Real-time detection of moving crowds using spatio-temporal data streams

Dmytro Arbuzin

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 17.05.2017

Thesis supervisor:

Prof. Keijo Heljanko

Aalto University
School of Science

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Dmytro Arbuzin

Title: Real-time detection of moving crowds using spatio-temporal data streams

Date: 17.05.2017 Language: English Number of pages: 4+81

Department of Computer Science

Professorship: Computer Science

Supervisor and advisor: Prof. Keijo Heljanko

Over the last decade we have seen a tremendous change in Location Based Services. From primitive reactive applications, explicitly invoked by users, they have evolved into modern complex proactive systems, that are able to automatically provide information based on context and user location. This was caused by the rapid development of outdoor and indoor positioning technologies. GPS modules, which are now included almost into every device, together with indoor technologies, based on WiFi fingerprinting or Bluetooth beacons, allow to determine the user location almost everywhere and at any time. This also led to an enormous growth of spatio-temporal data.

Being very efficient using user-centric approach for a single target current Location Based Services remain quite primitive in the area of a multitarget knowledge extraction. This is rather surprising, taking into consideration the data availability and current processing technologies. Discovering useful information from the location of multiple objects is from one side limited by legal issues related to privacy and data ownership. From the other side, mining group location data over time is not a trivial task and require special algorithms and technologies in order to be effective.

Recent development in data processing area has led to a huge shift from batch processing offline engines, like MapReduce, to real-time distributed streaming frameworks, like Apache Flink or Apache Spark, which are able to process huge amounts of data, including spatio-temporal datastreams.

This thesis presents a system for detecting and analyzing crowds in a continuous spatio-temporal data stream. The aim of the system is to provide relevant knowledge in terms of proactive LBS. The motivation comes from the fact of constant spatio-temporal data growth and recent rapid technological development to process such data.

Keywords: Online clustering, Complex Event Processing, Distributed Systems

Contents

1 Introduction

Data is the new oil!

Clive Humby, ANA Senior marketer's summit, November 2006

The phrase "Data is the new oil" was first time used about 10 years ago and since then was repeated dozens of times by different technology capitalists in different variations [1]. And indeed the concept of "Data" and especially "Big Data" has undergone a huge transformation from a buzz word and a fuzzy concept less then decade ago to a separate business and research industry nowadays. Moreover, according to the latest Gartners' Technology Hype Cycle chart [2] "Big Data" is not even on the chart anymore. It dropped out in 2015 meaning that technology is not a hype anymore and became an industry standard and a granted reality. Instead Machine Learning is on a peak of a curve currently 1.

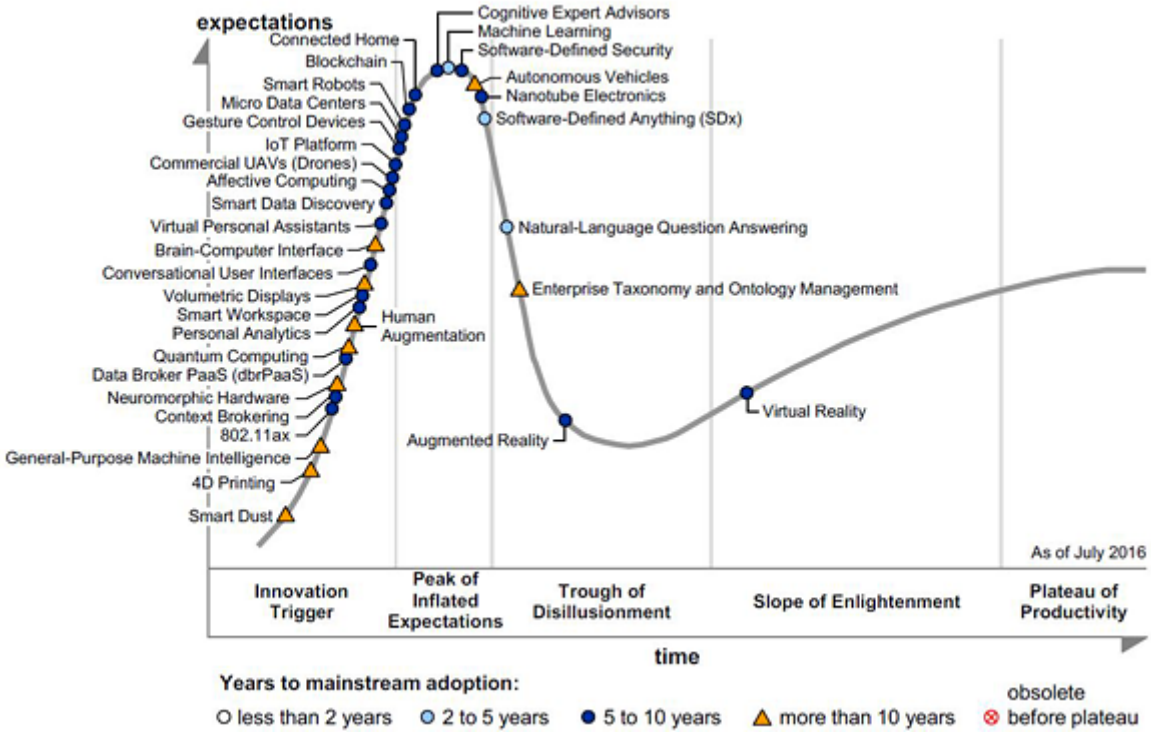


Figure 1: Hype Cycle for Emerging Technologies, 2016. Source: [2]

The full quote of Clive Humby makes oil analogy even more meaningful. *"Data is the new oil. It's valuable, but if unrefined it cannot really be used. It has to be changed into gas, plastic, chemicals, etc. to create a valuable entity that drives profitable activity; so must data be broken down, analyzed for it to have value."*. Following this logic, we can think of Statistics, Data Mining and Machine Learning as of tools to break down the data in order to obtain the real value.

Nowadays technologies can easily manage almost all steps of data processing and management.

- **Data Collection.** IoT (Internet of Things) is becoming another new different industry and with billions of sensors gathering all kinds of information. Currently we have much more data that we are able to analyze and exploit.
- **Data Transportation.** 5G, ZigBee, new standards of WiFi and Bluetooth protocols. They all were developed with the aim to handle huge amount of data with a low latency. Basically communication layer is one step ahead of our current needs and ready to transport all available bytes and bits.
- **Data Storing.** Database industry is also going through a tremendous change. NoSQL is no longer a "buzzword". NewSQL databases are appearing every day. In-memory databases have become a standard already due to cheap RAM.
- **Data Processing.** MapReduce framework [3] has become a golden standard for Big Data processing already a decade ago. Today there are many more sophisticated solutions such as Apache Spark and Apache Flink. These in-memory engines are able to process not only big datasets, but also heavy distributed datastreams.
- **Data Representation.** This area only indirectly related to traditional Big Data research and always stays a bit aside. However, Information Visualization is a different topic being toughed in academia and hugely important in business, especially in BI (Business Intelligence). The popularity of tools such as R, Tableau, QuickSight from AWS, Zoomdata and many others clearly indicates the demand and development, which is going on in this area.

All of these engineering and scientific fields together create a perfect environment to manipulate data and extract meaningful knowledge. Each of these areas has its own challenges and interesting problems to solve, however there is one more area, which has the most potential and space for improvement. This area is data analysis and algorithms. The actual state of play is that the majority of current most common and wide-spread algorithms and methods in machine learning and data mining were introduced long time ago to solve theoretical problems. Many of them assume by default that the data is offline and available in one place. Moreover, some of algorithms converge with a huge computational complexity and do not consider other important factors such as network latency and space complexity, which cause a lot of impact in distributed environments.

The motivation for this thesis comes from three different computer science areas: location-based services (LBS), real-time applications and data mining. We exploit the fact that current data processing technologies require modern algorithms and new approaches.

1.1 Real-time applications

In the recent years there was a tremendous shift from a batch processing and offline data analysis to a streaming data processing and in-memory computations. And indeed, almost every data source is a stream by its nature and current applications need to provide useful insights to their users as soon as possible. Data pipelines and data flow management have become important areas of a computer science. Technologies like Complex Event Processing (CEP) were introduced over decade ago, but until recently were applied only in narrow domain areas such as stock markets. The development of data processing technologies such as Apache Spark and Apache Flink inspired by MapReduce allows us to process huge amount of data with relatively small cost.

Stock markets, online advertising, internet of things and many more industries require efficient solutions to process heavy data streams.

1.2 Data mining

This is another hot topic in a business as well as in an academia nowadays. Much of recent research projects in data mining and machine learning are focused on transformation popular algorithms and methods to satisfy needs of online applications and distributed environments. As we will see throughout the thesis this is not a trivial task and there is still a lot of space for improvement. In fact, there are very few fully distributed machine learning algorithms, which are able to work with streaming data. Due to some fundamental reasons very few of them can be as efficient in a stream processing as in a batch processing. The development of incremental algorithms [4] solves the problem for many use-cases, however there is no known general solution across domains. One cannot generally transform a non-incremental algorithm into an incremental one. Moreover, incremental algorithms do not solve problems of distributed computing.

Inability of adjusting fundamental general purposed data mining algorithms to requirements of streaming and distributing processing has created a shift to domain-specific solutions. The main difference is that domain-specific algorithms exploit metadata and particular features of a dataset and usually solve some specific task.

1.3 Location-based services

Location-based services are currently changing as well. From basic services on-demand like online maps and route planning applications they evolved to real-time location and context-based notification applications, which are currently available on every mobile platform. However, almost all available services are targeting private users and still remain quite primitive. It is surprisingly shocking, how little information a person can obtain from location services, considering all volume and variety of geodata that are available.

Lately much research in this area is conducted in two particular areas: local positioning systems and personification of services. First ones are able to locate people

inside buildings using beacons, sensor identification technologies, Wi-Fi networks, etc. Personification allows better targeting. With the help of technologies, such as geofencig, it allows to provide people more personal and context-base information.

However, there are very few research projects ongoing, which try to exploit geolocation information of crowds and not individuals. This is especially true for a real-time context.

1.4 The main motivation behind the thesis

The topic of this thesis is *"Real-time detection of moving crowds using spatio-temporal data streams"*. This thesis is a concept design of a possible future system, which not only combines latest researches and technologies stated above, but also introduces some new methods and techniques, which were not used previously. The suggested system is able to detect clusters of moving objects, to recognize various interesting patterns based on cluster discoveries and to provide almost real-time insights to a user. From a technical point of view, the designed system is fully distributed and is able to work with heavy spatial data streams.

The main motivation behind such application comes from many domain areas. The most obvious use-case is real-time analyzing of a traffic flow. By detecting clusters of vehicles, the system can provide almost real-time information about the situation on roads. By informing other drivers about traffic congestion the system can help to avoid or reduce road rages. Besides vehicles system can analyze people flows in places of mass gatherings. For example, during concerts or football games the system can discover how the crowd moves and react of this based on provided information (for example we can open extra entrances or send more taxi cars to big group of people). The system can detect places, where crowds often occur. Based on this information we can dynamically place geofences in order to show more advertisement or just some important notifications.

All these problems are possible to solve nowadays even without developed system, however we should consider this research in a context of automation and a trend of self-driving vehicles. Currently, in order to detect a moving crowd and take some action on it, manual interference is required. Detecting can be done automatically already today, but triggering a specific action based on some specific pattern is not so common in the area of LBS. Most of today notification mechanisms are based on basic parameters such as thresholds, time intervals or simple counts. In this thesis we also present a number of interesting patterns, which can be detected by a system while analyzing geo-temporal data stream. Based on these patterns the system can emit new triggering events using CEP technology, which minimize human intrusion to a service.

As it was already mentioned, current data mining techniques move towards domain specific solutions in order to provide better results. This thesis is focused on, but not limited to, vehicle systems and traffic flow systems in big cities and urban areas.

2 Related Work

Being developed in an application area of LBS, this thesis deals with different research and domain fields. The core concept can be separated into two major research topics: cluster detection and pattern analysis. The nature of the data is geospatial. From a technical side the designed system operates on a continuous data stream in a distributed manner.

This chapter provides some fundamental knowledge about each of these areas as well as discusses relevant research papers related to the thesis.

2.1 Clustering. Fundamentals

The initial challenge of the thesis problem (detecting gatherings of moving objects) is an important part of data mining and statistics. Clustering is in a category of unsupervised learning, meaning that algorithms "learn by observing" and do not require previous training or set up.

Clustering or cluster analysis is the process of partitioning a set of data objects into subsets. Each subset is a cluster, such that objects in a cluster are similar to one another, yet dissimilar to objects in other clusters [5]. Unique features, based on which items are combined into clusters are also called dimensions. Using dataset with coordinates X and Y means that we have two dimensions. General purposed algorithms are aimed to handle N dimensions, however, as we will see in a following discussion, it is rather hard. The technique which decrease the number of data dimensions is called dimensionality reduction.

There is no common terminology and strict categorization of cluster methods or models, however it is useful to provide a brief overview and history of research in order to justify the clustering method, which was later developed for a thesis implementation. There are next common clustering method groups according to [5]: partitioning, hierarchical, density-based and grid-based methods. Modern algorithms often mix techniques from different groups, therefore it is hard to put some algorithms just to one group.

2.1.1 Partitioning methods

This is the simplest and the most fundamental clustering technique. Given a set of n objects and a number of required clusters k, a partitioning method constructs k partitions of the data, where each partition represents a cluster and $k \leq n$. Algorithms of this group use a distance measure as a main criteria of similarity (usually Euclidean, but can also use Mahalanobis). Partitioning methods discover clusters of a spherical shape.

Probably the most popular algorithm of this group is K-Means. Being introduced more than 50 years ago it still remains very popular and widely utilised [6]. This indicates the difficulty in designing a general purpose clustering algorithm. The algorithm uses entroids of a cluster C_i to represent it. In the best case scenario centroid is a clusters' center point. K-means calculates the mean variable of neighbourhood

points to detect a centroid. There are special variations of K-means that use medoids and other notions of average to define centroid. In general, algorithms of this group are easy to understand and implement. There are many known extensions of k-means, which address the problem of streaming and big datasets [7], [8] and can converge in one pass at maximum. Nevertheless, initial restrictions of the amount of clusters as input parameter and inability to detect clusters of arbitrary shapes together with poor outlier handling make this group unsuitable for this thesis problem.

2.1.2 Hierarchical methods

Hierarchical methods work by grouping data objects into a hierarchy or “tree” of clusters. Such method creates a hierarchical decomposition of the given set of data objects. Hierarchical methods can be two types. **Agglomerative** or "bottom up": each data item starts in its own cluster and pairs of clusters are merged as one moves up the hierarchy. **Divisive** or "top down: all data items groups as one cluster at the beginning. Methods of this group can utilise any notion of distance. To maintain cluster hierarchy methods usually use some kind of tree data structures (B-tree for example). The popular approach is to integrate hierarchical clustering with other clustering techniques, resulting in multiple-phase (or multiphase) clustering.

The most famous algorithm of this group is Balanced Iterative Reducing and Clustering using Hierarchies or just **BIRCH** [9]. Being introduced in 1996 by Zhang et al. BIRCH has become one of the most adapted and influential algorithms. BIRCH operates in two phases:

- **Phase 1:** It uses hierarchical technique to perform initial micro-clustering calculations. BIRCH was the first clustering algorithm, which used a clustering feature (CF) vector in order to maintain information about clusters. CF vector of the cluster is defined as a triple: $CF = (N, LS, SS)$, where N - a amount of data points, LS - a linear sum of N data points, i.e., $\sum_{i=1}^{N} X_i$, and SS - is the square sum of the N data points, i.e., $\sum_{i=1}^{N} X_i^2$. Structures LS and SS are n-dimensional arrays [10]. CF vectors for each cluster are stored in the CF-tree. It is a height-balanced tree.

- **Phase 2:** BIRCH applies a (selected) clustering algorithm (for example iterative partitioning) in order to cluster leaf nodes of a CF-tree, which removes sparse clusters as outliers and groups dense clusters into larger ones.

CF vector from phase one has two important features: *Incrementality* and *Additivity*. This allows very efficient usage of memory and also allows working with data streams. After being introduced in BIRCH for the first time, CF vectors were adopted and now widely used in many prominent clustering algorithms [8]. For a second phase an iterative method can be selected as well.

In general, BIRCH demonstrates $O(n)$ complexity, where n is a number of input objects. Due to algorithms' features it works well with streaming data and also have a good linear scalability. There are also few versions that operate in parallel [11]. However, using notion of distance BIRCH converges well only with the numeric data and discovers clusters only in a spherical shape.

2.1.3 Density-based methods

This group of clustering algorithms uses the notion of density and can expand clusters in any direction. In other words, these algorithms can discover clusters of arbitrarily shapes. They also handle outliers very well.

Density-based spatial clustering of applications with noise (DBSCAN) [12] is a perfect example of a such algorithm. It was introduced just two months earlier than BIRCH. And as well as BIRCH has created a whole new family of clustering algorithms, which are used today. DBSCAN was one of the first algorithms to address a spatial dimension's problem, even though it is still a general purposed method aimed to converge with a high dimensionality. One cannot treat spatial dimension as just any other dimension due to its significance. Usually it is the only dimension, which we take into consideration. As it will be shown later it often goes together with temporal dimension.

DBSCAN has only three input parameters. $minPts$ - the minimum amount of data points to be considered a cluster. It should be set carefully, too low parameter leads to many small clusters and too high will produce many noise data points and very few clusters. Another parameter is ϵ (eps) - the maximum distance between two points to be considered neighbors. The algorithm picks randomly any point p, marks it as visited and checks if there are any other points so the $distance(p, p_i) \leq \epsilon$. If there are enough points so that the amount of neighborhood points $>$ $minPts$, the algorithm creates a cluster. If there is no points within ϵ, the algorithm marks p as noise. In case there are neighbors, but not enough to create a cluster, the algorithm moves to another point among neighbors to check if any unvisited points reachable from there. Figure 2 illustrates this effect, which is called density-reachability.

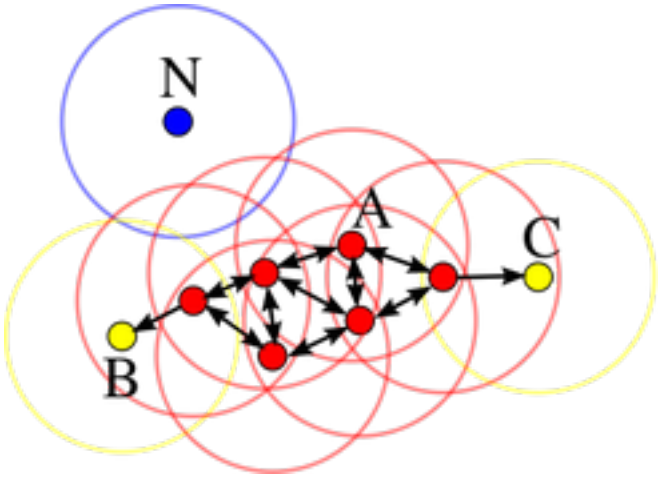


Figure 2: DBSCAN density-reachability example: points A, B, C form a cluster. N is a noise. $minPts = 4$. Source: [13]

Another well-known density-based algorithm is called OPTICS [14]. OPTICS is based on a very clever idea: instead of fixing $minPts$ and ϵ, it only fixes the $minPts$ and plot the radius at which a cluster would be considered dense by

DBSCAN. In order to sort the objects on this plot, it processes them in a priority heap, so that nearby objects are nearby in the plot.

DBSCAN time complexity is rather high, as it visits every point multiple times. The average complexity is $O(log(n))$ and the worst case remains $O(n^2)$. OPTICS comes at a cost compared to DBSCAN. Largely because of the priority heap, but also as the nearest neighbor queries are more complicated than radius queries of DBSCAN.

In general, performing very well in term of cluster discovery and flexibility both algorithms are costly and require more than a single pass. Also very hard to parallel, as a sequential pass through all points is required.

2.1.4 Grid-based methods

All previously discussed methods are data-driven, meaning that they all adjust to an initial distribution of objects and partition them based on this distribution. Grid-based methods from another hand locate data objects into predefined cell space (grids) and then analyze each grid independently. This approach is called space-driven.

Grid-based methods use multiresolution grid data structure. It quantizes the object space into a finite number of cells that form a grid structure on which all of the operations for clustering are performed [5].

Some of the most popular grid-based algorithms are STING: STatistical INformation Grid [15] and CLIQUE: An Apriori-like Subspace Clustering Method [16].

STING splits the data space in rectangular cells using hierarchical and recursive way. Figure 3 illustrates such rectangular structure, where each layer of cells correspond to different level of resolution. Each cell contains statistical summary regarding data points belonging to it, which is very useful for a data analysis tasks. Values are precomputed and stored as statistical parameters.

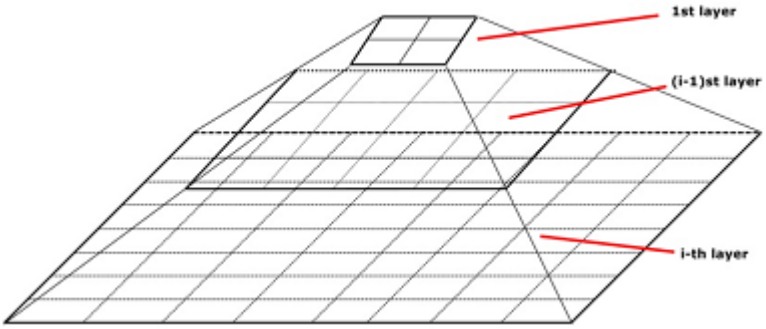


Figure 3: Hierarchical structure for STING clustering. Source: [5]

CLIQUE (CLustering In QUEst) is very simple and straightforward, but yet powerful grid-based algorithm for finding clusters. CLIQUE partitions each dimension into non-overlapping intervals, thereby partitioning the entire embedding space of the data objects into cells. However, in an opposite to STING it has only one layer.

It uses a density threshold to identify dense cells and sparse ones. A cell is dense if the number of objects mapped to it exceeds the density threshold [5].

It is important to notice that almost every grid-based algorithm utilizes the notion of density. They discover dense grid-cells using just simple count (CLIQUE) or some kind of density value based on the amount of data points and some weighted coefficients. Therefore, one can consider these algorithms as density-based with respect to a predefined space. Dense grid cells can be considered as micro-clusters (BIRCH).

