table 3, the filtering part using the bounding box shown its efficiency. In the worst case, we get a set of 4 candidates out of 96 initial polygons. In 40.9% of cases, the filtering outputs 1 candidate. In 45.9% of cases the filtering outputs 2 candidates, etc. The PIP component is then composed of two operators: filtering and refinement, connected in pipeline fashion as shown in Figure 16. We recall that the overall processing time of this component is determined by the slower operator.

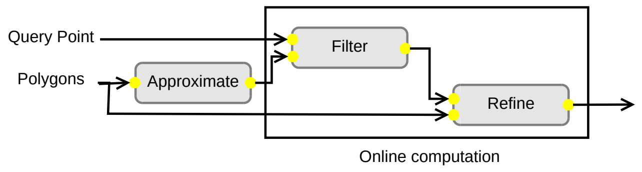


Fig. 16 The two steps streaming application

In Figure 17 we plot the performance of these two operators with regard to the rate of CVs (i.e., number of processed events). Even we remark a noticeable increase in the performance in comparison to the exhaustive approach, it remains however under the expected performance for automotive applications. In fact, the number of concurrent CVs per second does not exceed 600, which very far from the expected millions of CVs. Even the platform can support replicated components (i.e., increasing the number of this component and performing parallel processing associated with load balancing techniques), we considered, from practical point of view, that this approach is still mis-performing.

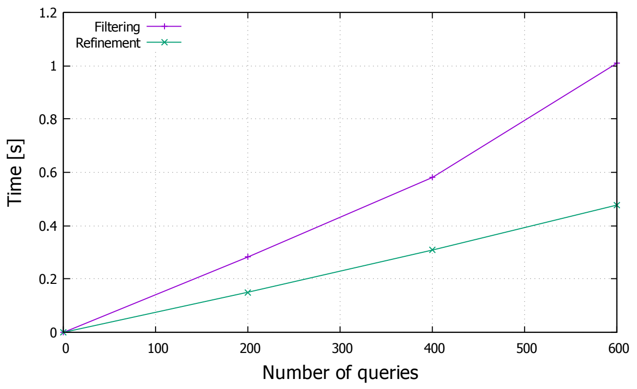


Fig. 17 The two-steps method

5.3 Decomposition Approach

We tested the two variants of this approach: the one using a balanced Quadtree and the other using an unbalanced Quadtree. The results are plotted in Figure 18.

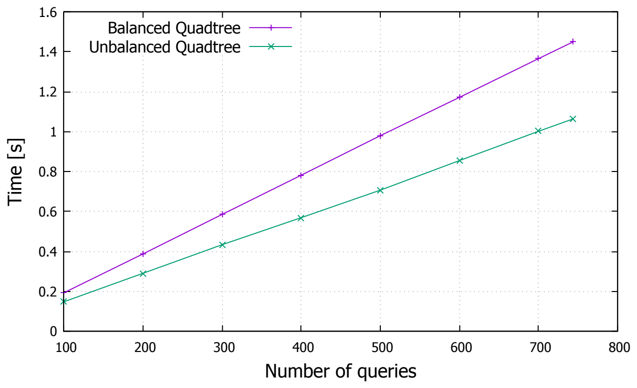


Fig. 18 Decomposition method

As expected, the unbalanced version improves over the balanced one within our target temperature application. We have to note one surprising result: the performance of the balanced version was poor compared to the two steps method. In fact, the number of supported concurrent CVs has no more exceeded 500. The reason is due to the fact that the depth of the indexing structure was important in order to cope with the complexity of the considered polygons within our target application. This increases automatically the "exploration" time within PIP function.

On the other hand, the performance of the unbalanced version is better than the previous approach (i.e., Two Steps Approach). However the improvement is not noticeable: from 600 to 750 concurrent CVs. Here again, the performance is still under what is expected within automotive applications.

5.4 GeoHash

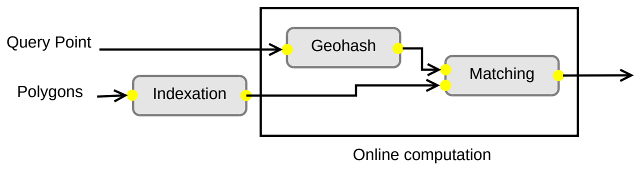


Fig. 19 The streaming application of GeoHash

In the GeoHash implementation, there are three main operations (see Figure 19).

