UNIVERSITY OF PIRAEUS

DEPT. OF DIGITAL SYSTEMS

MSc IN INFORMATION SYSTEMS & SERVICES

BIG Data & ANALYTICS

CLUSTERING STREAMING DATA IN DISTRIBUTED ENVIRONMENTS BASED ON BELIEF PROPAGATION TECHNIQUES

ZISIS ARAMPATZIS, ME 1601

SUPERVISOR: MARIA HALKIDI

ATHENS

SEPTEMBER 2018

TABLE OF CONTENTS

1 List of Figures

2 List of Tables

3 Acknowledgements

I would like to express my sincere gratitude to my supervisor Prof. Maria Halkidi for her patience, motivation and immense knowledge. The door to Prof. Halkidi office was always open whenever I ran into a trouble spot or had a question about my thesis. Her guidance and feedback helped me throughout the writing of this thesis.

4 ΕΙΣΑΓΩΓΗ

Σκοπός της παρούσας εργασίας είναι να εξετάσει ένα σύγχρονο πρόβλημα στον τομέα της ανάκτησης πληροφορίας, την ομαδοποίηση ροών δεδομένων σε κατανεμημένο σύστημα. Αυτό το πρόβλημα ανήκει στη κατηγορία προβλημάτων μεγάλων δεδομένων, το οποίο σημαίνει ότι σε αυτά τα δεδομένα δεν μπορούν να εφαρμοστούν παραδοσιακές τεχνικές, λογισμικό ή βάσεις δεδομένων για να πιαστούν, να επεξεργαστούν και να αναλυθούν χωρίς μεγάλη καθυστέρηση και για αυτό τον λόγο πρέπει να χρησιμοποιηθεί παράλληλη επεξεργασία. Επιπλέον όταν αυτά τα δεδομένα είναι με την μορφή ροών δεδομένων υπάρχουν ακόμα περισσότερες προκλήσεις που πρέπει να αντιμετωπιστούν. Από την άλλη μεριά αν λυθούν αυτά τα προβλήματα, η ανάλυση των ροών δεδομένων δίνει πολλά πλεονεκτήματα, όπως το να υπάρχει εικόνα για τα δεδομένα σε πραγματικό χρόνο το οποίο μπορεί να βοηθήσει στην αντιμετώπιση διαφόρων καταστάσεων σε πραγματικό χρόνο. Η συνεχής και κατανεμημένη μορφή που παράγονται τα δεδομένα από πλήθος συσκευών μαζί με το μεγάλο μέγεθός τους και περιορισμούς όπως ο χώρος αποθήκευσης και ο φόρτος δικτύου καθιστούν ένα πολύ δύσκολο πρόβλημα την ομαδοποίηση ροών δεδομένων. Στην παρούσα εργασία, προσπαθούμε να επιλύσουμε το πρόβλημα χρησιμοποιώντας μια προσέγγιση δύο επιπέδων ομαδοποίησης. Στο πρώτο επίπεδο, καθώς τα δεδομένα έρχονται σε πολλούς κατανεμημένους κόμβους, σε κάθε περίοδο του χρόνου, κάθε κόμβος ομαδοποιεί τα δεδομένα και εξάγει την πιο χρήσιμη πληροφορία από τα δεδομένα (exemplars) η οποία θα σταλεί σε έναν κεντρικό κόμβο, για να εκτελέσει με την σειρά του το δεύτερο επίπεδο ομαδοποίησης για να εντοπίσει τα ολικά cluster από όλα τα δεδομένα που κατέφθασαν σε κάθε κόμβο. Οι exemplars που υπολογίστηκαν στον κεντρικό κόμβο, θα σταλούν πίσω σε κάθε κόμβο για να αναθεωρήσουν το βάρος του κάθε exemplar και αυτή η διαδικασία θα συνεχιστεί σε όλη την διάρκεια της ροής δεδομένων. [1]

5 Summary

This study tries to examine a recent problem of computer science and specifically in data mining field, the online clustering of distributed streaming. This problem belongs to big data and analytics area, which means that we can’t apply the traditional techniques, software or databases to capture, process and analyze this data with low-latency and we need massive parallelism. Moreover, when these big data techniques applied to streaming data more challenges are emerged. On the other hand, the analyzing of streaming data will give a lot of advantages like real time insights of the data that will help to respond in emerging situations. The sequential and distributed fashion of the data produced from a variety of devices combined with the volume of them and constraints such as communication and storage make a major challenge the clustering of streaming data. In our study we address the problem of distributed clustering using two level of clustering approach, in first level, batch of data arrives in many distributed nodes in each time slot and the nodes performs clustering in these data extracting the most significant representatives of the batch (exemplars) which will be forwarded to the central node which in turn performs the second level of clustering in order to identify global patterns in the data arrived from every node The algorithm that we will try to implement uses belief propagation techniques in a distributed environment. The exemplars will feed back to the nodes with the appropriately modified weight which reflect their global significance. We adopt belief propagation techniques in both levels to perform streaming clustering. [1]

6 Introduction to big data analytics

6.1 Big data

Big data analytics is a term used in datasets which size is beyond the ability of the traditional relational databases and software to capture, manage and processing with low-latency. Data that is unstructured, time sensitive or simple too large for traditional data mining techniques to uncover its insight and meaning of the underlying data, requires a different processing approach called big data, which uses massive parallelism. Three key concepts of big data are: volume, velocity and variety known as 3Vs.

6.1.1 3Vs

6.1.1.1 Volume

Volume is most associated with big data. The volume can be big in quantities of data that reach almost incomprehensive proportions. As we move forward, we are going to have more and more huge collections. For examples data that come from sensors, social media, mobile devices, web applications, etc. As a result, it is now not uncommon for large companies to have Terabytes and even Petabytes of data in storage devices and on servers. This data helps to shape the future of company and its actions.

6.1.1.2 Velocity

Velocity is how fast the data is created, stored, analyzed and visualized. In the past, when batch processing was common practice, it was normal to receive an update from the databases every night or every week. Computers and servers required substantial time to process the data and update databases. In the big data era, data is created in real-time or near real-time. With the availability of internet connected devices (Internet of things), wires or wired, machines and devices can pas-on their data the moment it is created.

6.1.1.3 Variety

With increasing volume and velocity comes increasing variety. This third “V” describes the huge diversity of data types. In the past, all data that is generated by organization was structured data, in columns and rows. Nowadays, most of data generated is unstructured. Data today comes in many different formats: Structured data, semi-structured data, unstructured data and even complex structured data. The wide variety of data requires a different approach as well as different techniques to store all raw data. [2]

6.1.2 Who uses Big Data

6.1.2.1 Banking

Banks have many sources that streaming data at any moment and want to find innovative ways to manage big data. For banks it is important to understand customers needs and boost their satisfaction and it is also very important to minimize any risk and fraud. Big data brings big insights, but it is also requiring financial institutions to stay one step ahead of the game with advanced analytics.

6.1.2.2 Education

Educators using big data techniques and getting insight of the data can make a significant difference in education systems, students and curriculums. By analyzing big data can find at-risk students, make sure that students make progress and can implement a better system for evaluation.

6.1.2.3 Government

Governments can apply analytics in their big data to deal with very difficult and major problems like traffic congestion criminality and managing utilities. But while there are many advantages to big data, governments must also address issues of transparency and privacy.

6.1.2.4 Health Care

One of the fields that have big data and needs to process and analyze them efficiently are the healthcare. Patient records, treatment plans, prescription information and many more data produced from many sources can be processed to get hidden insights that could improve patient care.

6.1.2.5 Manufacturing

Armed with insights that big data can provide, manufacturers can boost the quality of their products while simultaneously can minimize the waste of their resources. More and more manufacturers working in an analytics-based culture, which means they can solve problems faster and make more agile business decisions.

6.1.3 Big Data Use Cases

6.1.3.1 Customer Satisfaction

Many companies use big data to build a dashboard application that provides 360^0 view of their customers. These applications use data from internal and external sources, analyze it and present it to customer service, sales or marketing department in a way that helps them do their jobs.

6.1.3.2 Fraud Prevention

For credit cards holders, fraud prevention is one of the most familiar use cases for big data. Even before big data analytics, credit cards issuers used rules-based system to help them flag potentially fraudulent transactions. For example, if a credit card were used to rent a car in America but the customer lives in Europe, a customer service agent might call to confirm that the customer was on vacation and that someone hadn’t stolen the card. Thanks to big data analytics and machine learning, today’s fraud prevention systems are better at detecting criminal activities and preventing false positives.

6.1.3.3 Security Intelligence

Organizations are also using big data analytics to help them prevent hackers and cyberattacks. An IT department generates enormous amount of log data. In addition, cyber threat intelligence data is available from external sources, such as law enforcement or security providers. Many organizations use big data techniques to help them aggregate and analyze all of this internal and external information to help them detect, prevent and mitigate attacks. Today with Internet of Things and smart homes and devices such as sensors (temperature, humidity) is crucial to detect any uncommon patterns in real time using streaming analytics techniques to stop attackers.

6.1.3.4 Data Warehouse Offload

One of the easiest and most common ways for an organization to start using big data technologies is to remove some of the burden from its data warehouses. It is common for the most companies to have a data warehouse that facilitates their business intelligence. In most cases data warehouses technology tends to be very costly to purchase and run and as the business leaders ask for more reports and insights from their BI teams, the data warehouses solutions haven't always been able to provide the desired performance. To deal with this problem many companies use open source big data technologies to complements their data warehouses. Big data open source solutions provides better performance while reducing licensing fees and other costs.

6.1.3.5 Price Optimization

Many enterprises use big data analytics to optimize their pricing. For any company the goal is to maximize its income. If the price is too high, they will sell to fewer costumers, but if the price is too low they leave money on the table. Big data analytics allows companies to see which price bring better results under historical market conditions. Businesses that are more sophisticated apply also dynamic pricing to maximize their income.

6.1.3.6 Recommendation Engines

One of the most common use cases of big data is the recommendation engine. When someone buys online there are a lot of similar items that suggested, these recommendations arise from the use of big data analytics which analyzed historical data. These recommendations engines become so common on the Web that many customers now expect them when they are shopping online. Organizations that haven't taken advantage of their big data in this way, lose customers.

6.1.3.7 Social Media Analysis and Response

Data produced from social media is one of the most obvious examples of big data. Today, companies try to analyze what people are saying about them in the social media and response appropriate. As a result, many enterprises are investing tools to help them monitor and analyze social platforms in real-time.