Grid-based methods provide some fundamental advantages compare to other methods. Grid structure by default allows parallel processing and incremental updating, as each cell can be computed separately. The only linear process, which is necessary, is to combine neighboring dense grid-cells into clusters. However, this step can be also optimized. In the case of a hierarchical grid structure (STING) merging can be done recursively by going each step one layer higher.

As for disadvantages these methods are heavily dependent on input parameters and underlying grid structure. Also finding arbitrary-shaped clusters can be challenging as it requires extra processing step to go from separate cell to single data-point and sometimes this not possible at all.

2.2 Streaming

Most of discussed clustering algorithms and methods assume by default to work with static datasets and have all data available in one place from the very beginning. The problem of processing a data stream into clusters was formulated long time ago and still widely discussed in academia. Some of classic algorithms such as BIRCH are incremental by default and easily fit to a streaming architecture. However, for most of clustering methods streaming extensions were developed.

Before the main discussion about streaming clustering algorithms it is necessary to discuss basic streaming related features and fundamentals, which play important role in an algorithm design and data processing. In this discussion we follow mainly "The world beyond batch: Streaming 101" article by Tyler Akidau [17], but also cite some research papers from academia.

2.2.1 Streaming architecture

Since data stream is an infinite sequence of data objects (or unbounded data) arriving in random order, it is impossible to apply many standard operations and queries such as aggregations, sorting, reduce functions, etc. In fact, there is a special kind of queries designed especially for streams, which run continuously over a time and incrementally return new results as items arrive. They are known as long-running, continuous, standing and persistent queries [18] , [19]. However, even they do not provide queries which cannot be done incrementally e.g. sorting. The only way to achieve these kind of operations on data stream is to cut it into chunks of finite size and then work with them independently. Method of cutting a data stream into smaller chunks is called windowing. According to common terminology in academia

[20], [21] there are three main types of window models i.e. temporal spans:

- **Fixed window model**. In this model data stream is sliced up into equal data segments using timestamps. Timestamps can be emitted based on event time (actual time, when the data item was created) or processing time (time, when the system observes the item for the first time). In the ideal world these times are equal. Figure 4 illustrates the difference. Besides timestamps slicing can be done based on a simple count i.e. tuple-based windowing, e.g., after every ten data items. The special case of a fixed window model is a landmark window model, where each time we cut a stream from the beginning until now.

- **Sliding window model**. Generalization of fixed windows. The data stream is divided in the same way as fixes model by a constant length or a fixed period. The only difference is that the period is less than window length, thus there is a window overlap on each iteration. This approach is useful for different kinds of optimizations. For example, if we are counting a sum for sliding window chunks we can re-use already calculated value for an overlapping part, which belongs to both chunks: previous and current.

- **Damped window model**. In this model the system assigns weighted coefficients to sliding windows, i.e. latest windows are more important than previous ones. Usually the weight of data decreases exponentially over the time.

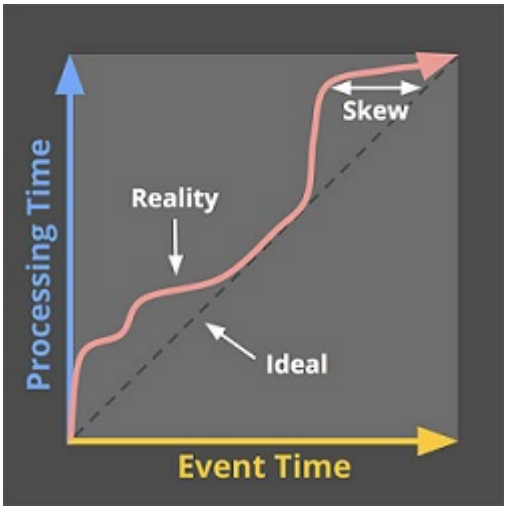


Figure 4: **Event time vs. processing time.** The X-axis represents event time completeness in the system, i.e. the time X in event time up to which all data with event times less than X have been observed. The Y-axis represents the progress of processing time, i.e. normal clock time as observed by the data processing system as it executes. Source: [17]

These three window models were introduced in early 2000s, when internet and web applications just started to grow. Fourth window model has become very popular recently, especially in a context of modern web applications:

- **Session window model**. Dynamic type of windows, which composed of events sequence terminated by a gap of inactivity greater than some threshold. Nowadays, sessions are very popular to analyze online user behavior. Sessions are very powerful, because their length cannot be defined beforehand and heavily depend on data source.

Figure 5 illustrates different window techniques. In most streaming real life scenarios data is pre-grouped by a key (some common parameter, which allows to subset smaller data stream from the main one). For example, in session window model key is usually a user id.

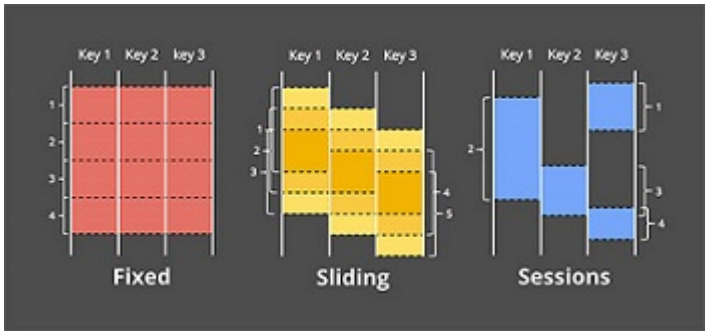


Figure 5: **Different window models.** Each example is shown for three different keys, highlighting the difference between aligned windows (which apply across all the data) and unaligned windows (which apply across a subset of the data). Source: [17]

2.2.2 Streaming clustering algorithms

Silva et al. did an outstanding job in 2013 by creating a survey for streaming clustering algorithms [10]. Since then the state of the art situation regarding clustering algorithms almost did not change. In their survey authors investigated 13 different algorithms regarding next 7 aspects:

1. Data structure for the statistical summary.
2. Window model.
3. Outlier detection mechanism.
4. Number of user-defined parameters.
5. Base offline clustering algorithm.
6. Cluster shape.
7. Type of clustering problem.

This taxonomy covers most of clustering problems, which were discussed in previous parts

Table 1 from [10] perfectly summarizes features of different streaming algorithms. Therefore, we can choose the right algorithm for solving our thesis problem by rejecting some algorithms based on particular features. For example, all algorithms based on K-means will not satisfy our case, as they require the number of cluster as an input parameter. Basically working with spatio-temporal data we are interested mostly in density-based algorithms and can focus only on D-Stream and DenStream for a deeper analysis.

Algorithm	Data Structure	Window model	Outlier Detection	Input Params	Base Algorithm	Cluster Shape	Cluster Problem
BIRCH	feature vector	landmark	density-based	5	k-means	hyper-sphere	object
CluStream	feature vector	landmark	statistical-based	9	k-means	hyper-sphere	object
ClusTree	feature vector	damped	—	3	k-means / DBSCAN	arbitrary	object
D-Stream	grid	damped	density-based	5	DBSCAN	arbitrary	object
DenStream	feature vector	damped	density-based	4	DBSCAN	arbitrary	object
DGClus	grid	landmark	—	5	k-means	hyper-sphere	attribute
ODAC	correlation matrix	landmark	—	3	hierarchical clustering	hyper-ellipsis	attribute
Scalable k-means	feature vector	landmark	—	5	k-means	hyper-sphere	object
Single-pass k-means	feature vector	landmark	—	2	k-means	hyper-sphere	object
Stream	prototype array	landmark	—	3	k-median	hyper-sphere	object
Stream LSearch	prototype array	landmark	—	2	k-median	hyper-sphere	object
Stream KM++	coreset tree	landmark	—	3	k-means	hyper-sphere	object
SW Clustering	feature vector	landmark	—	5	k-means	hyper-sphere	object

Table 1: Analysis of 13 Data Stream Clustering Algorithms. Source: [10]

In both cases authors of algorithms were solving the same problem: discovering clusters over an evolving data stream. This led to next constrains:

- No prior knowledge of the amount of clusters. Moreover, with a stream archi-

tecture the amount of clusters is often changing.

- Detected clusters should be an arbitrary shape.
- Algorithm should handle outliers well, as streaming context assumes much noise.

Having the same goal, two algorithms were designed using different approaches.

2.2.2.1 DenStream

DenStream [22] or Density-Based Clustering over an Evolving Data Stream with Noise was introduced in 2006 by Cao, Feng, et al. Being inspired by another fundamental streaming clustering algorithm CluStream [23] it solved couple of major disadvantaged which were present in its' predecessor: bad handling of outliers, because of adopted distance measurement together with poor performance and limitations because of keeping fixed and pre-defined number of micro-clusters in memory.

DenStream utilized one important novelty of using damped window model, where the weight of each data point decreases exponentially over a time t according to fading function $f(x) = 2^{-\lambda * t}$, where $\lambda > 0$. This formula is widely used in temporal application. The higher value of λ the lower importance of historical data.

DenStream operates in two phases: online part of micro-cluster maintenance and offline part of generating final clusters on demand by the user.

Online part. Algorithm constantly maintains groups of core-micro-clusters, potential-micro-clusters and outlier-micro-clusters, these clusters can be maintained incrementally, as according to authors' evaluation majority of incoming data points belong to one of already existing clusters. Each of this micro-clusters have properties of a radius r and a center c. Also each cluster has a weight w, which is calculated as a sum of weights of all point, which are located within radius r to the center c using Euclidean distance. The difference if the micro-cluster is core, potential or outlier depends on a integer μ. If the $w \geq \mu$ cluster if core. If the sum $w \geq \beta * \mu$, where β is a coefficient such as $0 < \beta \leq 1$, micro-cluster is considered potential. Otherwise, if the weight $w < \beta * \mu$ micro-cluster is considered outlier.

At each arrival of a new point, algorithm firstly tries to insert it into a nearest potential-micro-cluster. If the distance from the potential-cluster center to a new point is lower or equal some parameter ϵ algorithms inserts the point, otherwise it tries to insert it to a nearest outlier-micro-cluster following the same distance check. After that algorithms check if a new value of w exceeded parameter $\beta * \mu$. If so, it means that outlier–micro-cluster has grown a potential. In case no existing clusters can accept new point, a new outlier-micro-cluster created with a center of new point. As the weight of each cluster decreases over the time. The algorithm need to perform the status-check of all clusters and remove old clusters.

Besides basic principles authors use special kind of approximations and calculations to guarantee that the number of outlier-micro-cluster is finite together with other checks for convergence. The online part is initialized by running DBSCAN on a first amount of points N to initialize first potential and outlier micro-clusters.

Offline part. The online part allows the system to maintain dense areas of a data stream, however to achieve real meaningful clusters authors uses variation of DBSCAN, which can be run on request. They use the notion of *density-reachability* similar to the original DBSCAN in order to merge potential and core micro-clusters into full clusters.

In general, algorithm has demonstrated good results during the evaluation with different input parameters.

2.2.2.2 D-Stream

Almost at the same time, in 2007, Chen, Yixin, and Li Tu have introduced their own solution to a given problem: algorithm called D-Stream [24], which was also presented as an alternative and a successor of CluStream. D-Stream is also using density-based approach. It uses two-phase approach similar to CluStream and DenStream: online phase and offline. Another important feature is damped window model, very similar to DenStream. In this way there is no need to keep track of old data points, as the weight or density coefficient of each point is calculated according to the formula:

$$D(x,t) = \lambda^{t-T(x)} = \lambda^{t-t_c},$$

where $\lambda \in (0,1)$ is a constant called the decay factor and t and t_c is current time (processing time) and event time. As the current time is constantly changing there is a need to update density coefficient of each data point every second, however according to authors it is not necessary, as due to the grid structure we work only with grid cells, not with data points. Underplaying grid consists of some fix amount of grid cells, such as every arriving data point is mapped to a corresponding cell. Figure 6 illustrates the process of the algorithm. Every grid cell has among others a property called cell density, which is a sum of density coefficients of every data point mapped to particular cell:

$$D(g,t) = \sum (D(x,t)$$

Therefore, the entire cell grid density can be updated upon a new point arrival according to the following formula:

$$D(g,t_n) = \lambda^{t_n - t_l} * D(g,t_l) + 1,$$

where t_n is a new point processing time and t_l is a last update of grid cell g such as $t_n > t_l$.

Online part. Each data point is mapped to a corresponding grid cell g on arrival. Every grid cell has a feature vector maintaining next information: t_l - time of the last update, D - density coefficient, $status = \{\text{SPORADIC, NORMAL}\}$ is a label used for removing sporadic grids and a *type*. Each grid cell can be one of three types: *dense*, *sparse* and *transition*. Grid cell is considered as dense when density coefficient $D(g) \geq \frac{C_m}{N(1-\lambda)}$, where $C_m > 1$, a special parameter controlling the threshold and N - amount of grid cells. *Sparse* grid cell is a cell with density

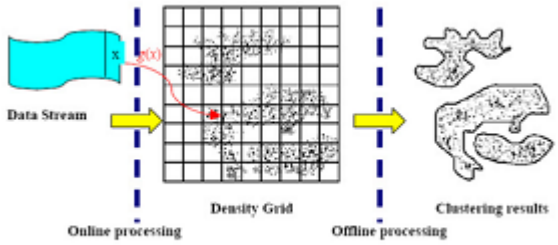


Figure 6: DStream workflow. Source: [24]

coefficient $D(g) \leq \frac{C_l}{N(1-\lambda)}$, where $0 < C_l < 1$, an input parameter. *Transition grid* is a grid, which density coefficient is in between values of being a dense or a sparse grid. Also grid cell can be *outside* or *inside*, meaning if it surrounded by neighbor cells from all sides or not.

Offline part. While online part is responsible for processing incoming data points and maintaining summarized information about grid cells, the actual clustering is done by running a grid inspection process with some time *gap* interval. During the inspection run algorithm checks the status of each grid cell assigning a special status (SPORADIC or NORMAL) to a cell and updates a *grid list* - a special hash table, which contain all NORMAL grid cells and SPORADIC. In this way, for the actual clustering algorithm does not need to check whole grid cell, but only that are present in a *grid list*. After each update of *grid list* the clustering procedure, which is similar to the standard method, is called. Neighboring *inside dense cells* together with *outside transition cells* form a cluster. The algorithm demonstrated very good results during the evaluation and has been widely adopted by academia.

2.2.3 Streaming summary

Working with data streams is not trivial and many traditional methods often cannot be applied. Manipulating data streams requires a special set of tools and methods in order to achieve meaningful results. Clustering over a data stream is even more difficult, as data arrives continuously and clusters are changing constantly. Therefore, there is no ultimate solution, which solves this issue. For our thesis goal we have discussed two, in our opinion, most suitable algorithms. However, none of them fully satisfies thesis needs.

DenStream is very good density-based streaming algorithm, but has a couple of disadvantages. Firstly, it is hard to parallelize as a feature vector of micro-clusters should be a solid data structure. Secondly, it is not completely online. Actual clustering is happening by a user request using DBSCAN-like algorithm.

D-Stream is more suitable for our purpose, as it uses grid structure, therefore it is possible to transform it into distributed algorithm. However, it comes with a cost as we will see in a next chapter. Also damped window model is not the best for our case. The discussion on it will be in the next chapter. In general, we use D-Stream as a base algorithm for this thesis, but modify it quite much.

2.3 Distributed and Parallel processing

All discussed clustering algorithms require at least one full pass at the best case scenario through all data points in order to converge, therefore parallel clustering is not a trivial task and not that many successful implementations of clustering algorithms exist. And indeed, if we look at algorithms like DenStream or DStream or any other algorithms, which act in iterative manner they all in one way or another require a single thread at some point during the execution and many of them need to maintain a common state, which is not always possible in distributed environments.

The difference between parallel and distributed computing is rather vague and up for arguing, however for this discussion we will stick to traditional definitions, where parallel computing - is a type of computation in which many calculations or the execution of processes are carried out simultaneously [25] and distributed computing - is a type of computing carried out in a distributed system, which is the one where components located at networked computers communicate and coordinate their actions only by passing messages [26]. In other words, the main difference between parallel and distributed computing is a memory state. Parallel processes have access to shares memory, while in distributed system every process has their own memory instance. Figure 7 illustrates the difference.

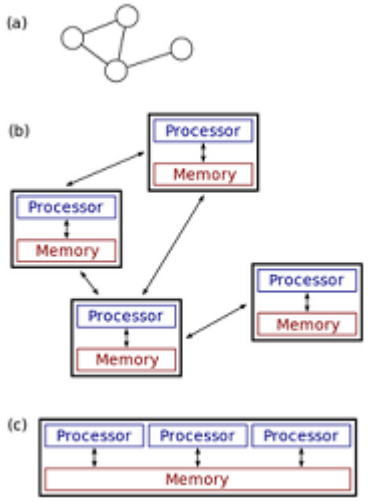


Figure 7: (a), (b): a distributed system. (c): a parallel system. Source: [27]

Parallel computing was always very hot topic in computer science and some parallel clustering algorithms were developed long time ago [28]. However, starting from mid-2000s, after the MapReduce framework [3] was introduced and became de-facto the standard for BigData processing, data mining area received a new wave of algorithms fitting new distributed paradigm.

Similar to streaming, extensions for almost all popular clustering methods were developed for distributed environments over the last decade. However, before discussing the actual algorithms, it is important to underline main challenges in parallel clustering as well as different kinds of parallelism.

According to Talia [29] main goals of the use of parallel computing technologies in the data mining field are:

- Performance improvements of existing techniques.
- Implementation of new (parallel) techniques and algorithms.
- Concurrent analysis using different data mining techniques in parallel and result integration to get a better model (which is more accurate).

And there are three parallelism strategies, that can be applied:

1. **Independent parallelism**. In this way processes are executed in parallel in an independent manner, so as each process has access to the whole dataset and they do not communicate with each other.
2. **Task parallelism**. Each process executes different operations on (a different partition of) the data set.
3. **SPMD parallelism**. SPMD (Single Program Multiple Data) parallelism where multiple processors execute the same algorithm on different subsets and exchange the partial results to cooperate each other.

These strategies are not necessary the only ones to be used in parallel data mining, moreover often they can be combined in so-called hybrid approaches in order to improve performance and accuracy of results. However, majority of parallel clustering algorithms follow the combination of tasks and SPMD parallelism with master-slave architecture [30].

From the very beginning many attempts to parallelize DBSCAN have been made. Many extensions similar to [31] adopt the master-slave model, when data is equally distributed among slaves. Clusters are computed locally at slaves and sent to a master node for a sequential merging in the end. This approach creates communication overhead and very low parallel efficiency during the merging phase.

Some of more recent extensions such as PDSDBSCAN [32] exploit disjoint-set data structure to break the access sequentially of the original DBSCAN. Figure 8 illustrates the algorithm. Singleton tree (single node tree) for each point of the dataset is created at the beginning. Merging is done hierarchically, intermediate trees created after exploring the eps-neighborhood for randomly selected points. After going one step (leaf) higher algorithm terminates. Flexibility and scalability of merging is possible due to main operations of Disjoint-Set Data Structure: *FIND* and *UNION*. Authors managed to obtain good evaluation results for both: parallel and distributed variations of PDSDBSCAN.

Particular interest for current thesis represent density-based clustering algorithms for a MapReduce framework. MR-DBSCAN [33] is one of the most popular extensions. The algorithm is done in a 4-stages MapReduce environment, meaning that there is a sequence of 4 MapReduce jobs required to be executed in order to converge. Figure 9 illustrates them. At the first step MR-DBSCAN summarises the data statistic such

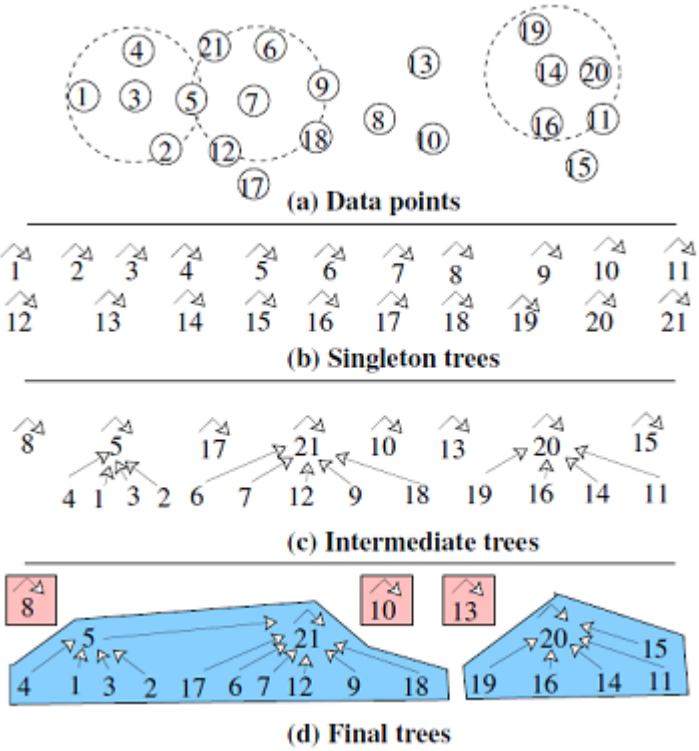


Figure 8: An example showing the PDSDBSCAN algorithm at different stages. Source: [32]

as spatial distribution, amount of records and creates an index. At the second step local distributed clustering is performed following the logic of base parallel algorithm [34]. During a third step bordering subspaces are being discovered using some of topology information from the first step in order to be merged. At the last step merging of bordering areas is done together with formatting data for an output.