– Indexation: A custom operator that creates the grid and maps the polygons into it.
– Geohash: Generates the GeoHash from the longitude, latitude coordinates of the query point.
– Matching: Matches the generated GeoHash with the corresponding polygon by verifying if the query point's GeoHash exist in the hash table.

To evaluate the effectiveness of this method, we measured the online operations (i.e. geohash and matching operations). These two operations are interconnected in pipeline mode and hence the overall time is the time of the slowest operation. To measure the performance

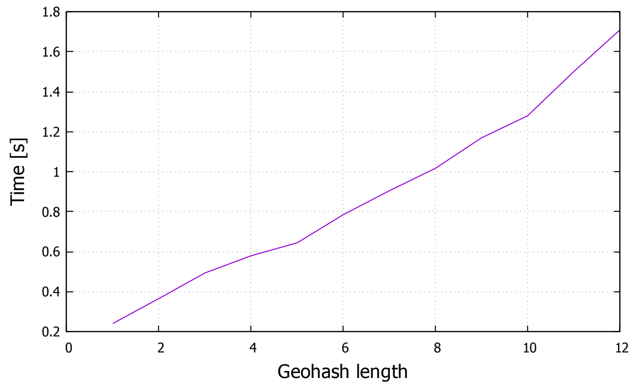


Fig. 20 Geohash computation for 1 million coordinates

of the first operation (i.e. Geohash computation), we variate the length of GeoHash and evaluate the impact by measuring the time to compute the GeoHash of 1 million coordinates, the results are plotted in Figure 20. As the Figure shows, GeoHash computation is dependent of the GeoHash length. Indeed, more precise is the GeoHash, higher the computation time. However, it stays acceptable, it takes 904 ms to generate a 7 characters length Geohash for 1 million coordinates. For automotive applications, a 7 characters length GeoHash is precise enough to determine the department (refer to Table 2).

We tested the two variants of the grid structure (i.e. uniform and non uniform grid) and we measured the time of matching the GeoHash with its polygon's cell, the results are plotted in Figure 21. We note that using the uniform grid, the key length does not affect the performance, a 1 character length key or a 12 characters length key will take the same time (about 232 ms for 1 million queries). However, it takes 5.62 s using the non uniform grid, this can be justified by the fact that we does not know beforehand the length of the containing

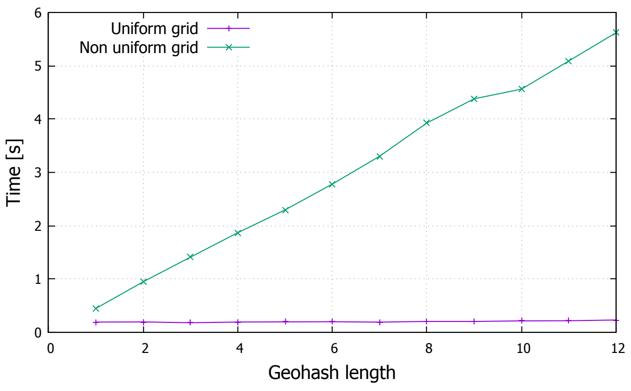


Fig. 21 Matching operation for 1 million coordinates

cell's index and hence we have to variate the length of the GeoHash.

6 Conclusion

Within this paper, we described in depth the problem of data fusion for automotive applications. We illustrate the problem of data fusion through a real use case of environment monitoring (i.e., temperature monitoring). In order to speed-up the process of data fusion, several approaches have been explored. Each of the proposed methods has been implemented and validated using the stream computing engine (i.e. IBM Streams). However, the first methods have not been efficient # which resulted in congestions within the stream computing platform. The last method based on GeoHash indexation, due to its simplicity, optimized drastically the query time compared to the other methods (i.e., exhaustive, two Steps, and decomposition using Quadtree). For future work, we will be implementing and evaluating the GeoHash approach within other services, more precisely in Car-to-Car over the cloud architectures [19]. In these services, we are interested to send information (e.g., alerts, advertising, etc) to vehicles based on its geo-location.