6.1.3.8 Preventive Maintenance and Support

Businesses in manufacturing, energy, construction, agriculture, transportation and similar sectors of the economy use big data analytics to improve equipment maintenance. As the Internet of Things (IoT) become a reality, factories and other facilities that use expensive equipment are deploying sensors to monitor and analyze that equipment and transmit data over the internet. Then they analyze these data in real time to detect when a problem is about to happen. This way they can perform preventive maintenance that may help to prevent accidents or costly line shutdowns.

6.1.3.9 *Internet of Things*

As in the preventive maintenance example, they are using sensors to collect data that they can then analyze to find insights that will lead to actions, companies may track customers or product movement, monitor the weather or keep an eye in security cameras. The number of the ways in which analytics can e applied to IoT solutions seems to be endless. [2]

6.2 Cloud

Big data technologies, tools and frameworks used in big data are highly connected with Cloud Computing. Most of the companies have their big data stack in a cloud clusters which give them many capabilities such as horizontal scaling and easy configuration and this make feasible and easy the processing of big data.

Cloud Computing is a model for enabling ubiquitous, convenient, on demand access to shared pool of configurable computing resources such as networks, servers, application and services that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics. Three service models and four deployment models. [3]

6.2.1 Essential characteristics

6.2.1.1 *On-demand self-service*

A consumer can use as computing resources as he wants without any interaction with his provider. He can also setup his environment as he wants. Like server time and network storage.

6.2.1.2 *Broad network access*

Cloud resources are available through network and accessed through standard mechanisms, these resources can be used from many different devices like mobile phones, laptops, servers, tablets and workstations.

6.2.1.3 *Resource pooling*

The providers' computing resources (virtual and physical) can be used from many consumers which can be assigned and reassigned dynamically according to consumer demand. Consumers doesn't have any control of the location of the resources just only in high level like country, state or datacenter. Examples of resources include storage, processing memory and network bandwidth.

6.2.1.4 *Rapid elasticity*

Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly. For the consumer the capabilities of provisioning often appears to be unlimited and can be appropriated in any quantity at any time.

6.2.1.5 Measured service:

Cloud systems automatically control and optimize resource use. Resource usage can be monitored, controlled, and reported, in this way there is transparency between the consumer and the provider. [3]

6.2.2 Service Models

6.2.2.1 Software as a Service (SaaS)

Provider's application running in a cloud infrastructure can be accessed from consumer from various client devices such as web browsers or program interfaces. The consumer does not know anything regarding the infrastructure the application runs and cannot manage or control any of them like network, servers, operating systems, storage or even individual application capabilities.

6.2.2.2 Platform as a Service (PaaS)

Consumer can deploy his applications onto the cloud infrastructure using programming languages, libraries, services and tools supported from the provider. The consumer does not manage or control the underlying cloud infrastructure like network, server, operating systems or storage but can controlled the deployed applications and possible the configuration settings for the application.

6.2.2.3 Infrastructure as a Service (IaaS)

Consumer can provision processing, storage networks and other fundamentals computing resources, to run software even an operating system. The consumer does not manage or control the underlying cloud infrastructure but has control over the operating systems, storage and deployed applications and possibly limited control over the networking components (e.g. host firewalls). [3]

6.2.3 Deployment Models

6.2.3.1 Private Cloud

The cloud infrastructure is provisioned exclusive for use only from one organization which serves multiple consumers. It is owned, managed and operated from the organizations, a third party or a combination of them, and it may exist on or off premises.

6.2.3.2 Community Cloud

The cloud infrastructure is provisioned for exclusive use of a community of consumers from organizations that may have the same concerns regarding (e.g. security requirements, policy and compliance considerations). It may be owned, managed and operated by one more organization, a third party, or a combination of them and it may exist on or off premises.

6.2.3.3 Public Cloud

The cloud infrastructure is provisioned for open use of public. It may be owed, used, managed or operated by a business, academic or government organization or some combination of them. It exists on premises of cloud provider.

6.2.3.4 Hybrid Cloud

It is a combination of two or more cloud deployment models (private, community or public) that remain unique entities but are bound together to enable data and applications portability (e.g. cloud bursting for load balancing between clouds). [3]

7 Streaming data

Streaming data is data that is generated continuously by thousands of data sources, which typically send the data records simultaneously. Streaming data includes a variety of data such a log files, information of social media, geospatial services, in-game player activity, mobile applications data and many more. This data needs to be processed sequentially and incrementally on a record bases or over sliding time windows and used for a variety of analytics including correlations, aggregations, filtering and sampling. Information derived from such analysis gives companies visibility into many aspects of their business and customer activity such as service usage, server activity, geo-location of devices and enables them to respond in emerging situations. For example, business can truck changes in public sentiment on their brands and products by continuously analyzing social media streams and respond in a timely fashion as the necessity arises.

In batch processing, data are available in databases or any other storage when we want to process it and mine useful information and insight of it. In streaming analytics if we not manage to store or proceeded data immediately, then it is lost forever. Moreover, we shall assume that data arrives so fast that we cannot store them in traditional databases. To deal with this issue data stream algorithms, try to summarize data for every batch that arrives in each timeslot. It is also very important to filter the data to get only the useful data reducing in this way the volume of it. [4]

7.1 Comparison between Batch Processing and Stream Processing

Before dealing with streaming processing, it is worth to comparing the difference with batch processing. Batch processing can be used to compute queries of different datasets, it usually computes results from all the data and makes complex and in deep analysis. This comes in contrast with stream processing, where ingesting a sequence of data is required incrementally updates metrics and reports in response to every batch of data that arrived in a time window. It is mainly used in monitoring and response functions. Batch processing needs minutes or hours, on the other hand streaming processing needs seconds, this happens because in batch processing we process and analyze big sets of data and in streaming processing we just process individual micro batches of a few records, moreover in most cases in batch processing we apply algorithms and we make very intricate computation instead of just aggregations and summaries and simple metrics that we make in streaming processing.[4]

7.2 Benefits of Streaming Data

Streaming data processing is beneficial in most scenarios where new, dynamic data is generated on a continual basis. It applies to most of the industry segments and big data use cases. Companies generally begin with simple applications such as collecting system logs. Then, these applications evolve to more sophisticated near-real-time processing. Initially, applications may process data streams to produce simple reports, and perform simple actions in response, such as emitting alarms when key measures exceed certain thresholds. Eventually, those applications perform more sophisticated forms of data analysis, like applying machine learning algorithms, and extract deeper insights from the data. [4]

7.3 Motivating Examples and Uses Cases

Examples motivating a data stream system can be found in many application domains including finance, web application, security, networking, transportation vehicles, and sensor monitoring.

- Modern security applications often apply sophisticated rules over network packet streams. For examples in internet of thing that many devices are connected to internet we can monitor and profiling the device behavior and detect attacks.
- Large web sites monitor web logs (clickstreams) online to enable applications such as personalization, performance monitoring and load-balancing.
- Sensors in transportation vehicles, industrial equipment, and farm machinery send data to a streaming application. The application monitors performance detects any potential defects in advance and places a spare part order automatically preventing equipment down time.
- A financial institution tracks changes in the stock market in real time, computes value-at-risk, and automatically rebalances portfolios based on stock price movements.
- A financial institution tracks changes in the stock market in real time, computes value-at-risk, and automatically rebalances portfolios based on stock price movements
- An online gaming company collects streaming data about player-game interactions and feeds the data into its gaming platform. It then analyzes the data in real-time, offers incentives and dynamic experiences to engage its players.
- A real-estate website tracks a subset of data from consumers' mobile devices and makes real-time property recommendations of properties to visit based on their geo-location. [4]

7.4 Challenges of Streaming processing

There are two main challenges of streaming processing. The storage layer and the processing layer. As the data arrives with high rate if we store all the data to a storage this storage will be full very soon, moreover this storage must be reliable to ensure that we will not miss data, inexpensive and support efficient read and writes actions. The processing layer should be very efficient to be able to process data fast and notify the storage layer to delete the data that is no longer needed. Moreover, both layers should be support scalability, fault tolerance and data durability. [5]

7.5 The Stream Data Model

A typical stream processing management system is presented in the image below.

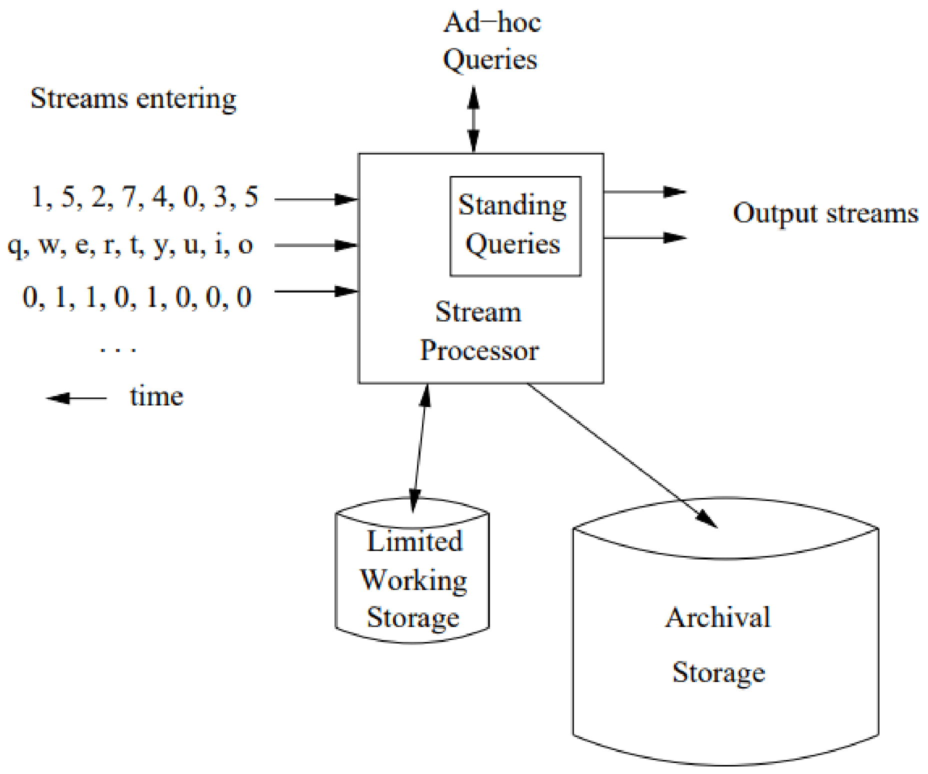

Figure 1: Stream data management system

We can view a stream processor as a kind of data management system. This system can handle many streams simultaneously and every stream can provide data at its own schedule and moreover the streams have also different rates, data types and time between data it can be constant or not. All the aforementioned facts make clear the challenge that a central data management system has to deal with that a traditional database management system doesn't have. The latter system controls the rate of the data which read from disk and therefore never has to worry about data getting lost as it attempts to execute queries.