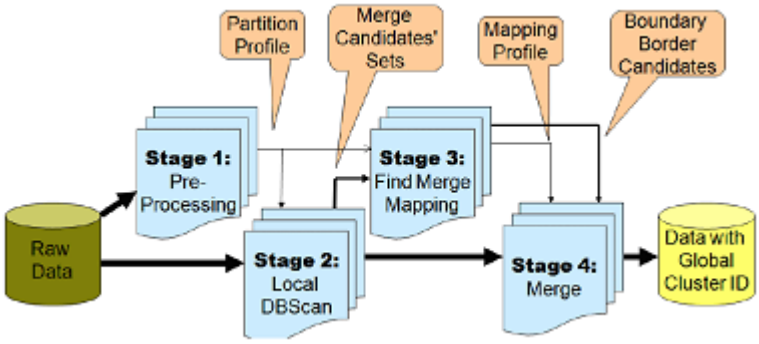


Figure 9: The process and dataflow of MR-DBSCAN. Source: [33]

pGrid: Parallel grid-based data stream clustering with MapReduce [35] is an interesting extension. It was developed for a streaming data on top on already discussed D-Stream. Algorithm follows two-phase architecture design, mapping

arriving data object to a base grid system but instead of doing merging sequentially as in original version, pGrid utilises MapReduce framework merging grid cells by each dimension and in the end combining overlapping grid cells. Figure 10 provides algorithm description for a 2-dimensional space.

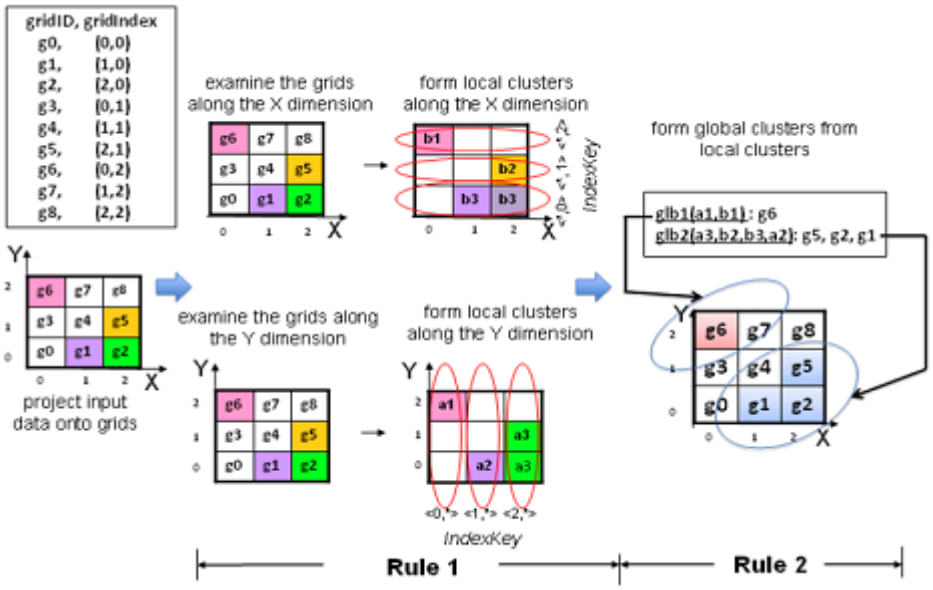


Figure 10: A 2-D example of the MapReduce clustering. Source: [35]

2.4 Cluster analysis

Discovering clusters is an important step, however modern applications require often not only to detect clusters but also to analyze them and provide some useful insights. Especially this is true at a streaming environment, where clusters are dynamic and change with the time. Detection of those changes is a difficult task and can be approached from different angles.

2.4.1 Cluster similarity

The first and the most obvious task in cluster analysis is a cluster similarity task, i.e. analysing how two objects are related between each other. Similarity is most often measured with the help of a distance function. The smaller the distance, the more similar the data objects.

Probably three most often applied distance measure functions are Euclidean, Cosine and Jaccard distances. As a cluster is just a set of objects, the one can treat these objects differently and get different results. The choice of a particular distance measure mostly depends on domain logic of an application.

Probably the most popular measure is Euclidean distance, where we treat clusters

as sets of individual points:

$$d([x_1, x_2, ..., x_n], [y_1, y_2, ..., y_n]) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

With a Cosine similarity clusters can be evaluated as vectors of objects:

$$d_{Cosine} = \frac{x * y}{\sqrt{\sum_i x_i^2 \sum_i y_i^2}}$$

Finally, with a Jaccard distance between two sets can be compared as "bags of objects":

$$d_{jaccard}(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

Jaccard distance which comes from Jaccard index, also known as Jaccard similarity coefficient, is a very simple but yet powerful statistic for comparing two sets of objects. The formula of a coefficient itself looks next:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad \text{where} \quad 0 \leq J(A, B) \geq 1$$

J(A,B) equals one means that sets are completely similar and equals zero that they are diverse. The idea is nicely presented with Venn diagrams at Figure 11.

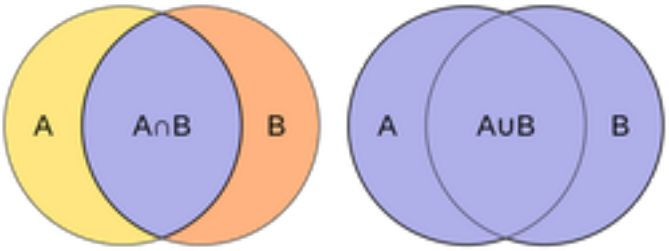


Figure 11: Intersection and union of two sets A and B. Source: [36]

The variation of a Jaccard similarity measure is Szymkiewicz-Simpson coefficient or also known as overlap coefficient [37]. It allows to identify if one set is a subset of another set. In order to calculate it, one need to divide an intersection size of two sets by the smaller size set of both:

$$overlap(A, B) = \frac{|A \cap B|}{min(|A|, |B|)}$$

2.4.2 Pattern detection

Analyzing cluster similarity is a relatively easy task. More challenging and sophisticated goal is to analyze evolving clusters according to a time dimension. By default,

in a dynamic streaming environment there is no prior knowledge about cluster creation and any dependencies. Clusters may appear and disappear in random order and random places. The task of streaming cluster analysis is to find some kind of patterns and dependencies in series of cluster discoveries.

The huge research topic is discovering and analyzing relative motion patterns in groups of moving point objects. This topic is not directly linked to clustering, but in our case notions of "group of moving points" and "moving clusters" are quite similar. The only difference is that moving point objects (MPOs) are represented as geospatial lifelines: a series of observations consisting of a triple of id, location and time [38] and clusters presented as single objects. Moving point objects are a frequent representation for a wide range of phenomena: from animal migrations to traffic analysis systems and even soccer player movement analysis. Understanding motion processes is an important precondition for the design of location based services.

Laube together with Imfeld and Weibel [39] has defined next popular pattern, that can be discovered within set of MPOs:

- **Flock.** This pattern describes group of objects, that are close to each other and moving in the same direction (i.e. moving clusters).
- **Leadership.** This pattern is very similar to the flock one. The only difference is that one object should be heading in a particular direction for some time.
- **Convergence.** This pattern describes group of objects, moving to the same location.
- **Encounter.** This pattern is a special type of Convergence, when group of objects reach the same location at the same time.

The authors also suggested efficient algorithms to discover these patterns, however without any particular clustering technique and working directly with MPOs, which are stored as a simple matrix.

Later Gudmundsson, van Kreveld and Speckmann have reviewed Laubes' work and improved convergence time complexity for the encounter pattern [40]. Gudmundsson together with Benkert [41] have also dedicated a separate research just to detecting flock patterns. Figure 12 presents the notion of a flock discovery.

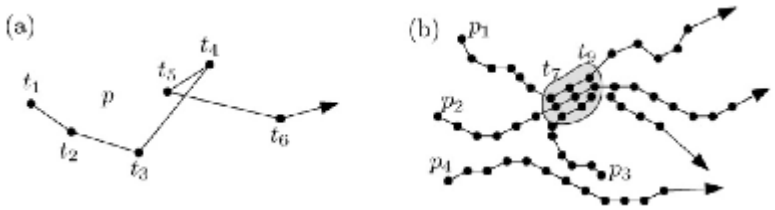


Figure 12: (a) A polygonal line describing the movement of an entity p in the time interval $[t_1, t_6]$. (b) A flock for p_1 , $_2$, $_3$ in the time interval $[t_7, t_9]$. Source: [41]

Yanwei et al.[42] extended the idea of a flock reporting to the trajectory online clustering using density-based clustering algorithm as a base method. Figure 13

demonstrates that clustering trajectories can be seen as an extension of a flock pattern detection.

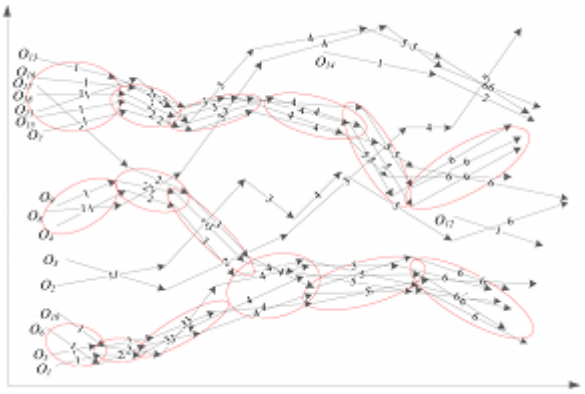


Figure 13: An example of line segment clusters and trajectory clusters. Each line segment with an arrow represents a trajectory line segment of the moving object, and the number on the line segment indicates the time interval of the line segment. Source: [42]

Kalnis et al. studied the discovery of moving clusters in a database of object trajectories [43]. The key difference of clustering trajectories or movement patterns and discovering moving clusters is that the identity of a moving cluster remains unchanged while its location and content may change over time. Presented algorithms utilize DBSCAN as a clustering method. Authors also provided a definition of a moving cluster, which we will follow in this master thesis. Figure 14 provides a visual definition of this concept. Authors used already discussed Jaccard coefficient to analyze cluster similarity. In their paper authors use value of $\theta = 0.5$.

2.5 Complex event processing

Complex event processing (CEP) is an emerging technology, which allows the system to track and analyze flows of information in forms of data streams using event-driven approach [44]. This concept was originally developed from a need of analyzing and processing streaming information in real-time without storing data to the database first. For example, the system is responsible for detecting fire in a building using smoke and temperature sensors. The alarm should be triggered as soon as possible from one hand, from the other hand, there is no need to store updates from sensors, if they are not related to a fire alarm. Such examples can be found in almost any domain area: from finance sector and stock markets to IoT area and monitoring different kinds of environments.

The core notion in CEP is an event. An event is an occurrence within a particular system or domain; it is something that has happened, or is contemplated as having happened in that domain. The word event is also used to mean a programming entity that represents such an occurrence in a computing system [44]. Basically any change of state in a particular system can be considered as an event. The word "complex"

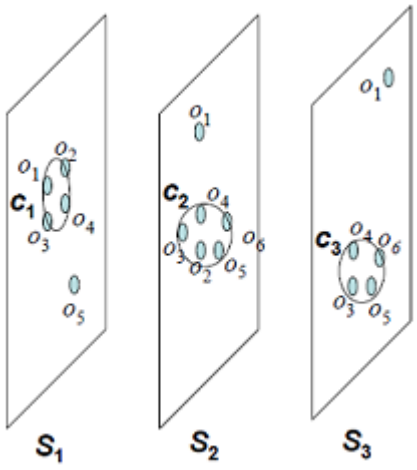


Figure 14: S_1, S_2, S_3 - snapshots of DBSCAN results. C_1, C_2, C_3 - discovered clusters at each snapshot at some timestamps. C_i, C_{i+1} is a moving cluster if $\frac{(C_i \cap C_{i+1})}{(C_i \cup C_{i+1})} \geq \theta$ where $0 \leq \theta \geq 1$. Source: [43]

in CEP means that the system can aggregate new events based on some basic events using specific set of rules and patterns, therefore there is a next terminology in terms of CEP:

- **Atomic Event**. A primitive (basic) event, which represents the smallest entity in terms of an information flow. Atomic events can be defined as instantaneous (happening in a specific point of time), significant (related to a domain context), indivisible (cannot be decomposed into smaller entities). Usually provided outside of the system and seen as a ground truth.

- **Complex Event**. Also known as composed or derived event. An event, which is created by a system as a result of event processing. Complex events are based on a sequence of atomic events. The logic or rules for creating such events are described by events patterns.

- **Event Pattern**. A formalized set of rules, which describe complex events inside the system. The system analyzes a stream of atomic events and emits new complex events recognizing predefined patterns. Patterns normally described with the use of formal logic and Boolean operators, such as sequence, conjunction, negation, composition, etc.

For example, Hu Wenhui et al. in their work on applying CEP technology to process high level business logic of RFID (Radio-Frequency IDentification) applications [45] have defined six event operators:

1. **Aggregation:** (A,n) means n instances of event type A occur.

2. **Disjunction:** (A|B) means either A or B occurs (two disjunction events are generated if both occur).

3. **Conjunction:** (A&B) means both A and B occur.
4. **Negation:** ($\tilde{A}$) means no instance of A occurs; this must be qualified by operator Within.
5. **Sequence:** (A;B,t) means A and B both occur, A first, within t time units of each other.
6. **Within:** Within(A,t) means A occurs with constraint that the interval of the instance is less than t time units.

The authors claims that these 6 operators are enough to describe business logic of most RFID applications. Graphically basic operators are presented at Figure 15.

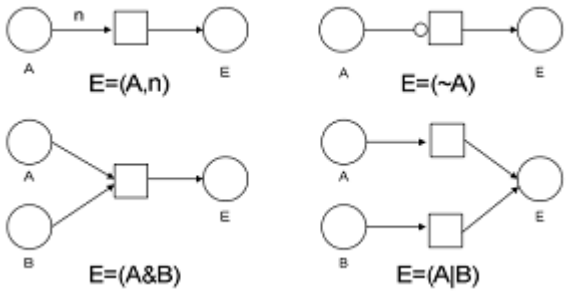


Figure 15: Visualization of Aggregation, Negation, Conjunction and Disjunction. A and B - atomic events. E - complex aggregated events. Source: [45]

The main advantage of CEP architecture is that the complexity of events basically remains unlimited. New event patters can be described based on previous simple patterns, producing new complex events in a system. Figure 16 illustrates such pattern.

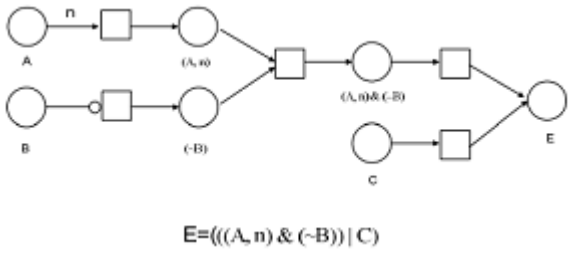


Figure 16: Example of a more complex composite event. Source: [45]

In general CEP provides very powerful and flexible way to analyze data streams. We will use its apparatus for a cluster analysis.

2.6 Location Based Services

This thesis is developed as a part of location based services. LBS have a rich history of research and take their roots from more than 20 years ago, when the GSM

communication went global and there occurred a need for detecting a mobile phone location inside the mobile network. In the academia LBS are often seen as a part of a bigger topic: context-aware services. Context-aware services can be defined as services, which dynamically change their behavior based on some specific context information [46]. Therefore, in a case of LBS the context is location. Activity, identity or time can also carry context information. Figure 17 presents the whole picture.

The turning point for the LBS development was over a decade ago, when GPS-capable mobile phones reached the mass market. Since then LBS has developed in a number of different directions:

- **Became proactive.** First LBS were reactive, meaning that the service was always triggered by the user. The example of reactive LBS is a map service (Google Maps), where user explicitly asks for a specific location or a route. Proactive LBS are triggered automatically, whenever a user enters a certain area. Applications like Foursquare are able to send push notifications about interesting places nearby without a user request.

- **Became user-centric.** Initially LBS were seen as services of mobile operators exclusively, where telecom networks were responsible for identifying users' locations and providing extra services. The development of smartphones and wireless mobile telecommunications (3G, LTE) has shifted the focus to the application level, where each application can request an access to a user location and provide context-aware information (e.g. Foursquare). However, the ownership of the data and the full control of which application gets this access belong to the user.

- **Spread to indoor.** Traditional positioning systems such as GPS have proved their efficiency for an outdoor tracking. Despite the fact that the research on indoor positioning technologies started long time ago [47], they remained quite primitive until recently. Due to the development of the network infrastructure and communication protocols (WiFi, Bluetooth) currently there are many indoor navigation solutions available based on fingerprinting [48] and Bluetooth beacons [49].

In general, LBS have shown a great development targeting single users. Using technologies such as geofencing modern systems are able to deliver relevant notifications to users, based not only on the actual location, but also on additional context information e.g. movement patterns [50]. However, LBS remain extremely poor in the area of providing information based on location of multiple objects. Even though multitargeting (ability to track many objects) services existed more than 10 years ago [51], they did not receive a lot of further development from the research community. Partly this is due to the historical user-centric shift, where the user is responsible for sharing its own location. Legal issues are also an obstacle here. Studies have shown that users trust less systems, which "silently" track their locations [52]. As for our knowledge, there is no ultimate solution efficiently utilizing the location information of crowds.

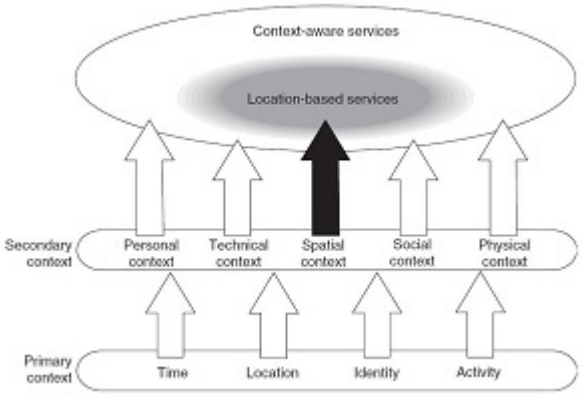


Figure 17: LBS as a part of context-aware services. Source: [46]

Current thesis can be seen as a middleware layer of proactive LBS, as we do not deal with the location extraction.

2.6.1 Geolocation

The main idea of LBS is to provide services based on users' location. The concept of geolocation is responsible for identification or estimation of the real-world geographic location of an object. The most common and popular system of describing objects' location is World Geodetic System (WGS) also known as WGS84, where 84 refers to a 1984 year, when the system was established. Many current technologies including GPS are using this system.

The position in WGS84 standard consists of a three-dimensional Cartesian coordinate system and an associated ellipsoid so that WGS84 positions can be described as either XYZ Cartesian coordinates or latitude, longitude and elevation coordinates [53]. In real world applications the location is usually described just with XY coordinates (latitude and longitude) and accuracy.

Even though WGS84 is very reliable and wide-spread system, sometimes it is not very convenient to work with each objects' location individually and there is a necessity to group them according to their position in some order. In 2007 Gustavo Niemeyer introduced a convenient way of expressing location by using short alphanumeric strings named *Geohash*. Geohash is a string representation of a location. It utilizes a hierarchical spatial data structure which subdivides space into buckets of grid shape. Single geohash identifies a rectangular cell on a surface of Earth. Initially the whole surface is divided into 32 cells, each cell represented by a single character. Each extra character added to a string divides a corresponded cell into 32 sub-cells. This approach allows achieving a particular precision and "zoom in" effect into any point on Earth. Table 2 demonstrates the idea of geohash with different precision levels and size dimensions. Any coordinate pair can be represented with a geohash with a certain precision. Usually nearby locations have similar prefixes, which allows very flexible and powerful manipulations while working with geohases. However, this is not always the case, there are edge-places straddling large-cell boundaries. For

example, in France, La Roche-Chalais (u000) is just 30km from Pomerol (ezzz).

Geohashes are particularly interesting in terms of the current master thesis, because they basically represent global scalable grid structure to encode spatial data.

Table 2: An example of geohashes with string length and size dimensions near the equator

Geohash	Length	Cell Dimensions
g	1	$\sim$ 5,004km x 5,004km
gc	2	$\sim$ 1,251km x 625km
gcp	3	$\sim$ 156km x 156km
gcpu	4	$\sim$ 39km x 19.5km
gcpuu	5	$\sim$ 4.9km x 4.9km
gcpuuz	6	$\sim$ 1.2km x 0.61km
gcpuuz9	7	$\sim$ 152.8m x 152.8m
gcpuuz94	8	$\sim$ 38.2m x 19.1m
gcpuuz94k	9	$\sim$ 4.78m x 4.78m
gcpuuz94kk	10	$\sim$ 1.19m x 0.60m
gcpuuz94kkp	11	$\sim$ 14.9cm x 14.9cm
gcpuuz94kkp5	12	$\sim$ 3.7cm x 1.8cm

3 Concept and Design

The aim of this thesis is to design a system for real-time crowd detection and analysis, using as an informational source a data stream with spatial-temporal data. The system is a natural part of the proactive LBS development. Before going to technical details and architecture, we need to define problems, which this system solves, and questions, which this system answers. Simply speaking we need to answer a question why do we even need this system.

The future system should be able to solve next tasks:

- **Crowd detection**. The system should be able to detect crowds in a continuous data stream.
- **Pattern detection**. The system should be able to detect and analyze interesting patterns and insights over time.

The application area of such system can be very broad: from monitoring traffic situation on roads to optimizing logistics or peoples' flow in places of mass gathering (airports, shopping centers, railway stations, etc.).

Also, the system should satisfy latest technical standard and modern trends, namely:

- **Real-time computing**. This is very critical for current systems, as today users demand the information to be delivered as soon as possible. And not only delivered, but also analyzed and presented in a nice readable format. The time gap between the data point being emitted and the same data point being processed by the system should be minimal.
- **Scalability**. With an exponential grow of the data it is very critical, that the system can scale and adjust to a huge workload and amount of data. Scaling in Big Data is achieved through distribution and parallelism. This also called scaling out.

In this section we answer how our designed approach solves stated problems and demonstrate how we address technical challenges.

3.1 Big Picture

The designed system consists of two parts:

- **Clustering**. This part is responsible for detecting clusters on a continues data stream. The data point should consist at least of a unique identifier emitting this point, timestamp and position information (latitude, longitude). Any additional information might be useful for extra analysis, but 4 fields are enough to perform clustering operation.

- **Pattern detection**. The second part is responsible for a pattern detection by analyzing found clusters from a previous step. The main criteria here, is that pattern detection should be done also online without storing intermediate results to a database (so called two-phase algorithms).

Figure 18 illustrates a general workflow of the system. The output of the system is a stream of events in terms of CEP terminology. Simply speaking the system triggers a special notification for each detected pattern and emits a new event. These events and patterns are described later in this chapter.