References

1. Météo france. URL `http://www.meteofrance.com`
2. Openstreetmap. URL `http://export.openstreetmap.fr/contours-administratifs/`
3. Apache: Hadoop. URL `https://hadoop.apache.org/`. Version 2.6.3
4. Berchtold, S., Keim, D.A., Kriegel, H.P.: The x-tree: An index structure for high-dimensional data. In: Proceedings of the 22th International Conference on Very Large Data Bases, VLDB '96, pp. 28–

39. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1996)
5. Brinkhoff, T., Kriegel, H.P., Schneider, R.: Comparison of approximations of complex objects used for approximation-based query processing in spatial database systems. In: Data Engineering, 1993. Proceedings. Ninth International Conference on, pp. 40–49 (1993)
6. Finkel, R.A., Bentley, J.L.: Quad trees a data structure for retrieval on composite keys. Acta Informatica **4**(1), 1–9
7. Fournier, A., Montuno, D.Y.: Triangulating simple polygons and equivalent problems. ACM Trans. Graph. **3**(2), 153–174 (1984)
8. GOOGLE: Keyhole markup language. URL `https://developers.google.com/kml/`
9. Gordon, M.I., Thies, W., Amarasinghe, S.: Exploiting coarse-grained task, data, and pipeline parallelism in stream programs. SIGARCH Comput. Archit. News **34**(5), 151–162 (2006)
10. Guttman, A.: R-trees: A dynamic index structure for spatial searching. SIGMOD Rec. **14**(2), 47–57 (1984)
11. Haines, E.: Graphics gems iv. chap. Point in Polygon Strategies, pp. 24–46. Academic Press Professional, Inc., San Diego, CA, USA (1994)
12. Hill, M.D.: Scalable Shared Memory Multiprocessors, chap. What is Scalability?, pp. 89–96. Springer US, Boston, MA (1992)
13. Hirzel, M., Soulé, R., Schneider, S., Gedik, B., Grimm, R.: A catalog of stream processing optimizations. ACM Comput. Surv. **46**(4), 46:1–46:34 (2014)
14. IBM: Ibm streams: Capture and analyze data in motion. URL `http://www-03.ibm.com/software/products/en/ibm-streams`
15. IBM: iscontained. URL `http://www-01.ibm.com/support/knowledgecenter/SSCRJU_3.2.1/`
16. Isaacson, C.: Understanding Big Data Scalability: Big Data Scalability Series, 1st edn. Prentice Hall (2014)
17. Kirkpatrick, D.G.: Optimal search in planar subdivisions. SIAM J. Comput. **12**(1), 28–35 (1983)
18. Labrinidis, A., Jagadish, H.: Challenges and opportunities with big data. Proceedings of the VLDB Endowment **5**(12), 2032–2033 (2012)
19. Lee, E., Lee, E.K., Gerla, M., Oh, S.Y.: Vehicular cloud networking: architecture and design principles. IEEE Communications Magazine **52**(2), 148–155 (2014). DOI 10.1109/MCOM.2014.6736756
20. Lipton, R.J., Dobkin, D.P.: Complexity measures and hierarchies for the evaluation of integers and polynomials. Theor. Comput. Sci. **3**(3), 349–357 (1976)
21. Niemeyer, G.: Geohash. URL `http://geohash.org/`
22. Orenstein, J.A.: Redundancy in spatial databases. SIGMOD Rec. **18**(2), 295–305 (1989)
23. O'Rourke, J.: Finding minimal enclosing boxes. International Journal of Computer & Information Sciences **14**(3), 183–199
24. Preparata, F.P., Hong, S.J.: Convex hulls of finite sets of points in two and three dimensions. Commun. ACM **20**(2), 87–93 (1977)
25. Preparata, F.P., Shamos, M.I.: Computational Geometry - An Introduction. Texts and Monographs in Computer Science. Springer (1985)
26. Rosenfeld, A.: A converse to the jordan curve theorem for digital curves. Information and Control **29**(3), 292 – 293 (1975)
27. Sahr, K., White, D., Kimerling, A.J.: Geodesic discrete global grid systems. Cartography and Geographic Information Science **30**(2), 121–134 (2003)
28. Samet, H., Rosenfeld, A., Shaffer, C.A., Webber, R.E.: A geographic information system using quadtrees. Pattern Recognition **17**(6), 647 – 656 (1984)
29. Seidel, R.: A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. Comput. Geom. **1**, 51–64 (1991)
30. Shahrivari, S.: Beyond batch processing: Towards real-time and streaming big data. CoRR **abs/1403.3375** (2014)