In most cases this data can be stored to a distributed data storage that supports horizontal scaling and stores huge amount of data. In this case is not efficient to get insights for the streaming data from this storage and in many cases it might also not possible. There is also a working store, into which a summary of streaming data is stored and as we can think this space is limited. [5]

7.6 Stream resources

Social media, Internet of things, sensors, surveillance systems, online internet traffic, mobile applications and RFID data are some examples of sources that produce huge amount of data every second that becomes available to a stream system and we want to take insights of this data in real time to make a decision.

7.6.1 Sensor Data

To understand the rate of sensors and the need of efficiently management of the data that produce, we will give an example of a sensor that is in the ocean. This sensor sends the temperature in a constant time interval, the rate of the data the system must handle is known and steady but what happens if we upgrade this sensor to send the GPS location in order to know the height of the see every time when the position of sensor changes? In this case the sensor will increase its rate and the time interval is not constant and known. Moreover, one sensor example is not a real case scenario, in a real case scenario we may need millions of sensors to know better the ocean an get useful metrics, this increase very much the data and as we definitely need to think what data to keep in a working storage and what can only be archived. In this scenario we must think also how to handle and distinguish the data send from a broken sensor, if a sensor is broken a send with high rate data to stream system, this data should be filtered and discarded from the system and not to be stored in none of the storage that we have.

7.6.2 Image Data

Satellites often send down to earth streams consisting of many terabytes of images per day. Surveillance cameras produce images with lower resolution than satellites, but there can be many of them, each producing a stream of images at intervals like one second. London is said to have six million such cameras, each producing a stream.

7.6.3 Internet

Nowadays internet is everywhere, many traditional daily actions in our days are performed through internet. You can buy everything from internet or book an apartment and even to call a taxi or send a message to your friends and pay your bills, you can state your opinion and many people informed about it and many more actions through your mobile phone or your personal computer. All these new modern innovative applications combined the easy internet access that everyone has lead to many data through the internet. All these data if processed properly can be very useful for companies. Moreover, fraud detection and illegal actions can be prevented if we analyze the data and the patterns of the data fast as many money transferred through internet every day. [5]

8 Distributed approach of StreamAP

8.1 Background: Affinity propagation

Affinity propagation is a clustering method originally proposed by Frey and Dueck [5]. Affinity propagation algorithm consider all items as potential centers of a cluster. Each item is considered as a node in a network and AP recursively transmits real-valued messages along edges of the network until a set of centers and clusters emerged. These messages updated in each iteration and they reflect the current affinity that a data item must select another data item as an exemplar at that iteration. AP tries to find a set of exemplars and each item to be assigned in one of these exemplars and the sum of similarities between data items and exemplars is maximized. Moreover, AP doesn't allow an item to be self-exemplar. Assume that X= $\{x_1, x_2 \ldots x_n\}$ is a set of distinct items and $S(x_i, x_j)$ is the similarity between items i and j with $i \neq j$. AP tries to maximize an energy function defined as:

$$\varepsilon[\varphi] = \sum_{i=1}^{n} S(x_{i,} \varphi(x_{i,})) - \sum_{i=1}^{n} x_i [\varphi] \qquad (1)$$

With $x_i[\varphi] = \infty$, if $\varphi(\varphi(x_i)) \neq \varphi(x_i)$ and 0 otherwise. The first term in (1) is the overall similarity between data items and their exemplars and the second express the constraint that if a data item is selected as exemplar by some data items, it should be its own exemplar. This is an optimization problem which is solved using message passing techniques. The AP algorithm takes as input the similarities between items. Similarity $S(x_i, x_j)$ describe how well the item j is suited to serve as exemplar the item i. For $i = j$, the similarity of item i, $S(x_i, x_i)$ indicates the possibility of item i to be chosen as exemplar and it is called preference. Usually similarity is set to a negative squared error:

$$\mathrm{S}(x_i, x_j) = - || x_i - x_j ||^2$$

The number of clusters is influenced from preferences values, but it is also coming out of the message passing procedure. There two types of exchange messages:

1. Responsibility message $\mathrm{r(i,k)}$. This shows how well item *k* serves as exemplar for item i and this message is sent from data item i to candidate exemplar *k.* This value computed from the following function:

$$r(i,k) = S(x_i\,, x_k) - \max_{k',k'\neq k} \{a(i,k') + S(x_i, x_{k'})\} \qquad (2)$$

Where $a(i,k')$ is the availability message presented below. This message shows how appropriate it would be for item i to select item k' as its exemplar. If $k = i,$ the responsibility message is adjusted as:

$$r(k,k) = S(x_k\,, x_k) - \max_{k',k'\neq k} \{\, S(x_k, x_{k'})\} \qquad (3)$$

2. Availability message, $a(i,k')$ that a candidate exemplar sends to data item i. The value of $a(i,k')$ is updated as:

$$a(i,k) = \left\{0, r(k,k) + \sum_{i',i'\notin i,k} \max\{0, r(i',k)\}\right\} \qquad (4)$$

For $k = i,$ the availability message reflects accumulated evidence that item k is an exemplar and is adjusted by :

$$a(k,k) = \left\{\sum_{i',i'\notin k} \max\{0, r(i',k)\}\right\} \qquad (5)$$

At each iteration the assignments of items to exemplars are defined as, $\varphi(x^i) = \arg\max k\,\{r(i,k) + a(i,k)\}$, where $\varphi(x_i)$ denotes the exemplar for data item x_i. The message passing procedure stops after a specific number of iterations, or after cluster structure does not significantly change for a given number of iterations. [1]

8.2 Streaming data and clustering models

Streaming data arrives in batches continuously in a specific time interval duration T referred as *epochs.* At each epoch $i, i = 1{,}2\ \ldots$, a new batch of data items arrives $b_i = \{d_1^i, \ldots, d_M^i\}$, where M is the number of items in every batch, we consider the M as fixed value for every batch that arrives because of memory and storage space constraints in nodes that processed the batches. The batch of data that arrives at epoch i has timestamp T_i and in case the data flow is periodic with period T, the the timestamp is $T_i = iT$, otherwise it depends on the batch arrival pattern. In, this work we consider the batch arrival pattern as fixed. The processing of data begins in the end of each epoch and after the data discarded of the memory to free the space for the data that will come to the next epoch. In streaming clustering, it is possible to a new epoch new clusters appear, and old ones disappear as the data evolve during the time. The goal is to identify clusters of data and study the evolution of the clusters across epochs.

To keep the historical data information across the epochs and take them into consideration during the clustering process we adopt a damped window model for data, according to which the weight of data decreases during the epochs according an attenuation function $f(t) = 2^{-\lambda t}$, where $\lambda > 0$ is the decay factor, that means that the weight at epoch k of a data item x_i with timestamp $T_i < T_k$ will be given by:

$$w(x_i, T_k) = 2^{-\lambda(T_K - T_\iota)} \qquad (6)$$

In this way we can control the impact of historical data in clustering process. For example, if we increase values of λ , we decrease the impact of historical data compared to the most recent ones.

Because of memory constraints we cannot assume that data items will not be available in next epochs, for this reason we want to keep information regarding this data for the next epoch in order to use this information in clustering process in order to have a better overview of the data and not only for the data that arrives at current epoch but also for the historical data taking always into account the weight of these. For example, at epoch i we would like to have a representation for all historical data that arrived at epoch $i' < i$ so that, together with new data that arrives at current epoch to proceed with the clustering process and evaluate the data evolution and detect any changes in this epoch compared to the history of stream process.

As we have already defined the $b_i = \{d_1^i, \dots, d_m^i\}$ as the batch of m data items that arrived at epoch i, we denote E_{i-1} as the representation of the historical data arrived up to epoch $i - 1$. At each epoch i we want to come up with an update rule of the form:

$E_i = f(b_i, E_{i-1})$, where $f(\cdot)$ is an appropriate defined mapping operator on current batch b_i and the previous representation E_{i-1} .

In this work we consider that E_{i-1} must consist of appropriately weighted exemplars of clusters defined in previous epochs, an exemplar represents all the data of the cluster which select him as their representative, in this way the exemplars provide a synopsis of underlying data. The weight of each exemplar e_i is the sum of the weights of the items that select this exemplar to represent them.

DEFINITION 1:

Weight of exemplar (or cluster). Suppose that data items $\{x_j\}_{j=1}^n$ with timestamps $\{T_j\}_{j=1}^n$ have selected e_i as their exemplar. The weight of e_i at epoch i is given by:

$$w(e_i, T_i) = \sum_{j=1}^{n} w(x_j, T_i - T_j) \qquad (7)$$

At each epoch i, the set of data items which used from clustering algorithm is the union of the current data that arrives in this epoch with the E_{i-1} that contains weighted exemplars coming from previous epoch $i-1$. We can easily prove that the weight of an exemplar can be calculated incrementally at each epoch. Specifically, assume that $w(e_k, T_{i-1})$ is the weight of exemplar e_k at epoch $i-1$ and $\{d_1^i, \ldots, d_m^i\}$ is the subset of data items that arrived during the next epoch, i (with timestamp T_i) and have also selected e_k as their exemplar. Based on (6) the weight of data items arrived at the current epoch i is: $w(d_j^i, T_i) = 1, j = 1, \ldots, m$. If $T_i - T_{i-1} = 1$, the weight of e_k defined in (7) will be updates as follows:

$$w(e_k, T_i) = 2^{-\lambda} \cdot w(e_k, T_{i-1}) + m \qquad (8)$$

where m is the number of items that arrives at epoch i and select e_k as their exemplar which is denoted by $C(e_k)$.

For each exemplar we compute also the sum of weighed dissimilarities between him and all the data items that select him as their exemplar and it can be given from:

$$\overline{D}(C(e_k), T_i) = \sum_{x \in C(e_k)} w(x, T_i) \cdot \|x - e_k\|^2 \qquad (9)$$

It can be proved that $\overline{D}(C(e_k), T_i)$ can be maintained incrementally, i.e.

$$\overline{D}(C(e_k), T_i) = 2^{-\lambda} \cdot \overline{D}(C(e_k), T_{i-1}) + \sum_{d_j^i \in C(e_k)} \left\|d_j^i - e_k\right\|^2 \qquad (10)$$

For each exemplar the following vector is forwarded to the next epoch: $i+1$: $[e_k\ ,\ w(e_k, T_i\ ),\ \overline{D}(C(e_k), T_i)\ ]$.