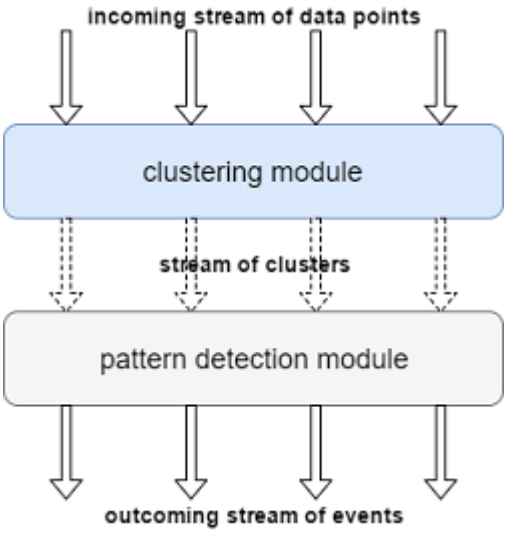


Figure 18: Workflow of the system

3.1.1 Current approaches

Before describing our approach to this problem it is useful to analyze already existing systems and methods.

To our knowledge there are no publicly available applications, which are able to detect clusters on evolving data stream, recognize patterns in them and at the same time satisfy technical demands, such as scalability, distributed processing and online architecture. There are many existing clustering algorithms, most prominent of which were presented in the previous chapter, however none of them satisfy stated needs.

Streaming algorithms such as D-Stream and DenStream are not distributed and require a single pass via all data points at least once, which is a serious bottleneck. In general, algorithms which are using a feature vector or some kind of a tree as an underlying data structure are much less scalable than algorithms using grid system. Also all presented streaming algorithms consists of two parts: online and offline. Online part usually responsible for mapping incoming points to a correspondent data structure and offline part, which is mostly triggered with some time interval, is responsible for clustering itself. Our aim is to design a system, which can operate in a pure streaming manner.

Due to already discussed architecture challenges not many distributed extensions for streaming algorithms exist. Presented in a related work parallel extension of a DStream algorithm pGrid follows MapReduce framework. However, being already outdated traditional MapReduce job should run on top of a database, which leads again to a two-phase approach. Moreover, in case of pGrid three MapReduce jobs are required in a case of the 2D grid. Two jobs for merging grid cells according to X and Y coordinates. And one final job in order to merge overlapping areas. This is not very efficient and MapReduce being performed on stored pre-aggregated data cannot be considered as a completely online method.

As for the second part of the problem - pattern detection, there are very few methods, which analyze moving clusters. Most of the research on a pattern recognition has been conducted for trajectories and not for clusters. Very few research projects try to analyze patterns in a cluster discovery. And those who do, actually remain quite primitive. For example already discussed work on moving clusters [43] can only provide information if the cluster is moving or not. Basically no major research has been conducted in this field. There is also no known classification and even generally accepted terminology on a dynamic cluster discovery and cluster patterns.

3.2 Clustering

Let us declare a single data-point P with the minimum required parameters for the algorithm to function.

$$P(id, timestamp, lat, lon),$$

where id - unique identifier of the object producing the data point, $timestamp$ - event time, when data the point was emitted, lat - latitude in a WGS84 system, lon - longitude an in WGS84 system.

The first step is to map an arriving data point to a corresponding data structure. A grid-based data structure was chosen for our system due to its ability to be processed in parallel. Moreover, since our system is designed only for spatial data, meaning two dimensional grid, we can take full advantage of a natural grid world-wide system: geohashes. This allow our system to operate almost at any point on Earth without any previous setup. Also geohash allows naturally "zoom-in" and "zoom-out" just by changing precision, providing hierarchical order, very similar to STING clustering algorithm. In this case the precision of a geohash can be an input parameter from a user, as for different data sources different cell size might work better.

The first computing step is to calculate a geohash for each incoming point converting an initial data point $P(id, timestamp, lat, lon)$ into $P(id, timestamp, geohash)$, where geohash is a character array with a set precision. After this step we work with a stream, where each point is mapped to a corresponding geohash (grid cell), therefore the following step is to group data points by this geohash. Figure 19 illustrates these two actions.

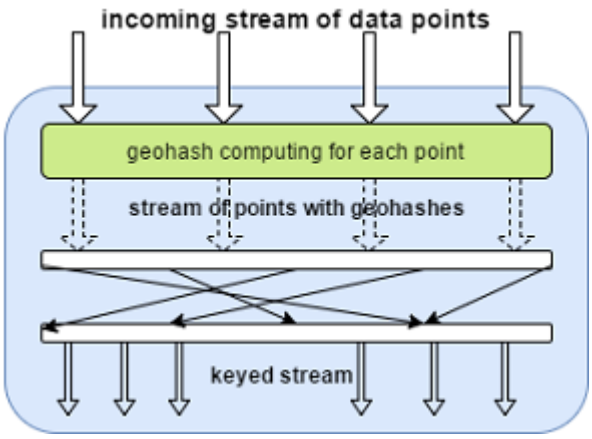


Figure 19: The system calculates a geohash for each data point and group data points into keyed stream

Since the data stream is infinite and non-stopping by default, the only way to analyze data, is to reduce data objects with some time gap, using a windowing technique. Section 3.2.1 provides a discussion about alternatives and possible outcomes. Our approach uses fixed window model with simple count technique of a density coefficient. Fixed window model is used in many streaming algorithms, for example BIRCH. Count technique of calculating a density threshold is used in CLIQUE. The aggregated Table 3 summarizes different ideas implemented in our method, which were borrowed from other prominent algorithms.

Table 3: Key features of a suggested algorithm together with idea sources

Feature	Implementation	Source
Grid data structure	Geohash system	STRING, D-Stream
Density calculation	Object count	CLIQUE
Window model	Fixed	BIRCH
Cluster shape	Arbitrary*	D-stream
* limited to a single grid cell shape		

The time gap for "cutting" a continues data stream according to a fixed model is another input parameter and can be changed to fit better a particular data source. After grouping a data stream by a geohash the algorithm performs reduce operation with a time interval T. The discussion on a time interval is presented in the Section 3.2.1. The outcome of the reduce operation is a stream of grid cells G , where each grid cell has a following format:

$$G(geohash, List < P >, timestamp),$$

where $geohash$ - name of a geohash, by which points where grouped previously, $List < P >$ - array of data points belonging to this grid cell, $timestamp$ - the timestamp of the latest point.

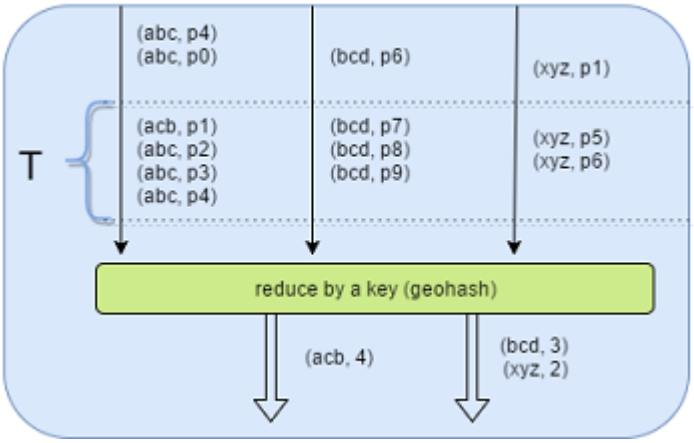


Figure 20: Reduce function groups all data points with the same geohash within time interval T into a grid cell

The reduce process in details is presented in the Figure 20 and a whole process is in the Figure 21. After the reduce operation the output stream consists of grid cell objects G, where at each grid cell contains all data points, which were detected in this area in a previous interval T.

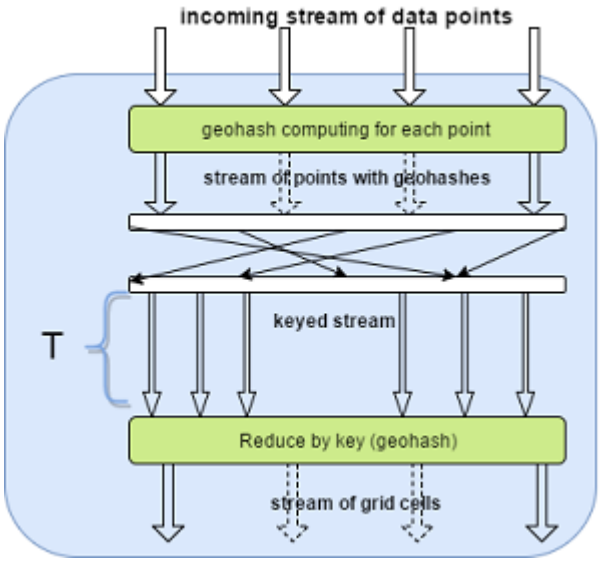


Figure 21: Algorithm pwrform a reduce operation over the keyed stream with a time interval T

Another user input parameter to the system, is a density threshold D, which is responsible for considering grid cells dense or sparse. If the amount of objects P in G more or equal than D, grid cell is considered dense, otherwise it considered sparse and the algorithms filters it out. Figure 22 illustrates this step with example. Therefore, at the next step we keep only dense grid cells.

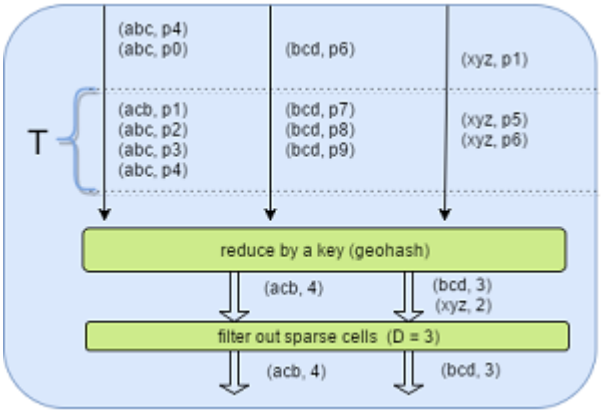


Figure 22: After the reduce operation algorithm filters out grid cells G, where $\sum P < D$. In our example $D = 3$

After filtering out sparse grid cells, the algorithm performs merging operation, to create cluster out of neighboring dense grid cells. This can be done in two ways: sequentially, simply by iterating through all dense grids in a data stream and recursively, by "zooming up" to a higher level of geohashes. The discussion on this issue is presented in the Section 3.7. In the initial version of the algorithm we perform merging sequentially with the maximum time complexity of $O(n^2)$, where n is amount of grid cells after filtering. The algorithm uses the same value of a time interval T, in order to be consistent and not to create a bottleneck. The output of a merging function is a list of found clusters C:

$$C(id, List < G >, gN, pN),$$

where id - is a unique identifier of a cluster; $List < G >$ - array of dense grid cells, which form the cluster, gN - number of grid cells in a cluster, pN - amount of data points, forming the cluster.

At this point the clustering procedure is considered finished. The overall delay from receiving the last data point to creating clusters is $2 * T$. The whole process is presented in the Figure 29.

3.2.1 Discussion on a Window Model

There are two window models, which are widely adopted for a streaming clustering: fixed and damped. The difference is that with fixed window model stream is divided in equal snapshots according to some time interval parameter (sometimes count) and with the damped window model each point is assigned a timestamp-based coefficient, which decreases over time. Each of these models has advantages and disadvantages. Here we evaluate both models in terms of a current thesis domain area.

However, let us firstly describe the data stream and some design challenges related to moving objects. Let assume that the window time interval parameter T_i equals 5 minutes (300 seconds) and T is a time of the latest taken snapshot. We have two points P_1 and P_2 arriving at the time $T_1 = T - 240_{seconds}$ and $T_2 = T - 10_{seconds}$

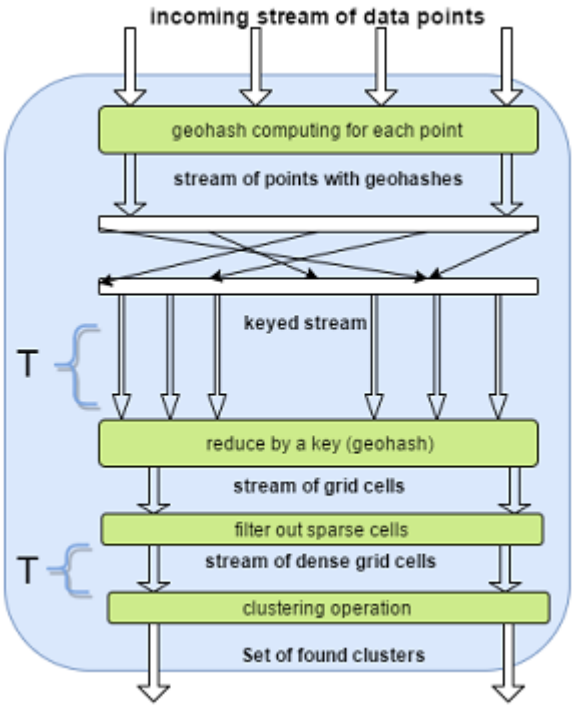


Figure 23: The overall process of clustering

accordingly. This looks alright, if we know that both points remained unchanged since the last update. However, in a dynamic environment the assumption is that each object is constantly moving. Therefore, there is a possible scenario, that point P_1 has significantly changed its' location during the time difference $T - T_1$ or 240 seconds as in our example. This might lead to the inconsistency of a false positive count, where the algorithms will consider the data point mapped to a particular grid cell, even though the actual location of a point P_1 at the time T will be different. The problem, is that in most cases we receive location updates from objects with some sampling interval T_S, moreover usually this interval is not a constant and in real world might vary a lot. For example, in case of vehicles GPS signal might be blocked by different obstacles (tunnels, high buildings, etc) or might arrive with some latency delay due to network issues. The optimal scenario is when $T_i = T_S$, however this is very unlikely. With the grow of source data objects, the delta of T_S will grow as well. Here two scenarios are possible: $T_i < T_S$ and $T_i > T_S$. In case of $T_i < T_S$ we will have a situation, where not all data points were captured by a single snapshot and clustering results might be damaged by a data fragmentation over time, meaning that a part of data will be clustered in one snapshot and another part in another snapshot. In case of $T_i > T_S$, the situation is that some moving objects can update its location multiple times during the snapshot, which leads to a double count of some points. This is not a big issue if there is a global state providing guarantees that each point is presented only once in a snapshot with the latest timestamp taking into consideration. During the development of a current thesis, we came to a conclusion that providing such global guarantees is very difficult in a distributed streaming

environment and will definitely come with a cost of performance.

In this context, choosing the windowing interval parameter T_i is critical for a successful execution of the algorithm and should heavily depend on data source and features of a data stream. Now let us see how different window models can help to overcome described above challenges.

The fixed window model is more straightforward and easy to work with due to it simplicity. At the time T_i of the latest snapshot the algorithm has access to all data points, which have arrived during the period (T_i, T_{i-1}). Following previously stated example, the weight of the point P_1, which has arrived at the time T_1 is the same as for the point P_2 which has arrived at the time T_2, even though P_1 is relatively old. In a distributed environment depending on T_S parameter we might have duplicate points affecting clustering results and create some data inconsistency. The grid cell density threshold in this case is an integer value, usually a sum of data points belonging to a particular grid cell.

Using a damped window model assumes that each data point has a density coefficient parameter and this parameter loss it's value over time. In this case the density threshold is an aggregated float value, the sum of density coefficients of all points belonging to a particular grid cell. This approach might be more accurate for clustering results, however besides that it does not create any additional value to the algorithm. Also in case there is no prior knowledge if the object is constantly moving or not, damped window model might be misleading. For example, in case of vehicles the object might just be standing in a traffic jam or at the city lights for some time. So technically it should not lose any weight, but with the damped window model it does. Also for the later cluster analysis we only care about if the object belongs to the cluster or not (true or false), having flexible density coefficients of grid cells make the analysis more complex and does not bring any value. In the end, while working with data objects like vehicles or mobile devices (people) we aim to obtain as much precision and correctness as possible. Either the object is there or not.

To sum up, we can state that the fixed window model is more suitable for our domain area. It has some disadvantages such as false positive location counts or double count, but the same disadvantages are true for a damped window model. Plus, damped window model adds some fundamental limitation for a later cluster analysis, as data points belonging to the same entity (grid cell) cannot be treated equally.

3.2.2 Discussion on a merging process

Forming final clusters from some smaller entities (grid cells or micro clusters) is a bottleneck and the least efficient part of all distributed algorithms. In most cases at least one full pass through the whole dataset is required. Here we discuss two possible approaches for merging dense grid cells in our algorithms.

The first one and the obvious one: is a simple loop through all dense grid cells. The worst time complexity with this approach is $O(n^2)$, where n - amount of dense grid cells. The algorithm randomly picks a dense cell and create a cluster out of it. Than through iterating over other dense cells the algorithm checks if a current dense

cell is a neighboring cell to an already created cluster. If yes, the algorithm adds a cell to a cluster, if no, new cluster is created out of a current cell. In the worst case scenario all dense cells will not have dense neighbors and the final amount of cluster will equal n. Algorithm 1 shows this process in details with a pseudocode.

Algorithm 1 Dense cells merging process

Result: Array of clusters
Input : Array of dense cells *grids*
Output : Array of clusters *clusters*

```
initialization of a cluster array clusters
 initialization of a flag

foreach grid cell G in grids do
    flag = false
     if clusters is empty then
        C = cluster from G
         clusters ← C
         continue end
        foreach cluster C in clusters do
            if C is neighbour of G then
                clusters ← C
                 flag = true
                 break end
            end
            if flag ≠ true then
                C = cluster from G
                 clusters ← C
            end
        end
```

Another approach for creating cluster is more elegant in terms of distributed environment. As we already discussed geohash names are designed in a way, that neighboring geohashes most of the times have same prefixes, the closer geohashes geographically to each other, the more characters will overlap in their names. This rule is not strict, as there are some areas (mostly edge cases near poles and also along the Greenwich meridian). Nevertheless, in most parts of the Earth we can group dense cells together by using the same geohash, but with a lower precision. Let us assume that initial clustering is done with a precision 7 (152.9m x 152.4m). After obtaining dense grid cells we can "zoom out" to a precision 5 (4.9km x 4.9km), group initial dense grid cells by first 5 characters in their names and create clusters in parallel within our new areas with a precision 5. There is still a possibility that neighboring grid cells might end up in different merging areas, therefore we still need to execute a sequential check. However, in case of hierarchical technique the amount

of objects to iterate through drops significantly with each geohash precision level.

The merging process is a bottleneck of every distributed clustering algorithm and there is no ultimate solution to solve this problem. In our final design we use straightforward solution presented at the algorithm 1 above, however this process can be optimized by using hierarchical merging. This very much depend on the amount of data and the geographical location.

3.3 Pattern detection

The second part of the algorithm is a pattern detection and knowledge discovery based on previously found clusters. For this task we are using CEP concept and its semantics. CEP satisfies our technical needs:

- **Real-time computing**. One of the core advantages of CEP is that everything is processed in a streaming environment. There is no need for storing intermediate results in a database. Another feature of CEP is that patterns are detected almost instantaneously, which provides almost real-time reaction.
- **Scalability**. By default, CEP is designed to work with multiple atomic event sources, so scalability is in its core. However, this feature is dependent of the actual framework.

The first task in order to implement pattern detection is to define and describe events, patterns and a formal logic of a future system. From a cluster detecting part we receive a set of discovered clusters in a form:

$$C(id, List < G >, gN, pN),$$

where id - is a unique identifier of a cluster; gN - number of grid cells in a cluster; pN - amount of data points, forming the cluster. $List < G >$ - set of dense grid cells, which form the cluster.

The first step is to decompose set of discovered clusters to a stream of events initialing the CEP process. Moreover, as G is also an aggregated value, we decompose it to smaller entities in order to make following mining easier and more flexible. Therefore, the system creates a stream of events called $newClusterEvent$ in a form:

$$newClusterEvent(id, Set < geohash >, gN, Set < objectId >, pN, t),$$

where id - is a unique identifier of an event; $Set < geohash >$ - set of geohash strings with an initial precision (7), they carry the location information of the event; $Set < objectId >$ - set of unique identifiers of objects, forming the cluster; pN - amount of objects, forming the cluster; t - timestamp of a snapshot, when the cluster was formed.

New cluster event $newCLusterEvent$ is considered as an atomic (primitive) event. Basically this is the only atomic event, which we have in the system. Therefore, our task is to describe set of rules to derive new events in order to extract some meaningful knowledge. Unfortunately, we cannot just apply CEP operators presented

in the Section 2.5, as they all require at least two types of atomic events (A and B in the example). Therefore, firstly let us describe what we strive to discover in a free language and then formalize it with CEP operators.

3.3.1 Basic definitions

The pattern detection system is receiving new batch of discovered clusters every time interval T_i. Every cluster has its own unique identifier. There is no previous knowledge or any additional information about the past. The first challenge is to detect if the cluster is new or repeating (the system has detected it before). Let us defined repeating cluster with the help of already discussed Jaccard similarity measure.

Cluster C is a repeating cluster, if the Jaccard coefficient of two new clusters is greater or equal than a coefficient θ, which can be a user input parameter. For example, in already discussed concept of moving clusters presented in the Section 14, authors use $\theta = 0.5$. This means that at least half of objects forming both compared clusters should have the same id.

Definition 3.1. Repeating cluster. Cluster $C_1(id, Set < geohash >, gN, Set < objectId >, pN, t)$ and $C_2(id, Set < geohash >, gN, Set < objectId >, pN, t)$ is a repeating cluster, if

$$J_{id} = \frac{(Set < objectId >_{C_1} \cap Set < objectId >_{C_2})}{(Set < objectId >_{C_1} \cup Set < objectId >_{C_2})} \geq \theta, \text{ where } t_{C_1} \neq t_{C_2}$$

The next step is to detect a moving cluster. Location of any cluster C is defined by a set of geohash strings, where every string represents a partial location of a cluster, therefore we can apply the same Jaccard coefficient measure to compare these strings. If strings are the same, the coefficient will be equal to one, which would mean the position is exactly the same. If the coefficient equals to zero, the position is different. In our system we use the value of $\theta = 0.5$, which means that we consider the location the same, if half of the underplaying grid cells are the same. This allows us to neglect some minor shifts, which could be triggered by noise data or missing updates from some objects. Finally, we introduce time gap parameter T_g, within which two clusters can be detected. This parameter is domain dependent and can vary based on the data stream origin. For example, in mining car movements we can set it to half an hour, so if the system discovers repeating cluster within half an hour, the system considers it a moving cluster, otherwise - it is just a repeating cluster. From this simple but yet powerful logic we can derive two definitions.

Definition 3.2. Moving cluster. Cluster $C_1(id, Set < geohash >, gN, Set < objectId >, pN, t)$ and $C_2(id, Set < geohash >, gN, Set < objectId >, pN, t)$ is a moving cluster, if

$$J_{id} = \frac{(Set < objectId >_{C_1} \cap Set < objectId >_{C_2})}{(Set < objectId >_{C_1} \cup Set < objectId >_{C_2})} \geq \theta$$

and

$$J_{geo} = \frac{(Set < geohash >_{C_1} \cap Set < geohash >_{C_2})}{(Set < geohash >_{C_1} \cup Set < geohash >_{C_2})} < \gamma$$

and

$$0 < t_{C_2} - t_{C_1} \leq T_g$$

Definition 3.3. Standing cluster. Cluster $C_1(id, Set < geohash >, gN, Set < objectId >, pN, t)$ and $C_2(id, Set < geohash >, gN, Set < objectId >, pN, t)$ is a standing cluster, if

$$J_{id} = \frac{(Set < objectId >_{C_1} \cap Set < objectId >_{C_2})}{(Set < objectId >_{C_1} \cup Set < objectId >_{C_2})} \geq \theta$$

and

$$J_{geo} = \frac{(Set < geohash >_{C_1} \cap Set < geohash >_{C_2})}{(Set < geohash >_{C_1} \cup Set < geohash >_{C_2})} \geq \gamma$$

and

$$0 < t_{C_2} - t_{C_1} \leq T_g$$

Standing cluster is a very useful pattern. For example, when working with vehicle data we can discover traffic jams or some traffic accidents.

Another interesting pattern, which can be derived from the same Jaccard equation is **hotspot**. When the location is the same, but objects' identifiers are different. In practice this means the geographical location, where clusters often appear. In traffic systems it can be intersections, traffic lights or just some bottlenecks. For this pattern we do not need the time constrain T_g, limiting two events occurrence. For the location detection, actual clusters do not matter, only the place, where they occur.

Definition 3.4. Hotspot. Cluster $C_1(id, Set < geohash >, gN, Set < objectId >, pN, t)$ and $C_2(id, Set < geohash >, gN, Set < objectId >, pN, t)$ define a hotspot, if

$$J_{id} = \frac{(Set < objectId >_{C_1} \cap Set < objectId >_{C_2})}{(Set < objectId >_{C_1} \cup Set < objectId >_{C_2})} < \theta, \text{ where } t_{C_1} \neq t_{C_2}$$

and

$$J_{geo} = \frac{(Set < geohash >_{C_1} \cap Set < geohash >_{C_2})}{(Set < geohash >_{C_1} \cup Set < geohash >_{C_2})} \geq \gamma, \text{ where } t_{C_1} \neq t_{C_2}$$

A couple of simple patterns can be declared with the help of an overlap coefficient. We can easily detect if some smaller cluster has joined a bigger one. There is actually an interesting observation, that if two clusters have the same size, an overlap

coefficient is equal to a Jaccard coefficient. We can utilize this knowledge in order to detect joining clusters. If the overlap coefficient between two clusters is high, let us say $\geq \theta = 0.8$, we know that one cluster is a subset of another. At the same time, when clusters are the same size, the Jaccard coefficient is also equal $\geq \theta = 0.8$. In reality this means that they are very similar and one is always a subset of another. However, if the overlap coefficient is high and Jaccard coefficient is small, we can state that one cluster is a subset of another, however the second cluster one is much bigger. Depending on what cluster is bigger chronologically C_1 or C_2 we can detect joined cluster or separated cluster. In case if a smaller cluster was first such as $t_{C_1} < t_{C_2}$ we consider C_1 a joined cluster. In case $t_{C_1} > t_{C_2}$ we consider C_1 a separated cluster. For these patterns T_g interval is used in order to filter out clusters, which occur far from each other in terms of time.

Until now we did not use location similarity to define these patterns. By adding it we can extend the definition and derive two very powerful patterns, namely *growing cluster pattern* and *shrinking cluster pattern*. If the overlap coefficient of locations is high, for example $\geq \sigma = 0.8$, we can state that one cluster if just joined another and the first one did not move, therefore we can consider it a growing cluster. If the timestamp order is different, such as the smaller cluster occur after the bigger one, we can call it a shrinking cluster. In real life, while analyzing traffic flows, with the help of this pattern we can detect the traffic jam and the cluster, which caused it.

Definition 3.5. Joined & Separated cluster. Cluster $C_1(id, Set < geohash > , gN, Set < objectId > , pN, t)$ is a joined cluster to a cluster $C_2(id, Set < geohash > , gN, Set < objectId > , pN, t)$ if

$$O_{id} = \frac{(Set < objectId >_{C_1} \cap Set < objectId >_{C_2})}{min(|Set < objectId >_{C_1}|, |Set < objectId >_{C_2}|)} \geq \theta$$

and

$$J_{id} = \frac{(Set < objectId >_{C_1} \cap Set < objectId >_{C_2})}{(Set < objectId >_{C_1} \cup Set < objectId >_{C_2})} \leq \gamma$$

and

$$O_{geo} = \frac{(Set < geohash >_{C_1} \cap Set < geohash >_{C_2})}{min(|Set < geohash >_{C_1}|, |Set < geohash >_{C_2}|)} < \sigma$$

and

$$0 < |t_{C_2} - t_{C_1}| \leq T_g$$

Definition 3.6. Growing & Shrinking cluster. Cluster $C_1(id, Set < geohash > , gN, Set < objectId > , pN, t)$ is a joined cluster to a cluster $C_2(id, Set < geohash > , gN, Set < objectId > , pN, t)$ if

$$O_{id} = \frac{(Set < objectId >_{C_1} \cap Set < objectId >_{C_2})}{min(|Set < objectId >_{C_1}|, |Set < objectId >_{C_2}|)} \geq \theta$$

and

$$J_{id} = \frac{(Set < objectId >_{C_1} \cap Set < objectId >_{C_2})}{(Set < objectId >_{C_1} \cup Set < objectId >_{C_2})} \leq \gamma$$

and

$$O_{geo} = \frac{(Set < geohash >_{C_1} \cap Set < geohash >_{C_2})}{min(|Set < geohash >_{C_1}|, |Set < geohash >_{C_2}|)} \geq \sigma$$

and

$$0 < |t_{C_2} - t_{C_1}| \leq T_g$$

These six basic patterns, which can be found by using just Jaccard and Overlap coefficients for objects and geohashes is a basis for a further knowledge discovery. Table 4 summaries them. Parameter values are presented just as an example to provide an approximate scope.

In the related work section four patterns were mentioned, which can be found while mining separate moving objects: flock, leadership, convergence and encounter. While mining clusters we can work only with full entities and not with separate objects, therefore our system can discover only half of them: flock and encounter. Already at this point they are equivalent to our moving cluster and evolving cluster definitions.

Table 4 presents 8 patterns, which were defined. Each of these patterns creates a corresponding event, which can be later used to define complex events.

3.3.2 Advanced patterns

Comparing two *new cluster events* already provides a lot of knowledge, however we can define even more interesting patterns comparing three clusters. In particular, we are interesting in two scenarios: splitting clusters and merging clusters.

In a first case we assume that one big cluster has split into two smaller ones. In a second case we want to find two smaller clusters, which have merged into one big one. These use cases are not as easy to detect as already defined basic patterns, however we expect to detect such behavior while analyzing traffic and people flows.

Merging and Splitting cluster patterns are extensions of joined and separated cluster patterns. The only difference is that instead of looking just for one small cluster, which has joined or has separated, we need to find two small clusters and one big. Depending on timestamps order they can be defined as splitting clusters or merging.

Let us say, we have three clusters A, B and C such as C, A match joined cluster pattern of the Definition 3.6 and B, A also match the same pattern. We can claim that C, B are merging clusters into A, in case Jaccard coefficient $J(C, B) < \theta$, where θ is some small value (≈ 0.2). In this way we ensure, that C and B is not the same cluster. Splitting cluster pattern is exactly the same, just with a different timestamp order.

Table 4: Basic patterns summary. Parameters can be changed depending on particular data. J_{id} - Jaccard similarity between objects. J_{geo} - Jaccard similarity between geohashes. O_{id} - overlap similarity between objects. t_{C1}, t_{C2} - timestamps of first and second clusters in consecutive time

Pattern name	**Conditions**	**Parameters estimation**
Repeating cluster	$J_{id} \geq \theta$	$\theta = 0.5$
Moving cluster	$\begin{cases} J_{id} \geq \theta \\ J_{geo} < \gamma \\ 0 < t_{C_2} - t_{C_1} \leq T_g \end{cases}$	$\theta = 0.5$ $\gamma = 0.5$ $\mathrm{T}_g = 30\ min$
Standing cluster	$\begin{cases} J_{id} \geq \theta \\ J_{geo} \geq \gamma \\ 0 < t_{C_2} - t_{C_1} \leq T_g \end{cases}$	$\theta = 0.5$ $\gamma = 0.5$ $\mathrm{T}_g = 15\ min$
Hotspot	$\begin{cases} J_{id} < \theta \\ J_{geo} \geq \gamma \end{cases}$	$\theta = 0.5$ $\gamma = 0.5$
Joined cluster	$\begin{cases} O_{id} \geq \theta \\ J_{id} \leq \gamma \\ O_{geo} < \sigma \\ 0 < t_{C_2} - t_{C_1} \leq T_g \end{cases}$	$\theta = 0.8$ $\gamma = 0.4$ $\sigma = 0.8$ $\mathrm{T}_g = 30\ min$
Separated cluster	$\begin{cases} O_{id} \geq \theta \\ J_{id} \leq \gamma \\ O_{geo} < \sigma \\ 0 < t_{C_1} - t_{C_2} \leq T_g \end{cases}$	$\theta = 0.8$ $\gamma = 0.4$ $\sigma = 0.8$ $\mathrm{T}_g = 30\ min$
Growing cluster	$\begin{cases} O_{id} \geq \theta \\ J_{id} \leq \gamma \\ O_{geo} \geq \sigma \\ 0 < t_{C_2} - t_{C_1} \leq T_g \end{cases}$	$\theta = 0.8$ $\gamma = 0.4$ $\sigma = 0.8$ $\mathrm{T}_g = 30\ min$
Shrinking cluster	$\begin{cases} O_{id} \geq \theta \\ J_{id} \leq \gamma \\ O_{geo} \geq \sigma \\ 0 < t_{C_1} - t_{C_2} \leq T_g \end{cases}$	$\theta = 0.8$ $\gamma = 0.4$ $\sigma = 0.8$ $\mathrm{T}_g = 30\ min$

Definition 3.7. Merging & Splitting cluster. Cluster $A(id, Set < geohash >, gN, Set < objectId >, pN, t)$ is a merging cluster from clusters: $B(id, Set < geohash >, gN, Set < objectId >, pN, t)$ and $C(id, Set < geohash >, gN, Set < objectId >, pN, t)$ if:

$$\begin{cases} O_{id}(B, A) \geq \theta \\ J_{id}(B, A) \leq \gamma \\ t_A > t_B \\ O_{id}(C, A) \geq \theta \\ J_{id}(C, A) \leq \gamma \\ t_A > t_C \\ J_{id}(B, C) \leq \sigma \end{cases}$$

This definition does not use the location information and assumes that locations

of three clusters are different. If we add the location constrain and θ is some high value for example $\theta = 0.8$:

$$O_{geo}(B, A) \geq \theta \;||\; O_{geo}(C, A) \geq \theta$$

so if the overlap coefficient between a small cluster and a big cluster is high, we can state that one moving cluster has joined a standing cluster, therefore together they are merging clusters.

Up to this point we have defined eight basic patterns and two advanced pattern: merging/slitting. With basic patterns we could not use CEP operators and syntax, however, advanced patterns can be already defined in terms of CEP. Moreover, for advanced definitions we already used concepts from basic patterns (joined cluster, standing cluster).

3.3.3 CEP operators

This section shows an example how complex CEP events can be aggregated. Currently pure CEP events are not a part of the final system, however they can be considered as a future work. After having a set of different atomic events, we are able to monitor them and define pure CEP patterns for detection. Let us redefine advanced pattern *Merging & Splitting cluster* of the Definition 3.7 using operators from the Section 2.5.

We know that in order to detect a merging cluster A we need that two clusters B and C had joined it. Therefore we are looking for a sequence of two joined cluster events.

Definition 3.8. Merging cluster event. Let $J_1(A, B)$ and $J_2(C, D)$ be two joined cluster events. $M(J_1, J_2)$ is a merging cluster event if $B = D$ and can be presented with CEP operator Aggregation and Within:

$$M(J_1(A, B); (J_2(C, D), t), \text{ where } B = D$$

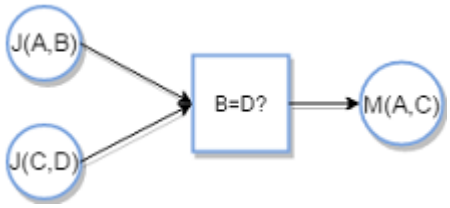


Figure 24: Merging cluster event is an aggregation of two joined cluster events with constrain

Time t is some small time constrain to filter out noisy data. Because in case $B = D$, timestamps $t_B = t_D$ are also the same. Therefore, two joined cluster events will occur almost simultaneously in the stream.

Traditional CEP language presented at the section 2.5 does not assume complex constrains, however in our use-case we need to ensure some condition at every stage. Merging cluster event of the Definition 3.8 is an example how new events can be derived from basic ones. This process allows very flexible and powerful set up of new and more meaningful patterns. Pattern logic is mostly based on a domain features of a data stream. We leave these definitions for a further work.

Figure 25 presents the overall workflow of the system. After detecting basic events, we have two option: create a new event based on a pattern and forward it further to a stream or just create a notification, that such event has been discovered and leave the action to the end user. Advanced pattern module is just for demonstration and not required for a system to function. From the other side, we can create basically infinite amount of additional layers to detect and take action on new events.

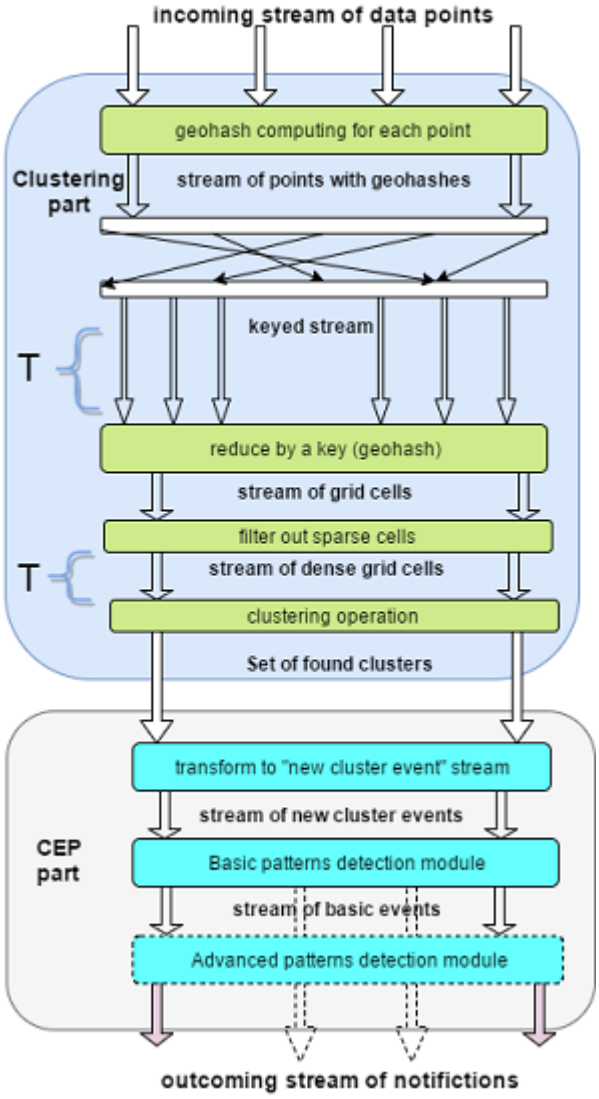


Figure 25: The overall workflow of the system

4 Implementation

This chapter presents the implementation of the previously described system. Requirements for the implementation framework remain the same, so for the design:

- **Stream processing**. The system should be able to process potentially heavy incoming data streams in a real-time, therefore the framework must support comprehensive stream computing.
- **Distribution and Scalability**. The framework must support distributed and parallel processing, in order to handle heavy workload and ensure the fault tolerance. In other words, the framework should ensure scaling out.
- **CEP-compliance**. For the pattern detection module of the system, we need the framework, which supports CEP programming.

In the following sections we will discuss software tools, which were chosen to satisfy our requirements. Also we will present the actual design of the clustering module and CEP module.

4.1 Software tools

The system is written in the Java programming language, version 8. Java has a number of advantages. First of all, it's arguably the most popular programming language, which provides us extensive support from the community in the form of different libraries and third-party software components. Secondly, Java programs run on the Java virtual machine (JVM), which allows them to run on any computer regardless of the hardware architecture. Another language, which is run on JVM is Scala. Scala has recently become very popular language for a distributed and parallel computing, therefore many libraries and even frameworks are written in Scala. Due to the JVM we can use Scala libraries, while programming in Java. Finally, Java is supported by major frameworks for distributed big data analytics.

Apache Flink was chosen as a framework for this master thesis.

4.1.1 Apache Flink

Apache Flink [54] is an open-source stream processing framework for distributed, high-performing, always-available, and accurate data streaming applications. It was developed as a result of the collaboration between TU Berlin, Humboldt University and Hasso-Plattner Institute [55]. Flink was developed as an alternative for the Hadoop MapReduce framework, providing fast in-memory computation.

Apache Flink satisfies all requirements, which we need, in order to build a system:

- **Stream processing**. Flink supports processing unbounded datasets out of the box. Despite many other frameworks (e.g. Apache Spark), Flink is natively designed to work with data streams. It also supports a dataset processing;

however, it treats them as a finite data stream. As a result Flink provides a great functionality (event time semantics, flexible windowing, iterative execution) for a stream processing.

- **Distribution and Scalability**. Being a successor of the MapReduce framework, Flink provides a fast parallel processing and easy distributed deployment following shared-nothing architecture.
- **CEP-compliance**. Finally, Flink has a number of additional libraries, one of which is FlinkCEP - a library for working with events and event patterns.

Besides listed benefits Flink also has a rich ecosystem, supporting different kinds of connectors as data sources and data sinks. However, being already very powerful out of the box, Flink lacks some specific functionality, which is required for our system. We compensate it, by using external libraries.

4.1.2 Libraries and software components

In this section we list all libraries and packages, which were used for the system implementation. We used Maven as a source for them.

- org.apache.flink - groudId for all flink packages in the Maven repository.
  - flink-java:1.0.3 - Java version of the Flink framework, version 1.0.3.
  - flink-streaming-java_2.10:1.0.3 - streaming Java component.
  - flink-clients_2.10:1.0.3 - additional package for Flink, which provides command line interface and remote execution on the server.
  - flink-cep_2.10:1.0.3 - CEP library for Flink.
  - flink-jdbc:1.0.3 - JDBC connector, which allows connection to a number of databases.
- ch.hsr:geohash:1.3.0 - implementation of geohashes in Java.
- org.elasticsearch:elasticsearch:2.3.4 - library for connection to the Elasticsearch. We used it just for testing purposes and debugging.

As you can see, the project is mainly developed by using just flink related libraries and Java 8 built-in components. The only one external library, which was used in the algorithm itself is a geohash library.

The whole application consists of 4 major components: data source, clustering, CEP and data sink.

4.2 Data Source

Our system is designed to work with continuous data streams. Even though streams are natural data source in many real-life scenarios, there are no available data streams, which we could use for the evaluation of the algorithm. So in order to test our system, the first task was to create a software component, which can emulate a data stream from a static dataset.

For this purpose, we followed the example of the Flink DataStream API Demo [56] provided by the dataArtisans, a company behind the Flink framework. The DataSource component is able to load data sequentially from a single CSV file, where records are sorted according to a timestamp field. Basically other fields do not play a role at this step. The program reads a file line by line and parses the timestamp field in each line. Stream emulation part has two input parameters:

- maxEventDelay - an integer value. Means time is seconds. It is responsible for producing events out of order, so the points, which are coming to the clustering part are unsorted within an internal of maxEventDelay. This is done in order to simulate real-life situation, where data often arrive with some delay due to network latency or other obstacles.

- servingSpeedFactor - the serving speed factor by which the logical serving time is adjusted. This parameter is responsible for "speeding up" the data flow. When the parameter equals 60, the events of 60 seconds real time arrive in one second of computing time. This is done for testing purposes.

After parsing the line, the program adjusts the actual timestamp with a maxEventDelay value and creates a new tuple with a new timestamp and the line itself. Therefore, the clustering part receives a data point with two timestamps - one, which is a part of a line (event time), and a timestamp, emitted by the data source component (processing time).

4.3 Clustering

At the beginning of the clustering part, we connect to the data source and read the stream of strings. Therefore, the first task is to parse the incoming string to a data object. We expect a single string to consists of next 4 parameters:

$$(id, timestamp, lat, lon),$$

where *id* - unique identifier of the object producing the data point, *timestamp* - event time, when data the point was emitted, *lat* - latitude in a WGS84 system, *lon* - longitude a in WGS84 system.

The first operation, which we perform is a map function, which transforms each line into a tuple of a String containing geohash and MovingtPoint object. MovingPoint consists of next fields:

- **id** - an integer field, keeping the identification number of the point.

- **timestamp** - a long value, keeping the event time in epoch milliseconds.
- **location** - a composite value of latitude and longitude, whcih are stored as float values.
- **geohash** - a string value, which keeps the geohash with a precision 7.

We on purpose duplicate the geohash value in a MovingObject itself and in a resulting tuple. In order to group objects by the geohash as a key in a following step.

As a result of the first parsing + mapping step we have a data stream of Tuple2<String, MovingObject> data types. Next operation is keyBy, which groups tuples with the same key into the same parallel processes.