At each epoch the similarities between data items are properly adjusted, so that patterns of knowledge extracted from previously arrived data are also exploited in data clustering, our algorithm use these similarities and defines new clusters using the message passing procedure. In the sequel we present how we update these similarities, between new items and previous exemplars, which is the key for transferring salient data features from one epoch to the next one.

DEFINITION 2.1:

Preference of exemplars from a previous epoch. The preference of an exemplar e_k at epoch $i+1$ will be defined based on its preference in the previous epoch i and the statistics of its cluster in i. The preference of e_k is updated as follows:

$$S_{i+1}(e_k, e_k) = S_i(e_k, e_k) \cdot \left(1 - \frac{w(e_k, T_i)}{\sum_j w\left(e_j, T_i\right)}\right) - var(C(e_k), T_i) \qquad (11)$$

Where $var(C(e_k), T_i) = \frac{\bar{D}(C(e_k), T_i)}{w(e_k, T_i)}$ is the weighted variance of cluster $C(e_k)$ which captures dissimilarities between the exemplar and the other data items in its cluster. At every epoch the variance of a cluster changes, thus each of previously selected exemplars gains advantage from the other data items of the current epoch which is analogous of the weight of the data items that selected it, minus the variance of its cluster.

DEFINITION 2.2:

Preference of current data items. The preferences of data items that arrived at current epoch $i+1$ are defined as follows:

$$S_{i+1}\left(d_j^{i+1}, d_j^{i+1}\right) = \frac{\sum_{x \in (b_{i+1} \cup E_i), x \neq d_j^{i+1}} S_{i+1}\left(x,\ d_j^{i+1}\right)}{|b_{i+1} \cup E_i|} \qquad (12)$$

Where $|b_{i+1} \cup E_i|$ is the number of data under analysis (i.e. exemplars from the previous epoch and the current data items).

DEFINITION 2.3:

Similarities between an exemplar from a previous epoch and a current data item.

At each epoch $i+1$, the similarities between an exemplar e_k from a previous epoch and a current data item are defined as follows:

$$S_{i+1}\left(d_j^{i+1}, e_k\right) = -\left\| e_k - d_j^{i+1} \right\|^2 \qquad (13)$$

$$S_{i+1}\left(e_k, d_j^{i+1}\right) = -w(e_k, T_i) \cdot \left\| e_k - d_j^{i+1} \right\|^2 \qquad (14)$$

Equation (13) show how possible is an old exemplar to be selected as exemplar for a current data item and on the other hand in equation (14) an old exemplar will evaluate the suitability of a current data item to be its exemplar based on their similarity and the significance of its cluster. This is due to the fact that an exemplar is not a single item but a representative for many items (the data of its cluster). Then the assignment of an exemplar e_k to a new one means that all the data that have previously selected e_k as their exemplar now must be assigned to the new one.

DEFINITION 2.4:

Similarity between exemplars coming from previous epoch.

If e_k and e_m are two old exemplars of the previous epoch, their similarity in the epoch $i + 1$ will be defines as:

$$S_{i+1}(e_k, e_m) = -w(e_k, T_i) \cdot \| e_k - e_m \|^2 \qquad (15)$$

As an exemplar is the representative of all data items in its cluster, we assume that is approximately corresponds to a set of identical copies of data items n_k (where n_k denotes the number of items that have selected e_k as their exemplar). Then the similarity of exemplar e_k to an exemplar e_m should reflect the similarity of the respective data cluster (i.e. n_k copies) to e_m. The impact of every data in similarity is analogously to its weight, the sum of similarities of n_k copies to e_m is: $w(e_k, T_i) \cdot \| e_k - e_m \|^2$.

DEFINITION 2.5:

Similarity between a pair of data items arrived at the current epoch, T_{i+1}. [1]

The similarity will be given by:

$$S_{i+1}(d_j^{i+1}, d_k^{i+1}) = -\left\|d_j^{i+1} - d_k^{i+1}\right\|^2 \qquad (16)$$

8.3 Clustering Streaming Data with Belief Propagation Techniques

StreamAP algorithm is a variation of the AP algorithm which handles sequence of streaming data in online fashion under limited memory as imposed by streaming applications. At each batch of data items arrive, and representatives of these data items are forwarded to next epoch, both the data items of the current epoch and the representatives of the previous epoch (exemplars) are used in clustering procedure. Once a batch of data items arrives, the similarity matrix is updated based on the streaming model that we presented in the previous section. The new clusters are defined from the AP algorithm and forwarded to the next epoch [1]. The proposed algorithm *StremAP* involves the following steps:

STEP 1:

When the first batch of data arrives the AP algorithm compute the clusters of current data items and initialize the process.

While the stream flows,

STEP 2:

For exemplar of the current epoch,

- Vector $[e_k\ ,\ w(e_k, T_i\ ),\ \bar{D}(C(e_k), T_i)\ ]$ is forwarded to the next epoch.
- The preferences of each exemplars are updated based on the (11)

STEP 3:

At the next epoch $i+1$ a new batch of data items arrives, these data items with the exemplars of the previous epoch is the set of data items that will be used for clustering and are referred as $b_{i+1} \cup\ E_i$. The similarity matrix of data items updated. Specifically:

- $\forall\ d_j \in b_{i+1}$, preferences $S_{i+1}\left(d_j, d_j\ \right)$ are updated based on (12).
- $\forall\ d_i\ , d_j \in$ as $b_{i+1} \cup\ E_i$, similarities $S_{i+1}\left(d_i, d_j\ \right)$ are adjusted based on (13) – (15).

STEP 4:

The extended version of AP is applied to $b_{i+1} \cup E_i$. This approach makes feasible to summarize the most significance information regarding data and clustering of historical data and use it in every epoch in order to monitor the data evolution and changes in clusters while new batches arrive. The results only depend of data similarities, while the decision maker may control the impact of historical data to the current data clustering through decay factor λ. [1]

8.4 Distributed Data Stream Clustering Approach

In a distributed system which contains many remote sites (nodes) and a central node (CS) we will try to apply a two-level clustering using the *StreamAP* algorithm [1]. The *StreamAP* algorithm runs in every node and receives batches of data items at each epoch and computes the clusters of them and define a local clustering model, C_i^t . The local clustering of node s_i, is described by its exemplars and their respective weights, that is $C_i^t = \{\left(le_1^i, w(e_1, T_i\ )\ \right), \dots, (le_k^i, w(e_k, T_i\ )\ )\}$. The local clustering model from every node is sent to the CS which aggregates them to define an updated global clustering. The updated global clustering is sent back to the remote sites so that they adjust their local exemplars. In this way the remote sites obtain a holistic view of the data in the entire network through the global clustering that they get form CS. The main steps of the proposed distributed *StreamAP* algorithm are:

STEP 1:

StreamAP runs at remote sites. Let C_i^t be the clustering model of site s_i at epoch t.

STEP 2:

The remote sites send their local clustering to CS.

STEP 3:

The CS receives local clustering model from each node and runs *StremAP* to update global clustering model from define in previous epoch. Let the global clustering be $\hat{C}^t$ at epoch t and $\{ge_1^t, \dots, ge_m^t\}$ be the respective set of selected exemplars. We note that the global exemplars can be either exemplars from previous epoch or one of the exemplars arrived in the current epoch from remote sites.

STEP 4:

The weight factor of global exemplars at CS is defined as $fw(ge_i^t) = \frac{w(ge_i^t,t)}{\sum_k w(ge_i^t,t)}$. It indicates the global significance of each exemplar ge_i^t with respect to the weight of points that have selected it as their exemplar.

STEP 5:

CS provide feedback to remote sites, forwarding them the global clustering and respective weight factors, that is, it sends the vector $[(ge_1^t, fw(ge_1^t)), \dots, (ge_m^t, fw(ge_m^t))]$.

STEP 6:

Once remote site s_i receives the feedback from CS, it updates the weights of its exemplars that belong to the intersection of the set denoting its local clustering C_i^t and the set denoting the global clustering. [1]

That is, $\forall\, le_j^i \in C_i^t, if\ le_j^i \in \hat{C}^t \cap C_i^t\ then\ w(le_j^i, t) = w(le_j^i, t) \cdot (1 + fw(ge_k^t))$, where $ge_k^t \equiv le_j^i$.

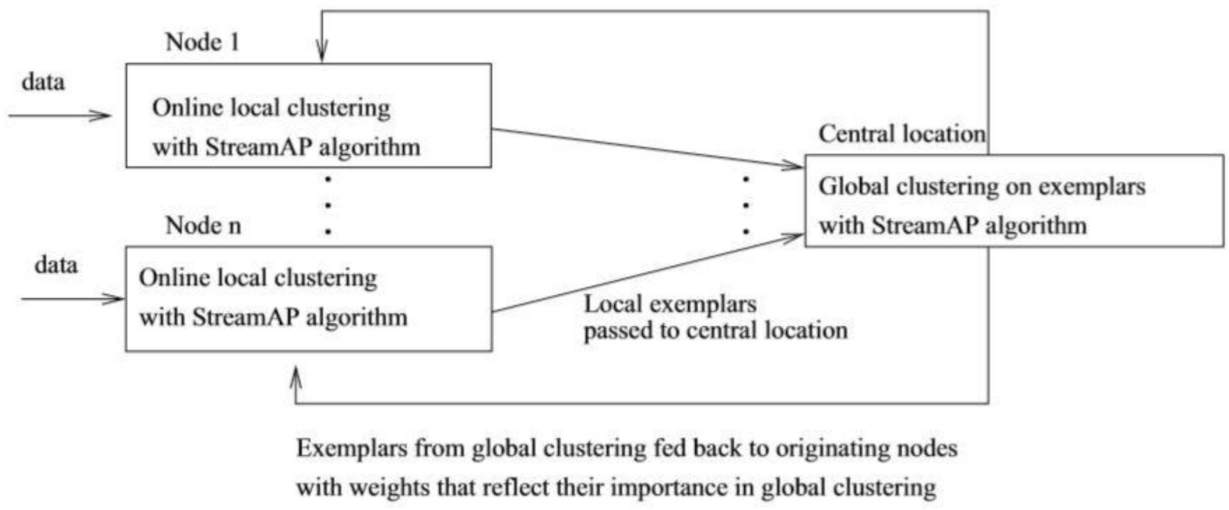
Node 1
data
Online local clustering with StreamAP algorithm
Central location
Global clustering on exemplars with StreamAP algorithm
Node n
data
Online local clustering with StreamAP algorithm
Local exemplars passed to central location
Exemplars from global clustering fed back to originating nodes with weights that reflect their importance in global clustering

Figure 2: Architecture of the proposed clustering approach for distributed streaming data

9 Implementation

During this work we implemented "sensor" scripts to produce data to test the implementation of our algorithm. We also implemented Central Site (CS) and Remote Sites (Nodes) which use the same source code but with different command line arguments to differentiate their behavior, in these applications we also implemented a visualization class to help us evaluate the results of our algorithm in real time.

9.1 Tools and libraries

9.1.1.1 Python

As programming language, we used PYTHON 3.5 [7], we concluded to this programming language as it provides a lot of benefits. Below we describe the reasons that lead us to choose it:

- *Python for data science.* Python is the preferred language from data scientists as it has many libraries with algorithms used in data science, one of these libraries we used to our work.
- *Smooth learning curve.* Python is one of the easiest programming languages and these help people who don't have any previous programming experience to use it.
- *Fast development.* The easiness and the syntax of Python increase the productivity of the final user. In the current work we want to produce the code rapidly to have time for evaluation and to focus more on the algorithm instead of how to write it.
- *Less hardware resources.* Instead of other programming languages that use virtual machines which consume a lot of memory, python doesn't need an this make it suitable for running in small devices such as possible devices that we will run the Remote Sites (Nodes).
- *Community.* As we already mentioned we want to have fast development of the algorithm and it will very difficult if we stuck in coding, for this reason we choose a mature programming language with great community to avoid such difficult situations.
- *Integrated development environment (IDE).* PYTHON has many IDEs and great tools for debugging to help you produce quality and fast code. Python ecosystem is really great, this is also a good reason to choose PYTHON.

9.1.1.2 Scikit-Learn

Scikit-learn [8] is an opensource software library largely written in PYTHON apart from some parts written in CYTHON [9] to achieve performance. It is a simple and efficient tool that is used in data mining, machine learning and data analysis and it implements various regression, clustering, and classification algorithms. In the current work we use the *Affinity Propagation* algorithm which is part of the *StreamAP.*

9.1.1.3 RabbitMQ

In the initial proposed *StreamAP* algorithm we consider a set of distributed nodes which communicate directly with a central location, the decision maker. In the current work and as an improvement we add a message broker (RabbitMQ) [10] between CS, nodes and sensors. The benefits of using a message broker instead of a direct socket communication are many. Below we describe the most significant reasons.

Better Performance

One of the most significant benefits of using a message broker, is that it provides better performance. Message queues enable asynchronous communication which means that the endpoints that are producing and consuming messages don't interact direct each other, but with a queue. Producers can add requests to a queue without waiting the consumer to processed them. Consumers process messages only when they are available. No component in the system is ever stalled waiting for another, optimizing the data flow.

Increased Reliability

Queues make data persistent and reduce the errors that happens when different parts of the system go offline. If one part of the system is ever unreachable, the other can continue to interact with the queue. The queue itself can also be mirrored for even more availability. For example, if the CS node for any reason is unreachable, the nodes they will continue sending their local clustering to message broker and when we the CS node go live again it will consume the results of nodes.

Scalability

Message queues make it possible to scale when is needed, when workloads peak, multiple instances of the application can all add requests to the queue without risk of collision. Producers, consumers and queues can all grow and shrink on demand.

Decoupling

Message queues remove dependencies between components and significantly simplify the coding of decoupled applications. Software components aren't weighed down with communications code and can instead be designed to perform a discrete business function.

Monitoring

Message queuing systems enable you to monitor how many items are in a queue, the rate of messages and other very useful statistics. This is very useful for monitoring your system and it is also useful when you implement your system in order to make an easy troubleshooting regarding communication issues. [11]

Finally it is very important to mention that we just use the RabbitMQ for its easiness during our implementation. Maybe there are most suitable message brokers for a production environment, like *KAFKA* [12] which is a very common message broker for streaming application or *ZeroMQ* [13] which is a very good choice for IoT systems, but the evaluation of the most suitable message broker regarding our system was out of scope of this work.

9.1.1.4 Pickle

When we want to send python items “through the wire” we must serialize and de- serialize them on the opposite side without losing their metadata and their schema. We don’t just want to send raw data, but we want to send lists, arrays, dictionaries, classes, instances of classes, etc. In our work we use a very common PYTHON object serialization library, the pickle library [14]. When nodes finish their local clustering in essance they have a list of exemplars. Then they send this list to CS using pickle to serialize it and CS with pickle de-serialize them. All the information exchanges between CS and nodes is serialized using pickle. We concluded to pickle because it is very easy to use with PYTHON, other available options for serialization exist too, including Google Protocol Buffers [15] which achieve very good compression and processing time, Apache AVRO [16] and Apache Thrift [17]. We highly suggest for a production environment and for real data the usage of one of the aforementioned libraries as they can decrease network traffic very much and increase computing performance. Finally, it is important to say that we didn’t make any benchmarking regarding these libraries as the choice of one of them it was out of the scope of this work.

9.1.1.5 Bokeh

There are plenty of visualization libraries in the python ecosystem, most known are matplotlib[18], plotly [19], seaborn [20] and bokeh [21]. We finally concluded to use bokeh as it has a lot of advantages in data visualization. Bokeh is an interactive visualization library that targets modern web browsers for presentation. Its goal is to provide elegant, concise construction of versatile graphics and to extend this capability with high performance interactivity over very large or streaming datasets. In the current work we use bokeh to generate html files for every epoch and evaluate the results of our algorithm. Bokeh as we already mentioned supports streaming data, an optimization of current implementation is to use this feature to have a real time evaluation of our algorithm.

9.2 Software implementation

To implement our approach, we follow object-oriented programming using PYTHON. The main classes we implement are:

<u>Producer</u>

Producer is a simple class which only responsibility is to send the results of clustering to appropriate sites. It has a *publish* method called after the algorithm is finished. It uses rabbit MQ library to send the results to message broker. If in the future, we decide to use another message broker we will have to replays just the public method of the producer in order to send to the new message broker. This way we have decoupled the functionality of producing messages with the rest functionalities.

<u>Consumer</u>

This class initializes the whole the process. Its responsibility is to make a connection with message broker and receives messages. It implements two methods *callback* and *callbackExemplars* which receives messages. The *Callback* method receives raw data and use *StreamAP* algorithm to extract clusters, after it publishes the results. The *callbackExemplars* method receives exemplars and in case we are a remote site we just update exemplars' weight and in case of central site (CS) we collect the results of all remote sites and we execute *StreamAP* algorithm and we send the results back to the remote sites.

<u>Item</u>

This class is responsible for holding information related to data items, including their coordinates x and y, that will eventually used for visualization, the weight of every item, the preference and the type which can be "item" or "exemplar". Moreover, it implements all the necessary methods to compute similarity and preference for every item.

<u>Exemplar</u>

Exemplar is a subclass of item which inherited all the fields and methods from *Item* and add some more fields, like timestamp that we need to compute the weight of each exemplar when the stream flows and *items* which is a list of items that choose this exemplar as their representative. It implements also methods to compute weighted dissimilarities needed from our algorithm.

<u>SimilarityMatrix</u>

This class is responsible to compute the similarity matrix that we will pass to the affinity propagation algorithm. It will compute appropriate the similarity of every combination between "point" and "exemplar". It also updates the preference for every item.

<u>StreamAP</u>

This class has all the implementation for the stream AP algorithm. It holds information regarding the decay factor, current timestamp, a list of exemplars, a list of items that we will use in clustering process and a list of legacy exemplars. It has two main methods which are used when we consume data from message broker. The *update* method which is used when we consume raw data and execute *StreamAP* algorithm and also increases timestamp field and the *updateFromExamplers* method which is responsible to update the weight for each local exemplar. In this class we have also implemented all the methods needed from *StreamAP* algorithm in order to implement the steps we described in previous section. It has also many other secondary methods needed from the algorithm.

<u>Visualization</u>

This class is very important for the whole process because it is responsible for visualizing the results of algorithm (exemplars, items) and let us evaluate the algorithm for every epoch. It saves the results to HTML files using *bokeh* [21] library.

10 Deployment and execution

All experiments and deployments were executed to a single computer, but the way the applications implemented provides the capability of deployment in a distributed environment.

To execute our algorithm, we need a message broker (RabbitMQ) and to execute our code with the appropriate command line parameters. To minimize the provisioning and the configuration we highly recommend the use of docker containers [22]. During this work we use a docker container for RabbitMQ and an optimization of our current work is to create docker images for *StreamAP* algorithm to can be executed in every environment that supports docker without any other dependencies. Now in order to deal with Python dependencies that our algorithm needs we create a Python virtual environment and install all he dependencies, for this reason before we execute our algorithm we have to load this environment.

When we execute our Python scripts we must pass two parameters, the first parameter is the name of the application and the name of the queue that the application will consume data and the second parameter is the name of the queue that the application will send the results of the clustering (exemplar). In case of a remote site the name of the queue that will send the results is: CS (Central Site) and CS will get as parameters all the name of the remotes sites to send back the results of the global clustering. This part of implementation needs improvement so as to eliminate the need of passing all the names of the remote sites which in a real environment this number can be very big. To deal with this issue what we need is just a better configuration in message broker. Most message broker provides the feature of *groupId* that means that consumers which belong to the same group they share the messages from queues but consumers which have different *groupId* the consume from the same queue without missing any messages because another consumer get these messages before them. In this the CS will send its results just to one queue and all the remote sites will consume from this queue. In the image below, we present our suggestion for a real system.