After the elements are grouped by a geohash, we are ready to apply window function and reduce moving objects into grids. The input parameter to a window function is time. It can be in seconds, minutes or milliseconds. However, we need to remember about servingSpeedFactor parameter from a previous part. In case servingSpeedFactor equals 60, we have a correlation such as 1 second of real time equals 1 minute of a logical time, therefore time parameter of 3 equals window time interval parameter $T_i = 3$ minutes.

After the window function is applied, we perform a fold operation. Fold is a variation of a reduce function, with the only difference that the output value can be different type of the input value. Our input value is MovingObject, the output object is a type of GridCell and it consists of:

- **name** - a string value, which keeps the geohash with a precision 7.
- **density** - a float value. Describes a density coefficient of cell, in our case it is a simple count of moving objects.
- **lastUpdate** - long value. The timestamp of the last moving object, mapped to this cell.
- **objects** - a HashSet of moving object ids, mapped to this grid cell.

After the fold function we have a stream of GridCell type, where new items arrive in a sort of batches with a time interval of windowing. Next step is to filter out sparse grid cells, therefore we apply a simple filter function with a parameter $N = 5$ in our case, so that we keep only grid cells with $Density > N$.

After only dance grid cells are left in the stream we apply windowAll function in order to merge neighbouring grid cells. The difference between a simple window function and a windowAll function, is that the second one is for non-keyed streams, therefore it is not paralleled. We apply it with the same time parameter T_i as the first windowing, in order to be consistent.

Merging clusters is done sequentially according to the algorithm in the previous chapter. The output of merging is a data stream of tuples containing key (long value) and a value (set of found clusters): long value is common timestamp for all clusters, being formed in this window. This is necessary in order to distinguish clusters from different timestamp periods in the later analysis. Cluster object itself has following fields:

- **name** - a unique identifier of the cluster. We use built-in Java universally unique identifier (UUID) function to create it.
- **timestamp** - the timestamp of the latest movingObject, which belongs to this cluster.
- **gridNumber** - integer value. The amount of dance grids, of which this cluster consists.
- **objectNumber** - integer value. The amount of moving objects, of which this cluster consists.
- **ids** - a set of ids of moving objects.
- **geohashes** - a set of geohash strings, which belong to this cluster.

At this point, the clustering part is complete. In the implementation we strictly follow the design description. The process is presented at the Figure 26.

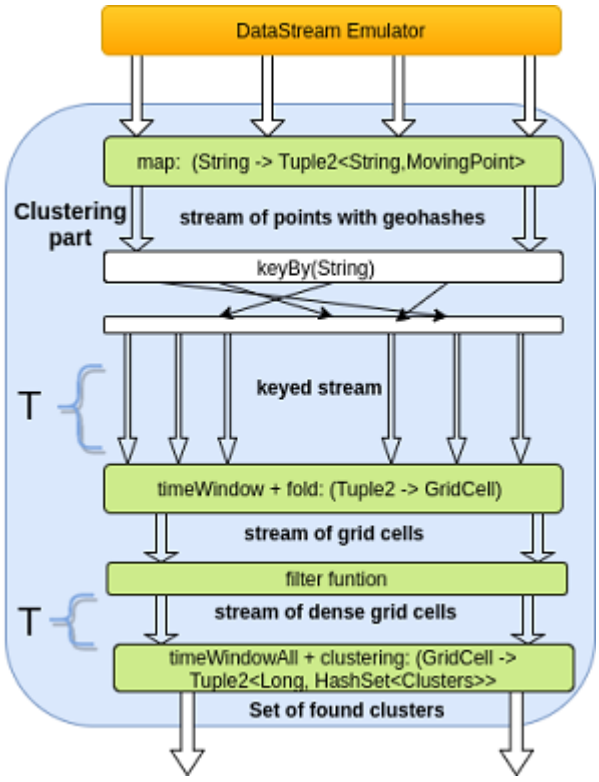


Figure 26: Clustering implementation with Apache Flink

4.4 Complex Event Processing

After the clustering is done, we have a stream of tuples Tuple2<Long, HashSet<Cluster», which arrives every time interval T_i. In order to start CEP analysis, the first task is to decompose each set of clusters, back to the cluster stream. As we

already discussed in the concept design chapter, we consider each cluster as a *new cluster event* in CEP terminology. In order to produce a stream of newClusterEvent objects, we apply windowAll function and iteratively transform each Cluster objects into newClusterEvent object. This time we apply windowAll function with a count parameter of 1 and not time. This is possible, because we need to transform each outcoming set of clusters and there is no need to use time parameter at this step. newClusterEvent object consists of next fields:

- **key** - a geohash string with a precision 7.
- **objectNumber** - integer value. The amount of moving objects, which this cluster consists of.
- **gridNumber** - integer value. The amount of dance grids, which this cluster consists of.
- **timestamp** - long value. The timestamp, when the cluster was created. We use this value to distinguish clusters from the same iteration.
- **ids** - an array of object ids. Ids stored as integers inside the array.
- **cell** - an array of geohash strings, which indicate current cluster location.

From this moment we work with a data stream of *new cluster events*. And can apply CEP operators in order to detect patterns.

Our implementation of the clustering algorithm strictly follows the initially designed concept, however with the CEP implementation this is not the case and the implementation is slightly different from the original design. This happened because there is no common API for the complex event processing, every library differs and does not provide the same functionality. Flink's CEP library pattern operators are listed in the Table 5.

These 7 operators are different from operators, which we have discussed at the Section 2.5 as an example. Only Within operator matches. Aggregation, conjunction and sequence can be defined with a Flink CEP API quite easily. However, it is not very helpful in our case. Our pattern detection is entirely based on a cluster analysis, where in order to detect a pattern and emit a new event, we need to compare two clusters, which occur in a sequence. Unfortunately, Flink CEP API does not provide such functionality. Operator *where* can be only applied to a single event. There is no way to trigger a pattern based on a complex condition, where we can compare two or more events. However, Flink CEP API provides a different way to apply complex conditions. This can be done with the help of *PatternSelectFunction<IN, OUT>* and *PatternFlatSelectFunction<IN, OUT>* interfaces. Therefore, in order to detect our defined patterns from the Section 3.3.1, we firstly need to create a pattern stream with a help of Flink CEP operators and then select desired patterns manually with our selectors, which implement two mentioned interfaces. This approach is definitely not the most efficient and creates a lot of overhead in terms of the workload of the system, however it is enough for us to test the algorithm.

Table 5: Flink CEP pattern API operators. Source: [54]

Pattern Operation	Description
Begin	Defines a starting pattern state.
Next	Appends a new pattern state. A matching event has to directly succeed the previous matching event.
FollowedBy	Appends a new pattern state. Other events can occur between a matching event and the previous matching event.
Where	Defines a filter condition.
Or	Adds a new filter condition to already defined one.
Subtype	Defines a subtype condition for the current pattern state. Only if an event is of this subtype, it can match the state.
Within	Defines the maximum time interval for an event sequence to match the pattern.

In order to detect basic patterns, we need to compare two *new cluster events* and for advanced patterns three *new cluster events*, therefore we created three Flink CEP patterns:

- begin(A).followedBy(B)
- begin(A).followedBy(B).within(T_g)
- begin(A).followedBy(B).followedBy(C).within(T_g)

A, B, C - *new cluster events*, T_g - is a time interval within which we consider clusters for comparison. We declare two patterns for the two clusters comparison, the one is with *within* operator and the second one is without. That is because for dynamic patterns, such as moving cluster or separated cluster, we take into consideration time constrain and for static pattern, such as evolving cluster, time interval is not important.

In order to detect all pattern from our algorithm design, we use combinations of three Flink CEP patterns and three selector functions, where conditions are defined:

- **EvolvingPatternSelector** - selects only patterns, which match the evolving pattern condition.
- **TwoClustersSelector** - selects all other patterns, with two patterns comparison.
- **ThreeClustersSelector** - selects our advanced patterns, where three clusters are involved.

In each selector function we firstly calculate Jaccard similarity coefficient and Overlap similarity coefficient and then check them for the set of constrains. The whole picture is presented at table:

Table 6: Initially designed patterns and their implementation with Flink CEP API

Pattern name	Flink pattern	Selector function
Moving cluster	followedBy + within	TwoClusters
Standing cluster	followedBy + within	TwoClusters
Evolving cluster	followedBy	EvolvingCluster
Joined/Separated	followedBy + within	TwoClusters
Growing/Shrinking	followedBy + within	TwoClusters
Merging/Splitting	followedBy + followedBy + within	ThreeClusters

The result of each selector function, is a stream of notifications. Notification on each pattern is emitted to a correspondent stream after each pattern is detected. In the end we combine three streams into one notification stream. At this point the process is considered finished. The CEP part implementation workflow is presented at the Figure 27.

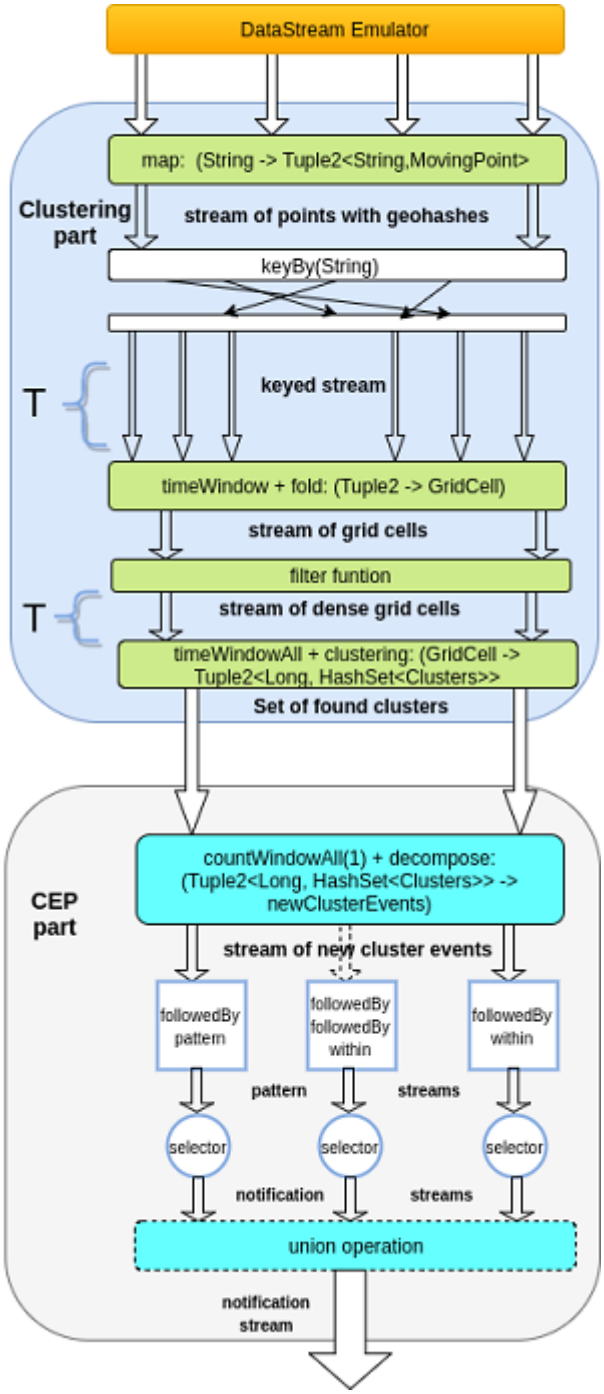


Figure 27: The CEP implementation workflow. Arrows presenting streams in the CEP part mean logical division, not distribution

5 Evaluation

This chapter evaluates the developed system. In order to make sure that the system works as expected it should run on the data, where we have the knowledge that clusters and moving patterns exist. To the best of our knowledge, there are no available data streams or even datasets with such information. Therefore, the evaluation strategy is next:

1. Check the cluster quality with the artificial data
2. Test pattern detection with the artificial data
3. Run the whole system with the real-life data

In order to test if the algorithm works correctly, artificial dataset should be created. It can be done by using CATLES simulation tool [57]. CATLES was developed as a part of the LBS research at the TU Berlin specifically for such purposes. It allows to create GPS traces of moving objects at any place of the Earth using Google Maps service as an underlying maps provider. It also allows to choose the mean of the transportation from walking, cycling or vehicle. The first task is to ensure that cluster discovery functions with a sufficient cluster quality. The second task is to test pattern detection with artificial data. After that the algorithm can be tested on real data streams.

T-Drive dataset [58] is used for the real-life scenario evaluation. It was collected as a part of the Microsoft research. The dataset contains GPS trajectories of 10357 taxis during the period of Feb. 2 to Feb. 8, 2008 within Beijing. The average sampling interval is 177 seconds with a distance of about 623 meters. This dataset was used in many other research projects [59], [60]. In our opinion it is one of the best location related datasets, which is available in a free access.

For evaluation purposes Elasticsearch engine is used as a database to store results. Analytics and visualization platform called Kibana is used for the graphical representation. Web Map Service (WMS) from the OpenStreetMap Foundation is used as a map provider to display results in Kibana.

5.1 Evaluation with artificial data

The first task is to evaluate the clustering part. The quality of discovered clusters should be analyzed. The most straightforward and efficient way to do it, is to compare, discovered by the algorithm, clusters against the ground truth. Therefore, a data stream containing clusters is required for this task.

5.1.1 Cluster quality evaluation

We created a dataset of GPS traces of 25 objects (vehicles), which are moving from Ernst-Reuter Platz to Alexander Platz in Berlin. Figure 28 shows the route.

There is no real time in CATLES, only relative logical time, which always starts with one millisecond and ends when the recording is stopped. In order to create real

Figure 28: The chosen route to test the cluster quality

timestamps, we used an estimated travel time from Google Maps (15 minutes) and made a correlation with a logical time of GPS traces from CATLES. In case of this route, CATLES time is 249617 milliseconds. This gives us a notion that one minute of the Google time approximately equals 16641 milliseconds in CATLES. Having such correlation, GPS traces of 25 objects were created with a sampling frequency of 1 minute. In order to obtain 25 traces, we divided CATLES trace into 15 chunks and took first 25 entries from each chunk. In this way each point has a different location and the same location is not used 25 times. However, these 25 points are still very close to each other. After having 15 chunks of 25 objects, we assign objects from a first chunk the real timestamp as of 12:00, 20 February 2017. Then through a simple loop the timestamp T_i is set for remaining chunks simply increasing it by one minute at each iteration. In the same loop a unique ID is also assigned for each entry. In order to make traces more natural, each timestamp is randomized by the time uncertainty T_r.

5.1.1.1 General test case

In the first test we use $T_r = 10$ seconds, so each timestamp is in the interval $[T_i - 5, T_i + 5]$. Using True Positive Rate or Sensitivity definition from statistics, let us declare that the cluster quality is:

$$Q_c = \frac{C_{found}}{C_{original}},$$

where C_{found} - amount of objects discovered by the algorithm and $C_{original}$ - amount of objects in the original cluster. In this case $C_{original} = 25$.

This definition of the cluster quality is also known as In general, there are five input parameters:

- T_r - an timestamp distribution of objects
- T_i - a time of a fixed window model
- D - density threshold, which is responsible for considering a grid cell dense or sparse

- T_s - sampling rate of an object
- $C_{original}$ - amount of moving objects

We run the first test with next parameters presented in the Table 7.

Table 7: Clustering quality test parameters

T_r	T_i	D	T_s	$C_{original}$
10 seconds	60 seconds	none	60 seconds	25

The visual result is presented in the Figure 29

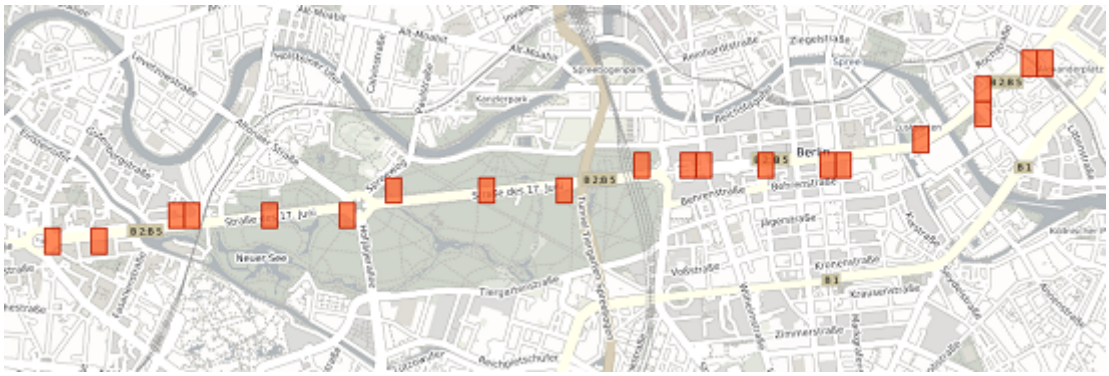


Figure 29: Clustering output with 25 objects. One tile - is one geohash with the precision 7

The algorithm has discovered clusters along the whole route in 15 different places. The final count of found clusters - 38. Figure 30 illustrates the distribution of clusters by size.

Figure 30: Cluster quality distribution. Test case without applying density threshold

At the first glance the result is quite random, however there is a major cluster with 20 objects or more at every time interval T. Also there is a clear division for small clusters and big ones. Small clusters consisting of 1 to 5 objects can considered as noise. This noise occurs because we have a small uncertainty of T_r and at each

window function objects from the same original cluster are left in different time slices. In the Figure 29 there are only 15 clusters seen, because there is an overlap at each place and below one main cluster there is a couple of noise clusters. If we consider only major clusters, where amount of objects $C_{found} \geq 5$, the average cluster quality is:

$$Q_c = 89\%$$

The 15th cluster at Alexander Platz is a noise cluster, therefore only 14 clusters were taking into consideration for the cluster quality calculation. The idea that clusters cannot be detected at every place, where they are expected, provides the notion of the consistency coefficient:

$$C_{coef} = \frac{P_{found}}{P_{original}},$$

where P_{found} is a number of places, where clusters were found and $P_{original}$ - number of original cluster locations. In our case:

$$C_{coef} = \frac{14}{15} * 100\% = 93\%$$

This result is extremely high and already at this phase we can state that clustering algorithm works. However real-life data usually is not that good. Chosen T_r parameter is optimistically low.

Using results form the first test, the density threshold coefficient D is introduced. It is responsible for filtering out sparse cells, therefore noise clusters.

5.1.1.2 Test case for checking time distribution

Let us run another clustering test with parameters presented in the Table 8.

Table 8: Clustering quality test parameters

T_r	T_i	D	T_s	$C_{original}$
60 seconds	1 minute	5	1 minute	25

In this test the time interval of $[T_i - 30, T_i + 30]$, which basically covers the whole period of the sampling time. Also sparse grid cells were filtered out. Figure 32 presents the result. This time due to a filter parameter results are much cleaner and only 14 clusters were discovered. One for each time gap. The cluster size distribution is presented in the Figure 31.

The average cluster quality and the consistency coefficient are:

$$Q_c = 88\% \quad C_{coef} = 93\%$$

As it is seen, the time distribution almost does not affect the clustering quality. This result is not really accurate though, as we only changed the time distribution

Figure 31: Cluster quality distribution. Test case with applying density threshold $D = 5$

Figure 32: Clustering output with 25 objects with the time distribution and density threshold

without changing locations. Basically the time delay does not play a role, as soon as cars update their locations from the same grid. This is very unlikely in real-life scenarios, where time is always strictly bounded with the location. However, this only matters in the pattern detection and at this phase of testing, the goal is to discover optimal input parameters for following tests.

5.1.1.3 Test case for checking optimal window size

As it was discussed in the Section 3.2.1 clustering quality should heavily depend on the window time interval T_i. Let us run two tests with $T_i < T_S$ and $T_i > T_S$ accordingly to see the difference.

Table 9: Clustering quality test parameters with $T_i < T_S$

T_r	T_i	D	T_s	$C_{original}$
60 seconds	30 seconds	5	60 seconds	25

The result of this test is presented in the Figure 33. The final amount of clusters is 17. Figure 34 shows the cluster quality (size) distribution.

Figure 33: Clustering output with 25 cars. $T_i < T_S$ case.

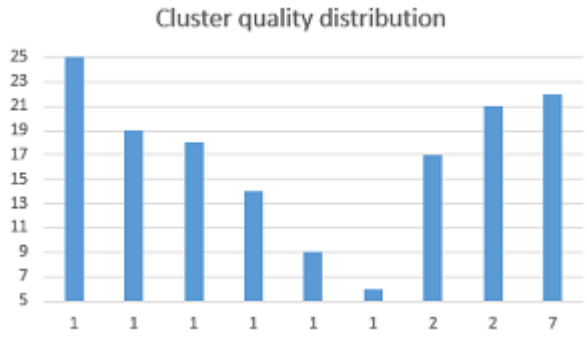


Figure 34: Cluster quality (size) distribution. $T_i < T_S$ test case

The overall cluster quality and consistency coefficient:

$$Q_c = 66\% \quad C_{coef} = 100\%$$

This result correlates with our discussion on the window model, with a higher fragmentation we receive more frequent results, but lose significantly in the cluster quality.

Scenario $T_i > T_S$, which is presented in the Table 11, demonstrates more stable results with 14 clusters and smooth distribution, presented in the Figure 35.

Table 10: Clustering quality test parameters with $T_i > T_S$

T_r	T_i	D	T_s	$C_{original}$
60 seconds	90 seconds	5	60 seconds	25

The average cluster quality is:

$$Q_c = 90\% \quad C_{coef} = 93\%$$

The quality of the most of clusters is higher than in previous tests. We run one more test with T_i being twice bigger than T_s.

Figure 35: Cluster quality (size) distribution. $T_i > T_S$ test case

Table 11: Clustering quality test parameters with $T_i > T_S$

T_r	T_i	D	T_s	$C_{original}$
60 seconds	120 seconds	5	60 seconds	25

Visual output of this test is presented in the Figure 36.