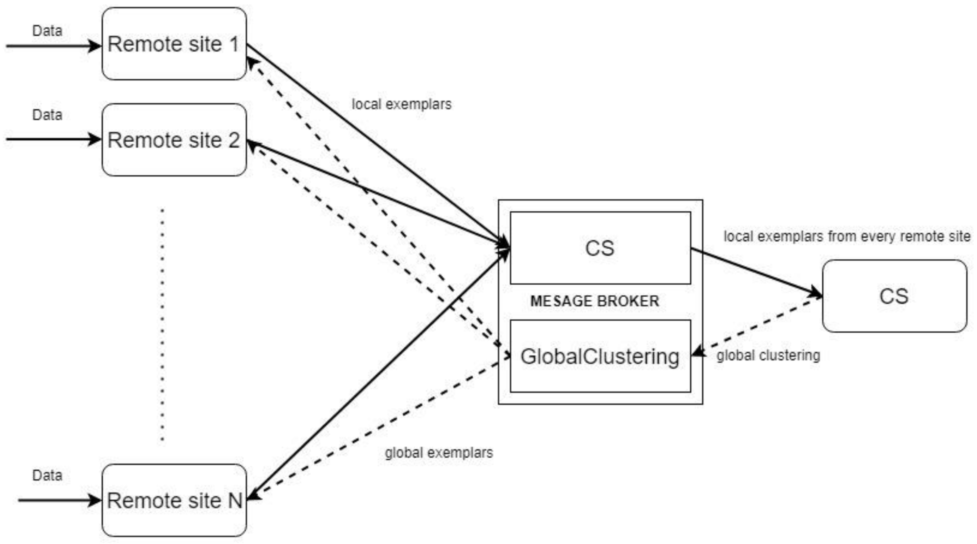

Figure 3: Proposed clustering approach for a real system

11 Result

We evaluate our algorithm under different decay factor parameters (λ = 0,01, λ =5) and we noticed that when we give lower emphasis in historical data, the algorithm produced more clusters in every epoch and when we give higher emphasis in historical data the data tends to choose previous clusters. We evaluate the algorithm in three different datasets. In the first dataset new clusters emerged after few epochs, in second dataset all clusters exist from the first epoch and in the last dataset one cluster exists only in the first epochs and after no items arrive relative to this cluster. Below we present the results of all datasets that we used to evaluate our algorithm, and how the algorithm created clusters in remote nodes and central site in some epochs.

11.1 Dataset 1

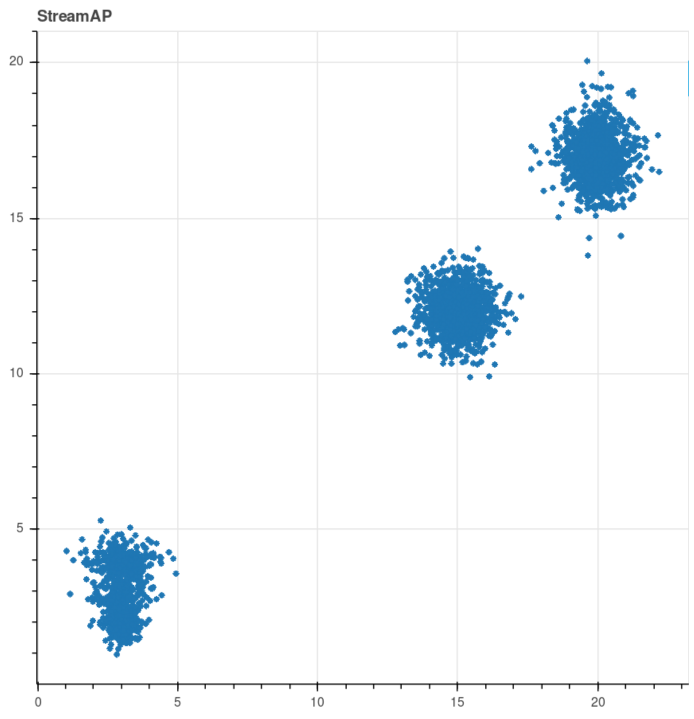

Figure 4: Initial dataset 1

We can see that the initial dataset has 3 clusters. Using this data, we create three different datasets from the points of initial dataset and use three different “sensors” to produce data and send to three different remote nodes to execute the proposed streaming process.

11.1.1 Epoch 0

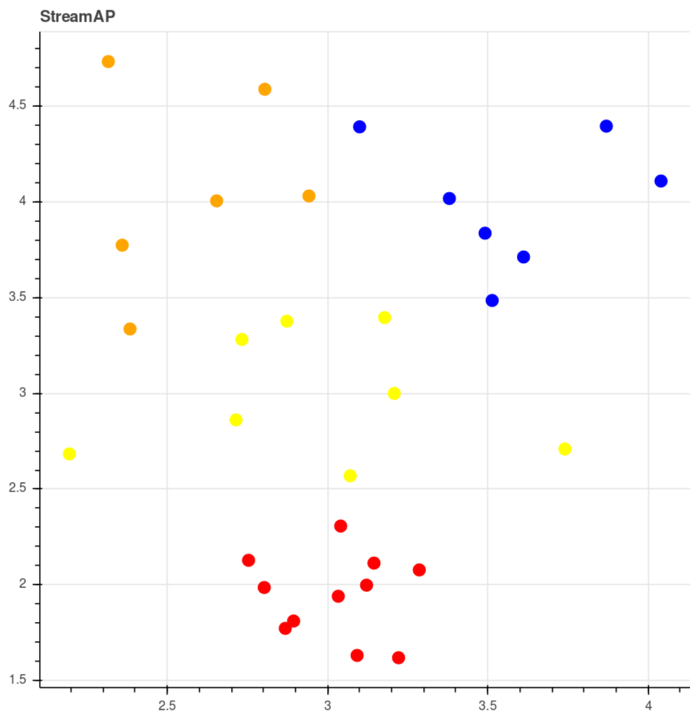

Figure 5: Global clustering in epoch 0 for dataset 1

All the points in the above picture are the center of clusters from three remote nodes (exemplars), as we can easily see in first epoch the three sensors send data only from one cluster of the initial datasets and every remote node create many clusters, then the CS make a second level clustering and create 4 clusters.

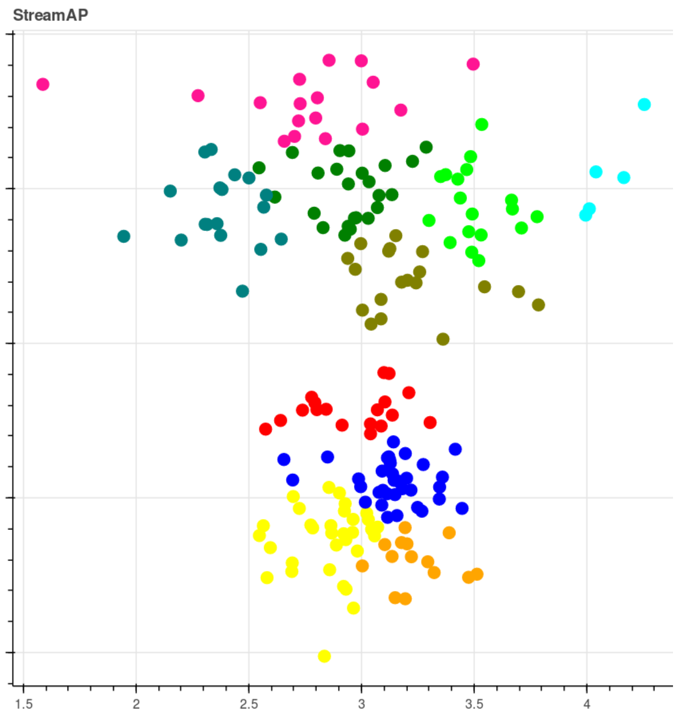

Figure 6: Local clustering in first remote node in epoch 0 for dataset 1

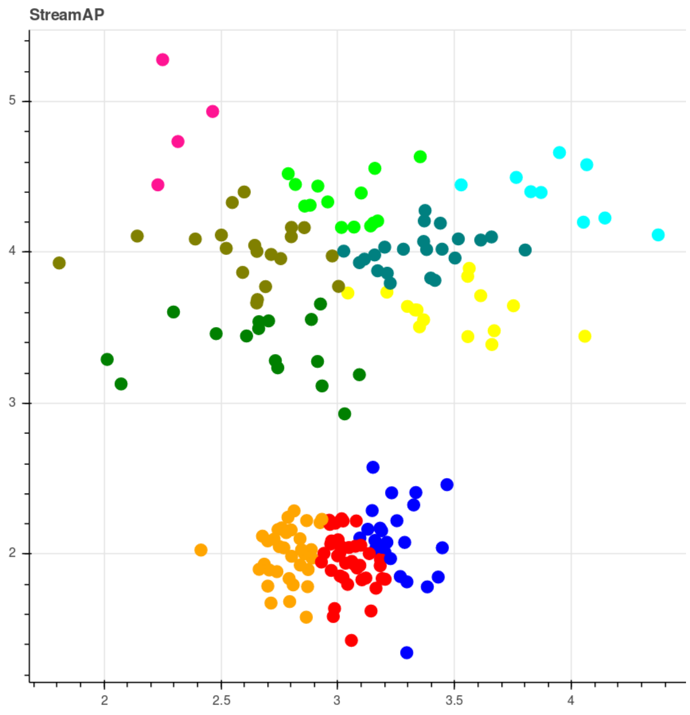

Figure 7: Local clustering in second remote node in epoch 0 for dataset 1

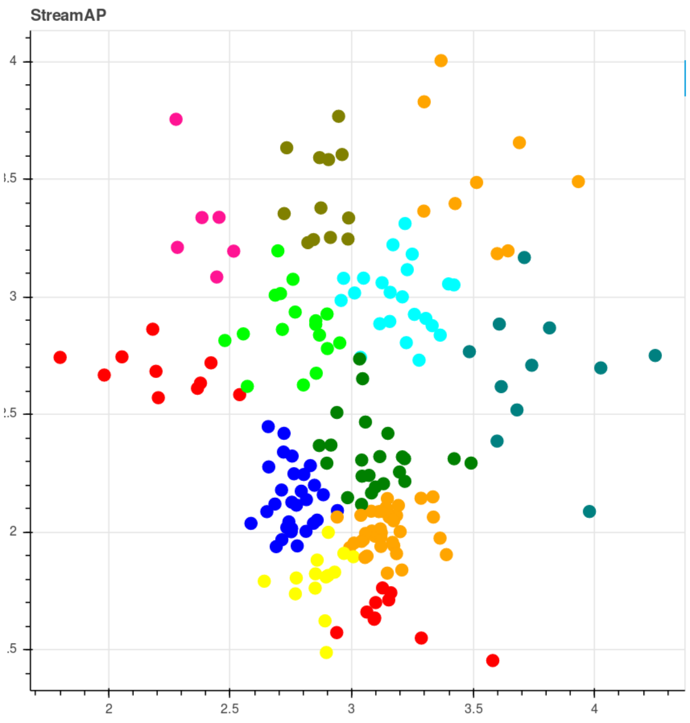

Figure 8: Local clustering in third remote node in epoch 0 for dataset 1

11.1.2 Epoch 20

We set the batch size at 200 points at every epoch. After many epoch sensors send data also from other clusters from initial dataset and as we present below it seems that the algorithm detect these new clusters and the number of exemplars that coming are lower.