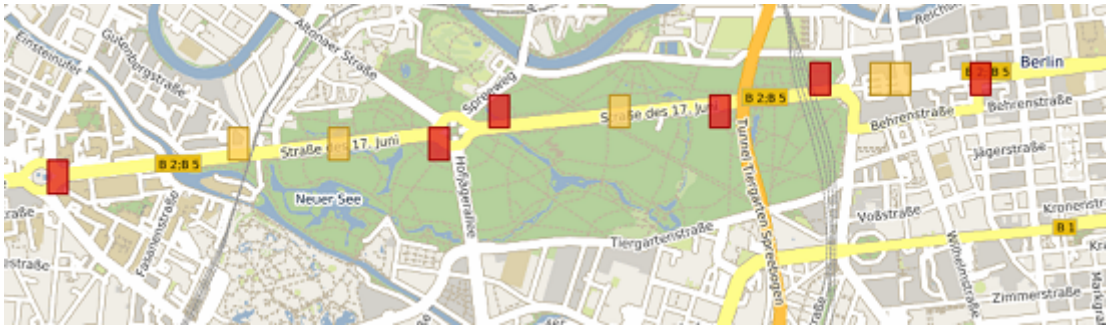


Figure 36: Clustering output with $T_i = 2 * T_S$

The algorithm has discovered 10 clusters. Figure 37 shows the quality (size) distribution.

Figure 37: Cluster quality (size) distribution. $T_i = 2 * T_S$ test case

This scenario provides high clustering quality, but low consistency:

$$Q_c = 94\% \quad C_{coef} = 66\%$$

Moreover, 10 clusters were emitted during 7 window operations, which means that some pairs of clusters have the same creation timestamp, which would affect during the pattern detection.

To sum up the clustering quality evaluation part we can state that algorithms works well. The only major input parameter, which effects the result is window size interval T_i. Figure 38 shoes the dependency. We can state that T_i should be bigger than T_s but lower than $2 * T_s$. In our case $T_i = 1.5 * T_s$. We will use this parameter for the pattern detection evaluation.

Figure 38: Optimal window size evaluation. $T1 = 0.5 * T_s$; $T2 = T_s$; $T3 = 1.5 * T_s$; $T4 = 2 * T_s$

5.1.2 Pattern detection evaluation

In the Section 3.3 eight basic patterns and two advanced patterns were defined. Here we evaluate the most interesting of them, namely: moving cluster, standing cluster, hotspot, growing and merging clusters pattern. There is no need to explicitly evaluate all patterns, as many of them are quite similar and based on each other. For example, merging cluster pattern is the same as splitting cluster pattern, only with the different timestamp order, therefore if merging cluster pattern functions well there is a guarantee that splitting also works. Moreover, as they both are extensions of joined and separated patterns, we can verify them automatically just by testing merging clusters pattern. Shrinking pattern equals growing, but with the inverse timestamp order.

In order to test each of these patterns, the specific dataset is required. The same route has been used for testing pattern detection. In the previous set of tests, the timestamp distribution value $T_r = 60$ seconds was used. However, it was only time distribution without the location correlation. Test results have shown that it almost does not affect clustering results as soon as updates arrive from the

same location. In order to simulate the data stream closer to a real-life scenario, location updates were bounded to timestamps. Also the delay parameter T_d is introduced. T_d or maxEventDelay (as it is named in the previous chapter) is an integer value, responsible for producing events out of order. In this way the event time is different from the processing time, which is the case for almost all real-life streaming applications.

Figure 39 illustrates features of time and location distribution. There are couple of changes from the previously used dataset. Firstly, the objects sampling rate T_s is now 2 minutes, as there is no need to receive updates every single minute with such small location changes. $T_r = 40$ seconds, so the moving object emits the location update at any point in the interval $[T - 40, T]$, where T is the exact sampling time, i.e, 2, 4, 6, etc. minutes. Now the location is correlated with the timestamp, so the moving object at the time $T - 40$ is approximately by 288 meters behind the object, which updates the location at the time T. The speed is 26 km/h (15 minutes' route and 6.5 km distance). The delay value $T_d = 20$ seconds, which makes the uncertainty time interval one minute and uncertainly location is still approximately 300 meters.

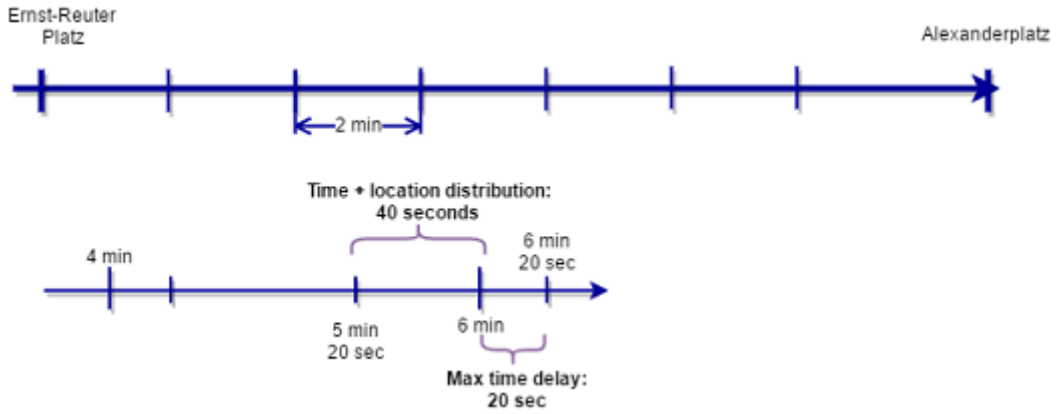


Figure 39: Time characteristics of the dataset. $T_s = 2$ minutes; $T_r = 40$ seconds; $T_d = 20$seconds

From the clustering quality evaluation, we have found that the optimal value of the window size T_i should be in the interval $[T_s, 2 * T_s]$, therefore, having $T_s = 120$ seconds, the $T_i = 150$ seconds for the pattern testing. Another parameter, which is changed to match real-life scenario is the density threshold D. Now we use two density thresholds: D_g and D_c. D_g is an original density coefficient, which is responsible for considering a single grid cell dense or sparse. D_c is an additional threshold responsible for filtering out clusters with less than half of original objects. This separation of the density coefficient into two parts allows us to consider more grid cells for clustering, but still work only with quality clusters in the pattern detection phase. Table 12 presents all input parameters for the following tests.

Table 12: Input parameters for the basic pattern detection

T_r	T_i	T_s	T_d	$C_{original}$	D_g	D_c
40 seconds	150 seconds	120 seconds	20 seconds	25	3	12

According to the pattern detection design, the algorithm also depends on next parameters:

- Jaccard coefficient $J_{id} - \theta$
- Jaccard location coefficient $J_{geo} - \gamma$
- Overlap coefficient $O_{id} - \alpha$
- Overlap location coefficient $O_{geo} - \beta$

Values presented in the Table 13 are used for the basic patterns evaluation.

Table 13: Parameters for the basic pattern evaluation

	θ	γ	α	β
Moving cluster pattern	0.5	0.5	-	-
Standing cluster pattern	0.5	0.5	-	-
Hotspot	0.5	0.5		
Growing cluster pattern	0.4	-	0.8	0.8

5.1.2.1 Moving cluster pattern

The result for the moving cluster pattern is presented in the Figure 40

Figure 40: Three pairs of moving clusters

The algorithms detected 3 moving cluster events. Tables 14 presents short summary about each of them.

Clusters became more spread and the quality of clustering dropped significantly due to the distribution. Nevertheless, the algorithm has detected three moving cluster events by analyzing 6 discovered clusters. In reality this is the same cluster,

Table 14: Three moving cluster events

	First cluster	**Second cluster**
First moving cluster event		
Location (geohash)	u336xpt	u336xrd, u336xrs, u336xre
Amount of objects	25	19
Cluster quality	100%	76%
J_{id}	0.76	
Second moving cluster event		
Location (geohash)	u336xrs, u336xr	u336xxz, u336xxy, u336xzb
Amount of objects	13	19
Cluster quality	52%	76%
J_{id},	0.55	
Third moving cluster event		
Location (geohash)	u33db24, u33db21	u33dbbs, u33dbbe
Amount of objects	13	14
Cluster quality	52%	56%
J_{id},	0.5	

Table 15: Standing cluster event summary

	First cluster	Second cluster
Location (geohash)	u33dc1r, u33dc1q	
Amount of objects	24	17
Cluster quality	96%	68%
J_{id}	0.76	
J_{geo}	1	

however the system treats it as independent clusters and creates three independent events. There is a possibility to combine moving cluster events together to create traces with the help of advanced CEP patterns. It can be a task for the future work.

5.1.2.2 Standing cluster pattern

In order to detect this pattern, we extended the dataset for 2 periods of the sampling time (4 minutes). Meaning that when the cluster reaches the destination location (Alexanderplatz), it keeps standing and updates the location two more times. The result can be seen in the Figure 41.

Figure 41: Two standing clusters at Alexanderplatz

The Figure 41 also illustrates that firstly algorithm has discovered moving clusters and then triggered standing pattern event. The Table 15 provides information about the event.

Standing cluster is easier to detect than the moving cluster. Also the cluster quality rises as the location remains unchanged and the location distribution does not affect the result.

5.1.2.3 Hotspot

For testing this pattern, the original dataset was extended by adding another 25 objects, which are repeating the original route, however half an hour later. Also new 25 objects have different ids. Figure 49 illustrates findings.

Table 16 shows the events summary. Hotspot is basically the simplest pattern to discover.

Figure 42: Two hotspot locations at Alexanderplatz and Ernst-Reuter Platz

Table 16: Summary of two hotspot events

	First cluster	Second cluster
Alexanderplatz hotspot		
Location (geohash)	u33dc1r, u33dc1q	
Amount of objects	25	18
Cluster quality	100%	72%
J_{id}	0	
J_{geo}	1	
Ernts-Reuter Platz hotspot		
Location (geohash)	u336xpt	
Amount of objects	25	18
Cluster quality	100%	72%
J_{id}	0	
J_{geo}	1	

5.1.2.4 Growing & Joined cluster

For testing a growing cluster pattern the original dataset was extended by prolonging the waiting time at Alexanderplatz of the first group of objects (the same idea as for the standing cluster, but now about 20 minutes waiting). Also one smaller cluster of 12 objects was following the same route 15 minutes after the first one. The idea is that the second cluster will arrive at Alexanderplatz and join the first cluster, so they will form a growing cluster. For this test the cluster density threshold D_c was lowered to 6 (50% of the smaller cluster size).

Growing cluster pattern is just an extension of the joined cluster pattern, therefore results of the joined cluster detection were also saved to Elasticsearch for the analysis. Figure 43 presents all results.

The algorithm has discovered 4 joined cluster events, however only one of them was a real event. It is presented in the Table 17. Three other events are false positives and were triggered on the same moving cluster. This happened because of the eventual fragmentation, which happens often with the time and location distribution. The idea is that the big cluster is divided into two parts by the window operation, so two parts being in reality one cluster are analyzed in different time slices and considered as two clusters. At the next window operation, they are again

Figure 43: Smaller cluster has joined the bigger one at Alexanderplatz

in the same time slice, therefore seen as one cluster by the system. Unfortunately such fragmentation always occurs with the time window strategy $T_i > T_s$.

Table 17: Joined cluster pattern summary

	First cluster	Second cluster
Location (geohash)	u33dc1k	u33dc1r, u33dc1q
Amount of objects	6	28
Cluster quality	50%	75%
J_{id}	0.222	
J_{geo}	0	
O_{id}	1	

The system did not catch the growing cluster event, because this pattern is not possible to detect by the design.

5.1.2.5 Merging cluster

For testing the merging clusters patter next scenario was simulated: 20 objects are moving from Erntst-Reuter Pltaz to Brandenburg Gate and another 20 objects moving from the Mexican Embassy to Brandenburg Gate as well. According to the simulation two groups should meet at the Victory Column and arrive together as one cluster to the destination point. Figure 44 illustrates routes.

Two datasets were created and later combined into one for this test. Both datasets follow the same distribution strategy as in previous tests. However, because they have different length and travel time, they were not synchronized against each other. Table 18 presents input parameters for this test.

Table 18: Input parameters for the basic pattern detection

T_r	T_i	T_s	T_d	$C_{original}$	D_g	D_c
40 seconds	150 seconds	120 seconds	20 seconds	20	3	10

Figure 45 presents the outcome result.

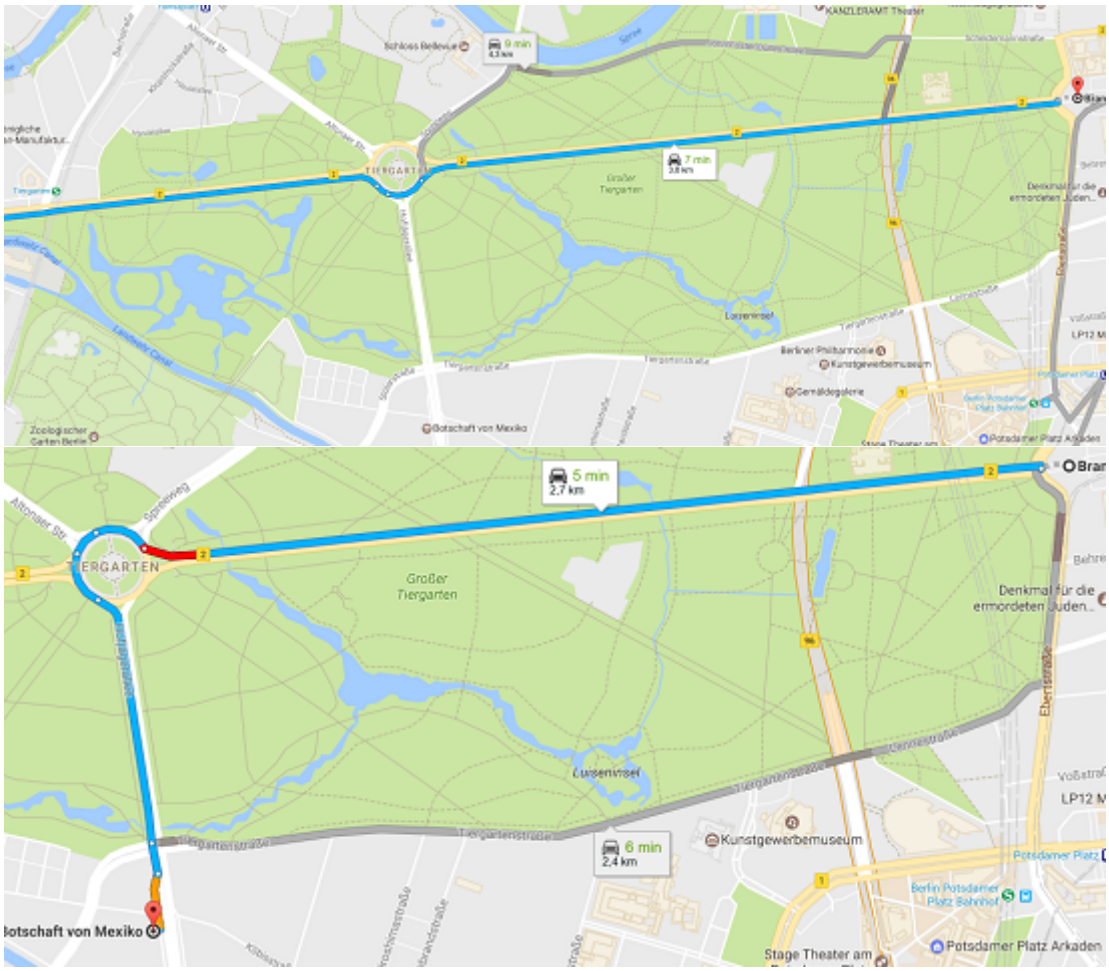


Figure 44: Two groups of objects should meet at the the Victory Column

Table 19: Two small clusters resulted into one

	First cluster	Second cluster	Result cluster
Location (geohash)	u33db0j	u33db20, u33db0p	u33db2k, u33db2m, u33db27
Amount of objects	11	16	34
Cluster quality	55%	80%	85%
$J_{first-result}$	0.222		
$J_{second-result}$	0.351		
$J_{first-second}$	0		

The algorithm has successfully recognized the merging clusters pattern. Table 19 presents the result of 3 clusters analysis. In this scenario clusters should had been detected earlier, however due to the time asynchronization the algorithm only detected the final cluster at the destination point.

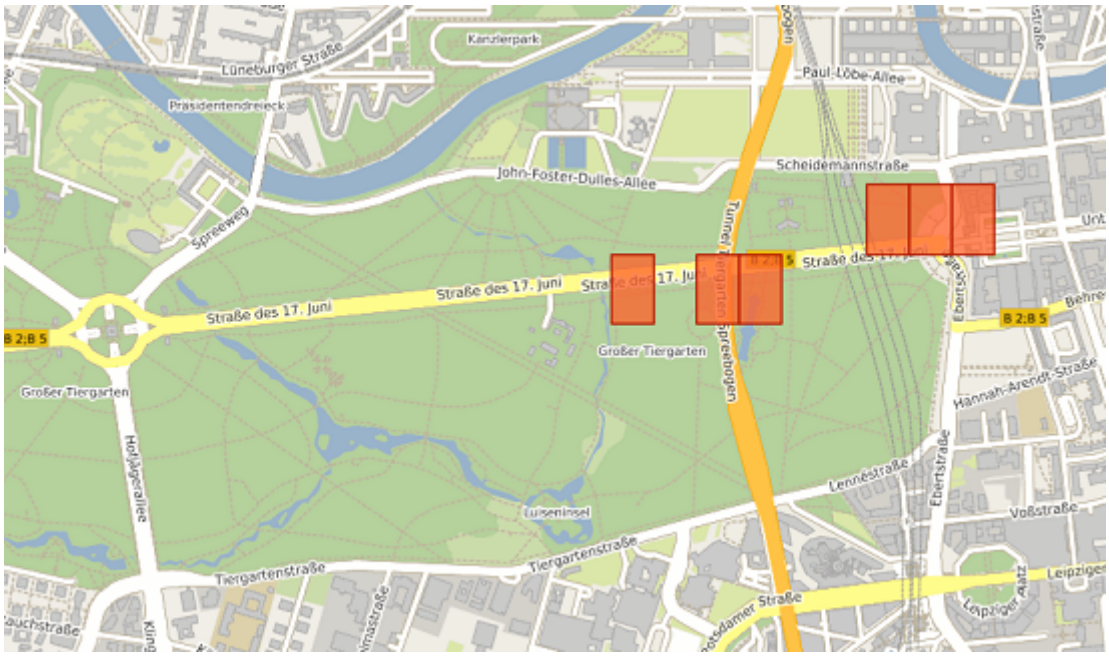


Figure 45: Two clusters have merged into one at the Brandenburg Gate

5.1.3 Discussion on the evaluation

To sum up, the system has proved its' functionality and an ability to analyze moving objects. However, closer to real-life conditions (location + time distribution) the quality of results decreases. The average cluster quality from all tests is 72%, which can be considered high, however this is mainly because of the standing cluster pattern and hotspot pattern, where cluster quality is usually high. In more complex patterns, such as moving cluster or merging clusters, there are often clusters with 50% to 60% quality.

The algorithm depends on many input parameters and they affect the output result significantly. The most important parameter is the density threshold D. The strategy with splitting D into two parameters, D_g - density threshold for grid cells and D_c - density threshold for cluster, has proved its efficiency. This allows to discover clusters more accurate in terms of the location distribution. The second most important parameter is the window size T_i. Experiments have shown the best result with the T_i being a bit bigger than the sampling time.

Joined cluster pattern and Growing cluster pattern did not work as expected. Joined cluster pattern produces a lot of false positive results because of the fragmentation. Growing cluster pattern, being an extension of the joined cluster pattern (it has one more constrain) did not provide any result. Partly this was due to the implementation issues, but mainly because the design of the pattern is not optimal and requires some modifications.

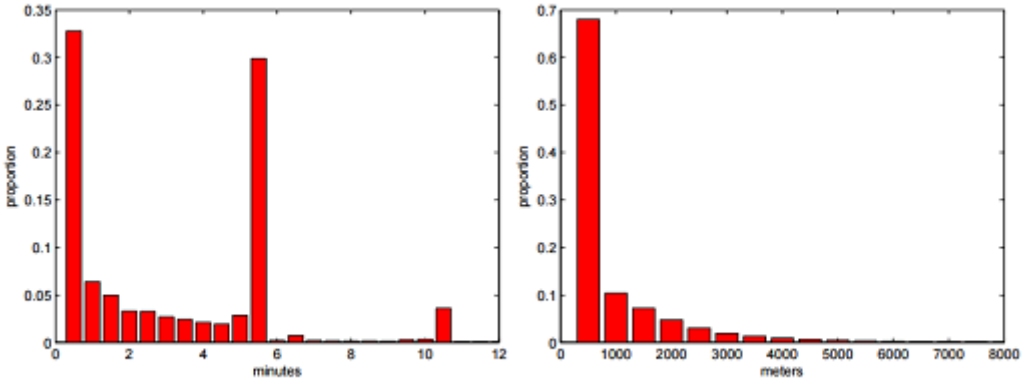
The main outcome is that the algorithm needs to be calibrated before working in the real-life environment. Having many input parameters, it is hard to evaluate it properly and provide some final general conclusion. However, as the proof of concept,

it has shown good results and should be considered for the future development.

5.2 T-Drive example

In the previous Section we have verified the system's functionality. Now it can be applied to a real dataset. This cannot be considered as a true evaluation, because there is no ground knowledge about clusters and moving patterns in the dataset. However, it is important to test the system on the real-life scenario and try to analyze the result.

From the dataset description we only know the time and distance distribution. Figure 46 presents them.



(a) time interval (b) distance interval

Figure 46: Histograms of time interval and distance between two consecutive points. Source: [58]

Even though the average time distribution is 177 seconds, it is clearly seen from the Figure 46, that there is a major division. Majority of the location updates arrive within 1 minute. The second big group updates the location information every 5 minutes. Because the dataset consists of taxi cars, the assumption is that cars on shift (moving cars) update their location frequently and cars waiting at parking lots send signals not so often. Therefore, while choosing input parameters for the algorithm execution, we should focus on the smaller sampling value.

We execute the algorithm with next parameters:

- Density threshold $D = 5$.
- Window size $T_i = 1, 5$ minutes.

The system was set up to discover moving, standing, merging and hotspot patterns. Joined and growing patterns were excluded based on previous test results.