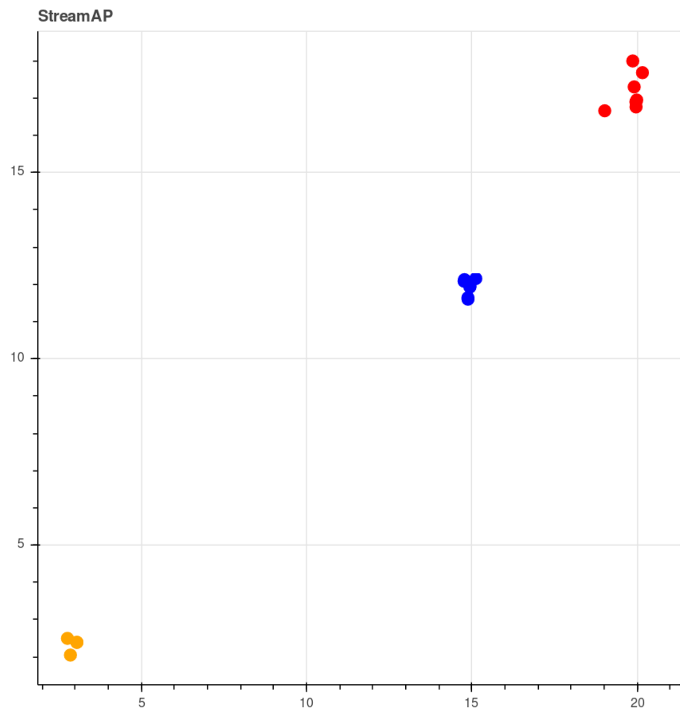

Figure 9: Global clustering in epoch 20 for dataset 1

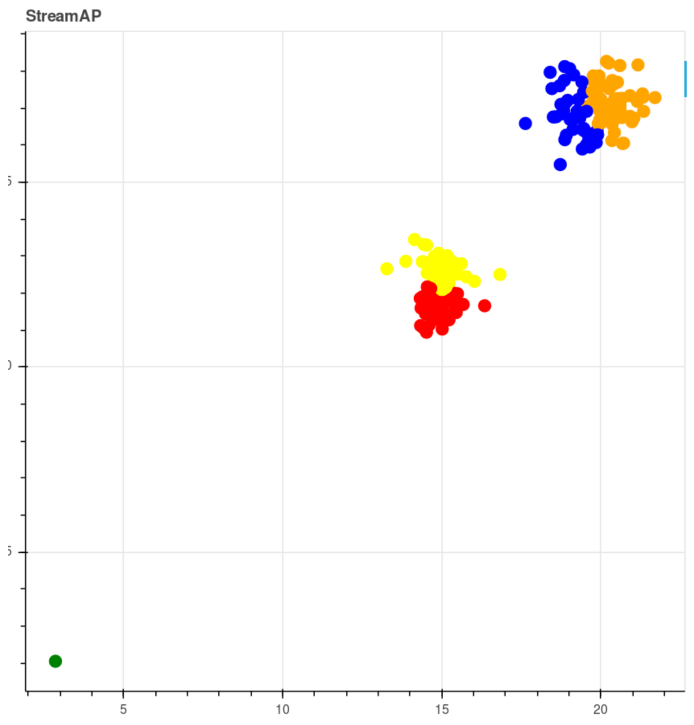

Figure 10: Local clustering in first remote node in epoch 20 for dataset 1

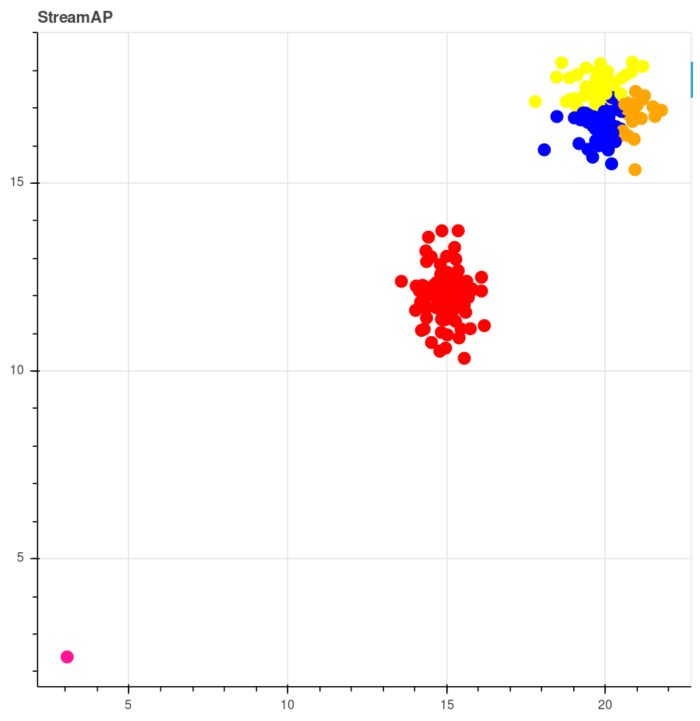

Figure 11: Local clustering in second remote node in epoch 20 for dataset 1

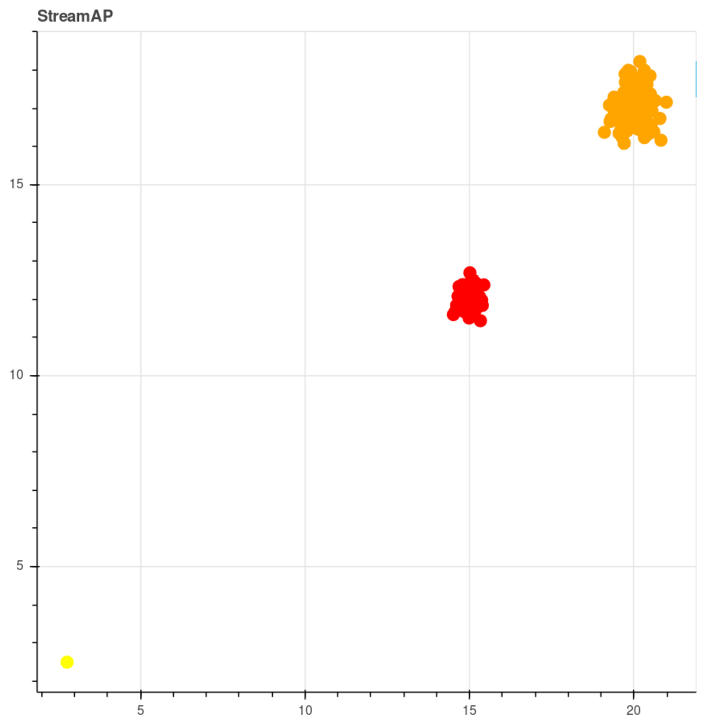

Figure 12: Local clustering in third remote node in epoch 20 for dataset 1

After many epochs the clusters seems to be constant in every remote site.

11.1.3 Final clustering, epoch 50

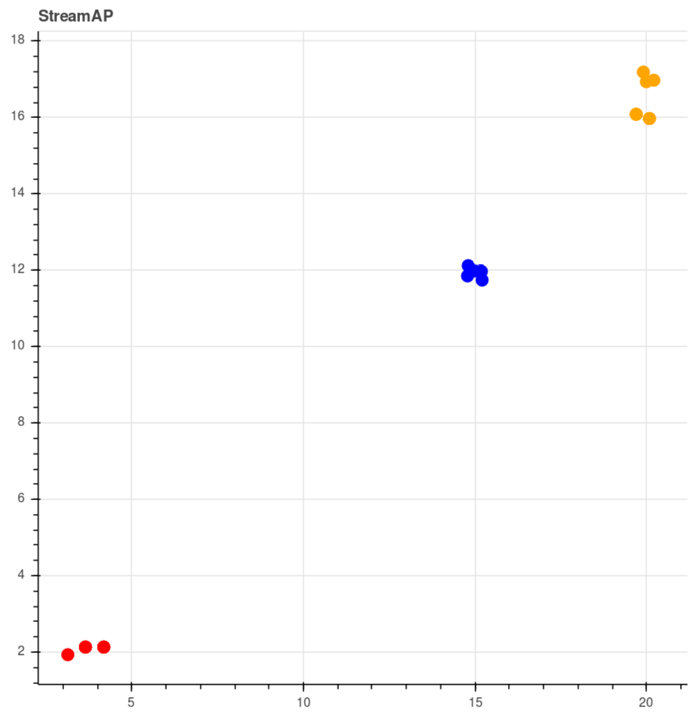

Figure 13: Final Global clustering in epoch 50 for dataset 1

Final global clustering seems to detect correctly the three clusters that has the initial dataset, moreover the remote sites also make correct clustering