As it is impossible to analyze the result against the ground knowledge, here we provide some interesting statistics. The system has discovered 3075 clusters. Figure 47 presents the cluster distribution over time. It is clearly seen that in the beginning there are much more clusters than in the end. Maybe it is somehow related to the

fact that last two days is a weekend. However, even for the weekend it seems strange.

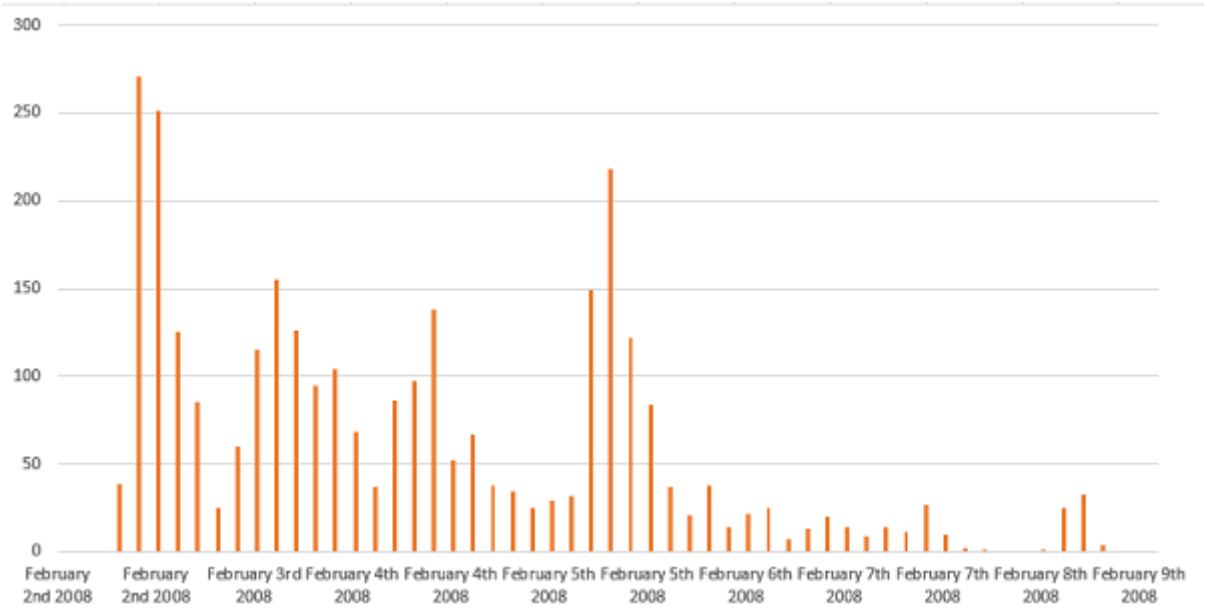


Figure 47: Histogram of the cluster count according to the time

Figure 48 presents the distribution of clusters by size. As it is seen, the majority of clusters consist of 6 cars. Also 2188 (83%) of all clusters consist of 1 grid cell. Actually, considering the single grid cell size ($\approx 152.9x152.4$ meters) and the dataset origin (taxi cars), the density threshold parameter $D = 5$ seems high for a single grid. It should be split into two parameters in the later analysis, as we already discussed in the previous Section.

As for the pattern detection, the statistic is next: only 30 standing clusters and only 8 moving clusters were found. The result is extremely low. Partly it can be explained that taxis do not move in groups, however too high density threshold probably also affected the result. 107 merging clusters were discovered. This might seem high comparing to the low moving clusters result, however most of merging clusters were found at parking lots, where big amount of cars are located. Here we can see the effect of the fragmentation, cause by the windowing. The same cluster is being considered as two small clusters at one time slice and later it is considered correctly as the big one, triggering merging cluster pattern.

The most useful pattern in this test case is the hotspot. 153 locations were detected as hotspots. Figure 49 illustrates the big picture. The most busy area is an airport (top right corner on the map). The second busiest area is in the city center, next to the Beijing Railway Station. A lot of hotspots were also detected at the beginning of the highway on the way to the airport. Our assumption is that there is the main city taxi parking.

From the algorithm evaluation on the T-Drive dataset, we can state that clustering part of the algorithm works well and the designed system can be used for the cluster discovery of moving objects. Pattern detection performed poorly on this dataset.

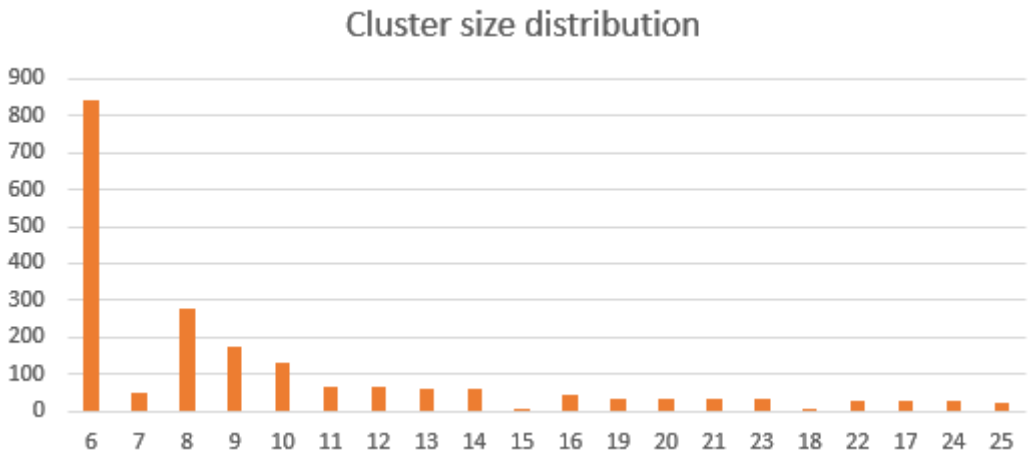


Figure 48: Clusters distribution by size

However, taking into consideration that almost all patterns were verified on artificial data, we can make a conclusion that T-Drive dataset does not contain many moving patterns.

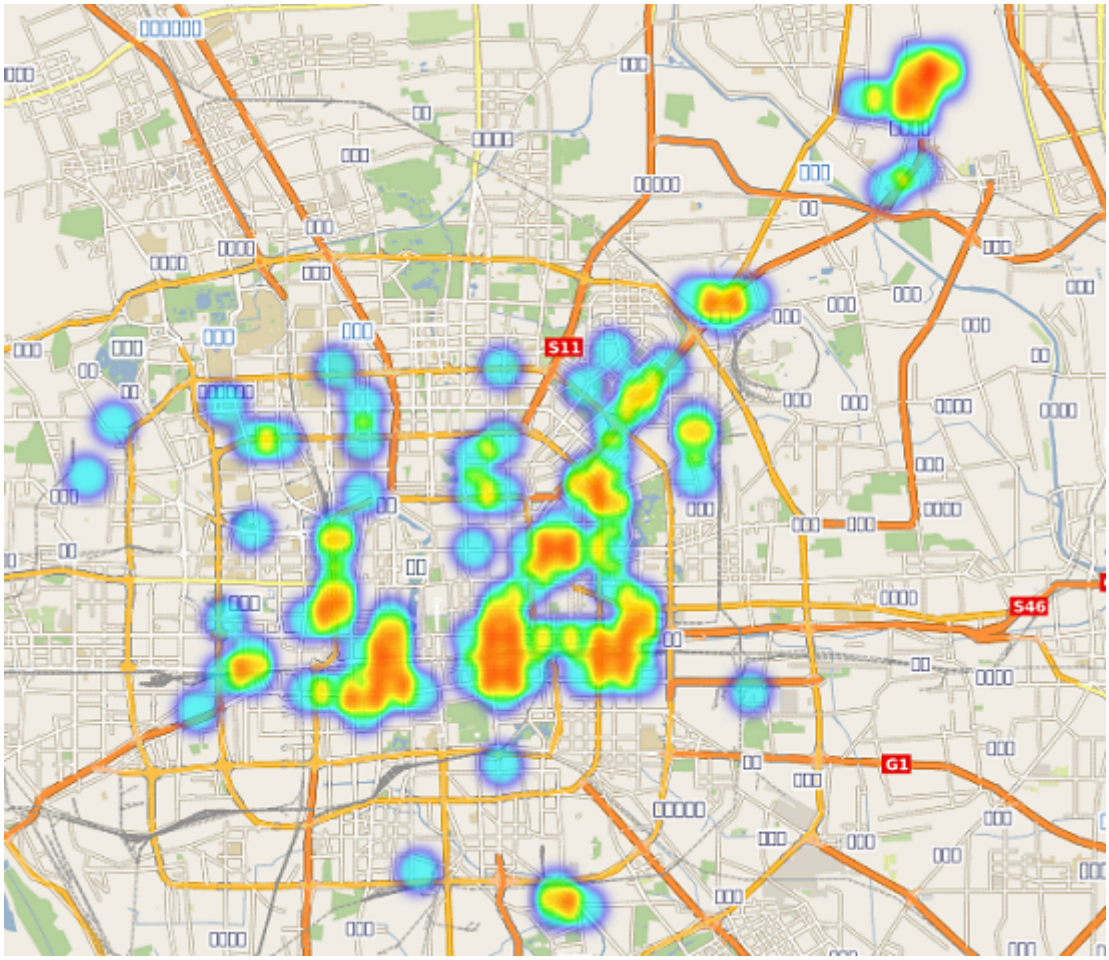


Figure 49: Heatmap of hotspot locations in Beijing

6 Conclusion

In this thesis we have designed, implemented and evaluated the system for detecting and analyzing moving crowds in the continuous spatio-temporal data stream.

This project started as a natural continuation of the proactive Location Based Services development, however it has rapidly evolved beyond the scope of LBS and resulted into a comprehensive research at the intersection of Data Mining, Stream Processing and Distributed Computing.

The main motivation for this thesis was encouraged by three indisputable facts:

- Poor development of proactive LBS targeting multiple users
- Rapid growth of spatio-temporal data
- Recent rise of various Big Data processing technologies

The outcome of such diversified project is a the unique system, which is capable of real-time cluster recognition in a continuous spatio-temporal data stream. Furthermore, it is able to extract meaningful knowledge and discover interesting behaviour by exploring just created clusters using innovative pattern detection approach based on Complex Event Processing technology.

From the technical point of view, developed system combines some of the best practices of distributed computing and cluster analysis, which allows it to operate in a truly distributed manner. Also this is one of the first completely online systems, which can perform clustering operation without using a database. This is a great advantage comparing to many so-called two-phase algorithms. The system is based on powerful stream processing framework Apache Flink.

From the research point of view, developed clustering algorithm is absolutely unique, as it was developed from scratch adopting some of the best clustering practises as well as introducing some novelties, such as utilising geohash as a natural grid structure. The pattern detection part is particularly important, as it is a pioneer in the area of moving cluster analysis. There were almost no attempts to analyze moving clusters with the help of CEP. There is no even common terminology in this area.

The evaluation of the system has shown very promising results. Especially clustering part, which performed well not only with artificial data, but also in the real-life scenario. Current system is a perfect proof of concept example and can be a foundation for a future research and development. Real-time distributed data mining is not a trivial task and there are many natural obstacles, which are just impossible to neglect. As we figured out, data fragmentation is a huge challenge. Trade-offs between quality and consistency will always affect the result. Timestamp delays are able to confuse even the best algorithm ever. However, current thesis demonstrates that by targeting very specific problem in the particular domain area it is possible to solve many of those problems, which are true for general purposed algorithms.

By this research we have shown that existing technologies in a combination with innovative algorithms and fresh approaches are able to boost the development of

Location Based Services and shift the focus from current single-target user-centric applications to proactive multi-target systems.

References

[1] Michael Haupt. Data is the new oil, 5 2016.

[2] Gartner. Gartner's 2016 hype cycle for emerging technologies identifies three key trends that organizations must track to gain competitive advantage, 8 2016.

[3] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[4] Alexa Megan Sharp. *Incremental algorithms: solving problems in a changing world*. PhD thesis, Cornell University, 2007.

[5] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[6] Anil K Jain. Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8):651–666, 2010.

[7] Marcel R Ackermann, Marcus Märtens, Christoph Raupach, Kamil Swierkot, Christiane Lammersen, and Christian Sohler. Streamkm++: A clustering algorithm for data streams. *Journal of Experimental Algorithmics (JEA)*, 17:2–4, 2012.

[8] Paul S Bradley and Usama M Fayyad. Refining initial points for k-means clustering. In *ICML*, volume 98, pages 91–99. Citeseer, 1998.

[9] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: An efficient data clustering method for very large databases. In *ACM Sigmod Record*, volume 25, pages 103–114. ACM, 1996.

[10] Jonathan A Silva, Elaine R Faria, Rodrigo C Barros, Eduardo R Hruschka, André CPLF de Carvalho, and João Gama. Data stream clustering: A survey. *ACM Computing Surveys (CSUR)*, 46(1):13, 2013.

[11] Jianqiang Dong, Fei Wang, and Bo Yuan. Accelerating birch for clustering large scale streaming data using cuda dynamic parallelism. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 409–416. Springer, 2013.

[12] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[13] Chire. Illustration of dbscan cluster analysis, 10 2011.

[14] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: ordering points to identify the clustering structure. In *ACM Sigmod Record*, volume 28, pages 49–60. ACM, 1999.

[15] Wei Wang, Jiong Yang, Richard Muntz, et al. Sting: A statistical information grid approach to spatial data mining. 1997.

[16] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. *Automatic subspace clustering of high dimensional data for data mining applications*, volume 27. ACM, 1998.

[17] Tyler Akidau. The world beyond batch: Streaming 101, 8 2015.

[18] Jianjun Chen, David J DeWitt, Feng Tian, and Yuan Wang. Niagaracq: A scalable continuous query system for internet databases. In *ACM SIGMOD Record*, volume 29, pages 379–390. ACM, 2000.

[19] Ling Liu, Calton Pu, and Wei Tang. Continual queries for internet scale event-driven information delivery. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):610–628, 1999.

[20] Johannes Gehrke, Flip Korn, and Divesh Srivastava. On computing correlated aggregates over continual data streams. In *ACM SIGMOD Record*, volume 30, pages 13–24. ACM, 2001.

[21] Yunyue Zhu and Dennis Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 358–369. VLDB Endowment, 2002.

[22] Feng Cao, Martin Ester, Weining Qian, and Aoying Zhou. Density-based clustering over an evolving data stream with noise. In *SDM*, volume 6, pages 328–339. SIAM, 2006.

[23] Charu C Aggarwal, Jiawei Han, Jianyong Wang, and Philip S Yu. A framework for clustering evolving data streams. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 81–92. VLDB Endowment, 2003.

[24] Yixin Chen and Li Tu. Density-based clustering for real-time stream data. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2007.

[25] George S Almasi and Allan Gottlieb. Highly parallel computing. 1988.

[26] George F Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed systems: concepts and design*. pearson education, 2005.

[27] Miym. Distributed and parallel systems, 07 2009.

[28] Xiaobo Li and Zhixi Fang. Parallel clustering algorithms. *Parallel Computing*, 11(3):275–290, 1989.

[29] Domenico Talia. Parallelism in knowledge discovery techniques. In *Applied Parallel Computing Advanced Scientific Computing, 6th International Conference, PARA 2002, Espoo, Finland, June 15-18, 2002, Proceedings*, pages 127–138, 2002.

[30] Wooyoung Kim. Parallel clustering algorithms: Survey. *Parallel Algorithms, Spring*, 2009.

[31] Stefan Brecheisen, Hans-Peter Kriegel, and Martin Pfeifle. Parallel density-based clustering of complex objects. In *Advances in Knowledge Discovery and Data Mining, 10th Pacific-Asia Conference, PAKDD 2006, Singapore, April 9-12, 2006, Proceedings*, pages 179–188, 2006.

[32] Md. Mostofa Ali Patwary, Diana Palsetia, Ankit Agrawal, Wei-keng Liao, Fredrik Manne, and Alok N. Choudhary. A new scalable parallel DBSCAN algorithm using the disjoint-set data structure. In *SC Conference on High Performance Computing Networking, Storage and Analysis, SC '12, Salt Lake City, UT, USA - November 11 - 15, 2012*, page 62, 2012.

[33] Yaobin He, Haoyu Tan, Wuman Luo, Huajian Mao, Di Ma, Shengzhong Feng, and Jianping Fan. MR-DBSCAN: an efficient parallel density-based clustering algorithm using mapreduce. In *17th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2011, Tainan, Taiwan, December 7-9, 2011*, pages 473–480, 2011.

[34] Xiaowei Xu, Jochen Jäger, and Hans-Peter Kriegel. A fast parallel clustering algorithm for large spatial databases. In *High Performance Data Mining*, pages 263–290. Springer, 1999.

[35] Xin Sun and Yu Cathy Jiao. pgrid: Parallel grid-based data stream clustering with mapreduce. *Report*, 2009.

[36] Stephan Kulla. Intersection and union of two sets in a venn diagram, 04 2015.

[37] Dezydery Szymkiewicz. *Une contribution statistique a la géographie floristique*. Polskie Towarzy-stwo Botaniczne, 1934.

[38] Kathleen Hornsby and Max J. Egenhofer. Modeling moving objects over multiple granularities. *Ann. Math. Artif. Intell.*, 36(1-2):177–194, 2002.

[39] Patrick Laube, Stephan Imfeld, and Robert Weibel. Discovering relative motion patterns in groups of moving point objects. *International Journal of Geographical Information Science*, 19(6):639–668, 2005.

[40] Joachim Gudmundsson, Marc J. van Kreveld, and Bettina Speckmann. Efficient detection of motion patterns in spatio-temporal data sets. In *12th ACM International Workshop on Geographic Information Systems, ACM-GIS 2004, November 12-13, 2004, Washington, DC, USA, Proceedings*, pages 250–257, 2004.

[41] Marc Benkert, Joachim Gudmundsson, Florian Hübner, and Thomas Wolle. Reporting flock patterns. *Computational Geometry*, 41(3):111–125, 2008.

[42] Yanwei Yu, Qin Wang, Xiaodong Wang, Huan Wang, and Jie He. Online clustering for trajectory data stream of moving objects. *Comput. Sci. Inf. Syst.*, 10(3):1293–1317, 2013.

[43] Panos Kalnis, Nikos Mamoulis, and Spiridon Bakiras. On discovering moving clusters in spatio-temporal data. In *Advances in Spatial and Temporal Databases, 9th International Symposium, SSTD 2005, Angra dos Reis, Brazil, August 22-24, 2005, Proceedings*, pages 364–381, 2005.

[44] Opher Etzion and Peter Niblett. *Event processing in action*. Manning Publications Co., 2010.

[45] Wenhui Hu, Wei Ye, Yu Huang, and Shikun Zhang. Complex event processing in rfid middleware: A three layer perspective. In *Convergence and Hybrid Information Technology, 2008. ICCIT'08. Third International Conference on*, volume 1, pages 1121–1125. IEEE, 2008.

[46] Küpper Axel et al. *Location-based services: fundamentals and operation*. John Wiley & Sons, 2005.

[47] Paramvir Bahl and Venkata N. Padmanabhan. RADAR: an in-building rf-based user location and tracking system. In *Proceedings IEEE INFOCOM 2000, The Conference on Computer Communications, Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies, Reaching the Promised Land of Communications, Tel Aviv, Israel, March 26-30, 2000*, pages 775–784, 2000.

[48] Kamol Kaemarungsi and Prashant Krishnamurthy. Properties of indoor received signal for WLAN location fingerprinting. In *1st Annual International Conference on Mobile and Ubiquitous Systems (MobiQuitous 2004), Networking and Services, 22-25 August 2004, Cambridge, MA, USA*, pages 14–23, 2004.

[49] Sudarshan S Chawathe. Beacon placement for indoor localization using bluetooth. In *Intelligent Transportation Systems, 2008. ITSC 2008. 11th International IEEE Conference on*, pages 980–985. IEEE, 2008.

[50] Sandro Rodriguez Garzon and Bersant Deva. Geofencing 2.0: Taking location-based notifications to the next level. In *The 2014 ACM Conference on Ubiquitous Computing, UbiComp '14, Seattle, WA, USA, September 13-17, 2014*, pages 921–932, 2014.

[51] A Cupper, Georg Treu, and Claudia Linnhoff-Popien. Trax: A device-centric middleware framework for location-based services. *IEEE Communications Magazine*, 44(9):114–120, 2006.

[52] Louise Barkhuus and Anind K. Dey. Location-based services for mobile telephony: a study of users' privacy concerns. In *Human-Computer Interaction INTERACT '03: IFIP TC13 International Conference on Human-Computer Interaction, 1st-5th September 2003, Zurich, Switzerland*, 2003.

[53] Ordnance Survey. A guide to coordinate systems in great britain, 8 2016.

[54] Apache Flink. Apache flink: Scalable stream and batch data processing, 2014 - 2017.

[55] Alexander Alexandrov, Rico Bergmann, Stephan Ewen, Johann-Christoph Freytag, Fabian Hueske, Arvid Heise, Odej Kao, Marcus Leich, Ulf Leser, Volker Markl, Felix Naumann, Mathias Peters, Astrid Rheinländer, Matthias J. Sax, Sebastian Schelter, Mareike Höger, Kostas Tzoumas, and Daniel Warneke. The stratosphere platform for big data analytics. *VLDB J.*, 23(6):939–964, 2014.

[56] dataArtisans. Demo applications for apache flink^{TM} datastream, 2015.

[57] Sandro Rodriguez Garzon, Bersant Deva, Benoît Hanotte, and Axel Küpper. CATLES: a crowdsensing-supported interactive world-scale environment simulator for context-aware systems. In *Proceedings of the International Conference on Mobile Software Engineering and Systems, MOBILESoft '16, Austin, Texas, USA, May 14-22, 2016*, pages 77–87, 2016.

[58] Yu Zheng. T-drive trajectory data sample, August 2011.

[59] Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. Driving with knowledge from the physical world. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, pages 316–324, 2011.

[60] Jing Yuan, Yu Zheng, Chengyang Zhang, Wenlei Xie, Xing Xie, Guangzhong Sun, and Yan Huang. T-drive: driving directions based on taxi trajectories. In *Proceedings of the 18th SIGSPATIAL International conference on advances in geographic information systems*, pages 99–108. ACM, 2010.