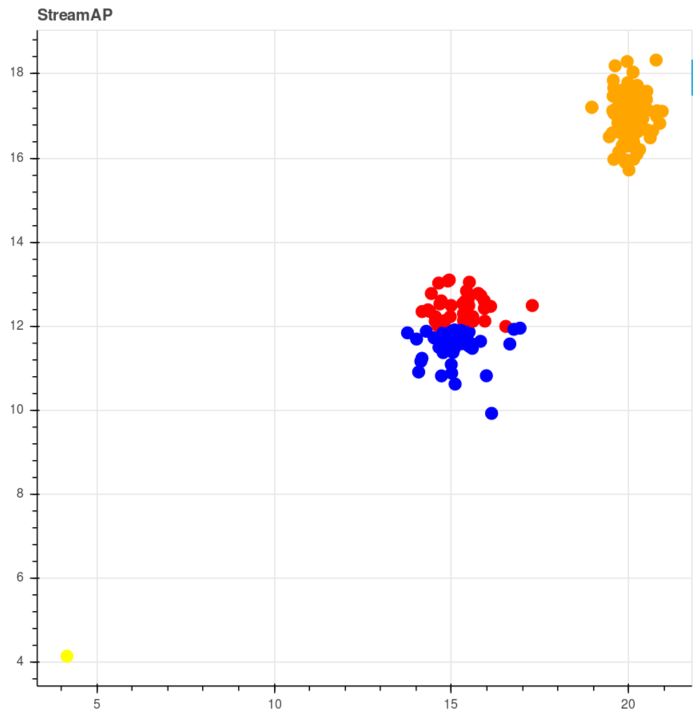

Figure 14: Local clustering in first remote node in epoch 50 for dataset 1

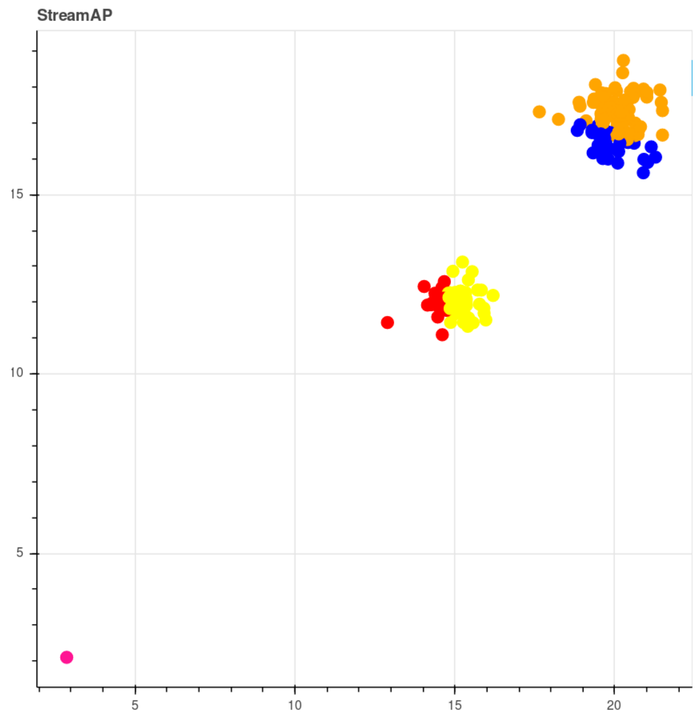

Figure 15: Local clustering in second remote node in epoch 50 for dataset 1

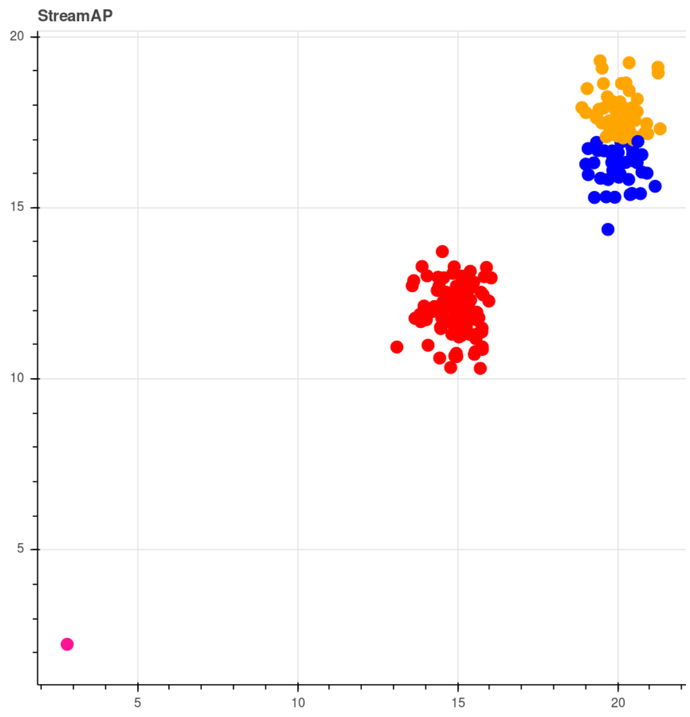

Figure 16: Local clustering in third remote node in epoch 50 for dataset 1

11.1.4 Compare with Central approach

In the following chart we will see the centers of the clusters in Central approach (Red) and in Distributed approach (blue). As we can see, both approaches detect correctly the three clusters but the center of the cluster in each case is different.

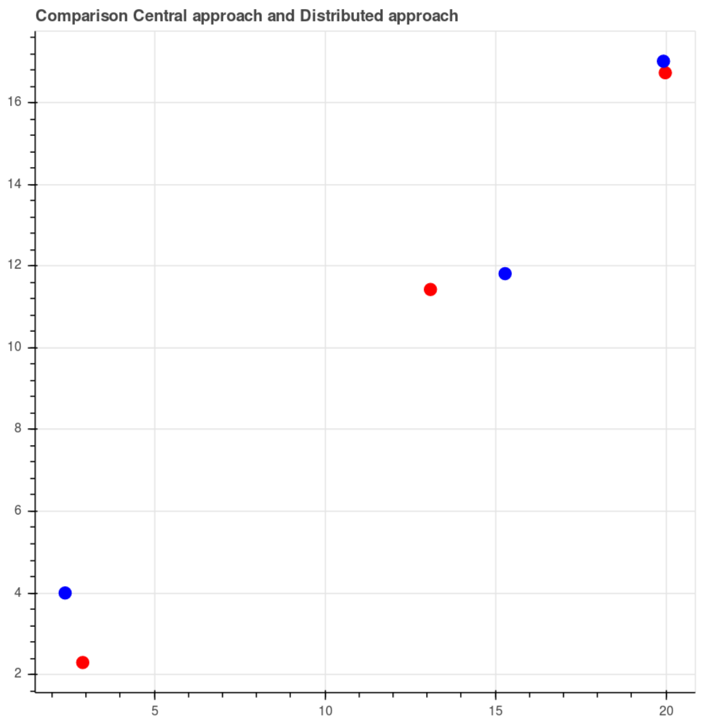

Figure 17: Compare Central approach and Distributed approach

	CENTRAL	DISTRIBUTED
CENTER 1	2.8976, 2.29373	2.3814, 3.9967
CENTER 2	13.0950, 11.4299	15.2839, 11.8179
CENTER 3	19.9823, 16.7374	19.9283, 17.0200

Table 1: Centers of clusters in Central and distributed approach

11.2 Dataset 2

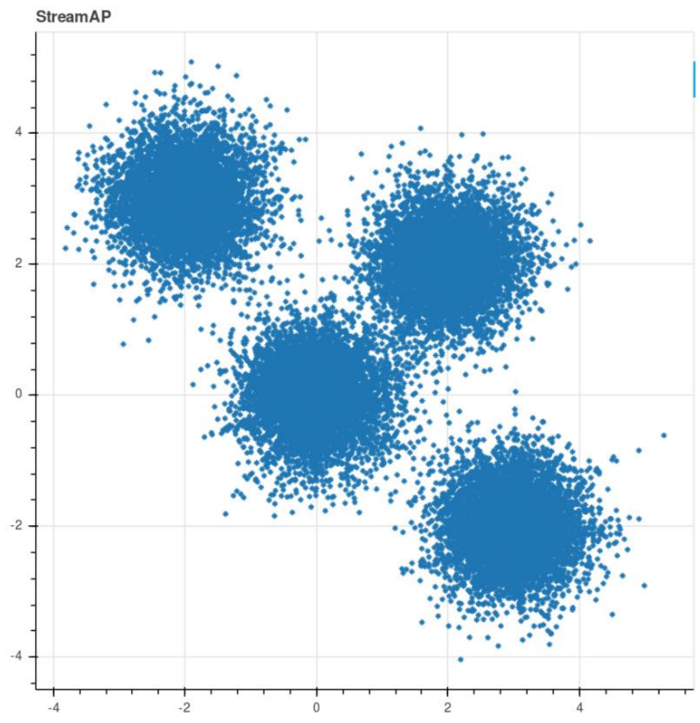

Figure 18: Initial dataset 2

In the above image we can see 4 clusters with center: [0,0,], [2,2], [-2,3], [3,-2]. We generated this dataset using the PYTHON [5] library scikit-learn [6]. In our experiment we use 3 sensors that send data to 3 remote sites and one CS. Data of all clusters arrives from the first epoch till last epoch. In this experiment we noticed that the algorithm finds, in CS node, 4 clusters in almost all epochs.

11.2.1 Epoch 0

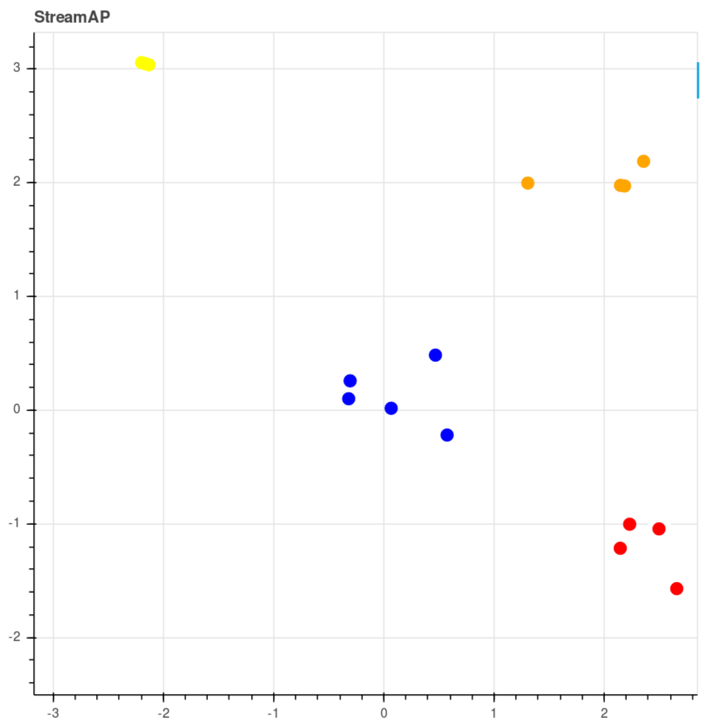

Figure 19: Global clustering in epoch 0 for dataset 2

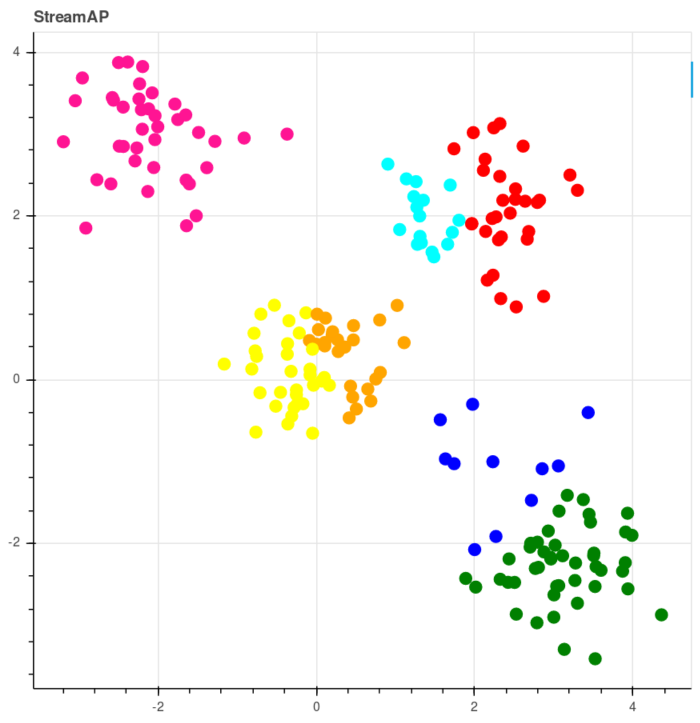

Figure 20: Local clustering in first remote node in epoch 0 for dataset 2

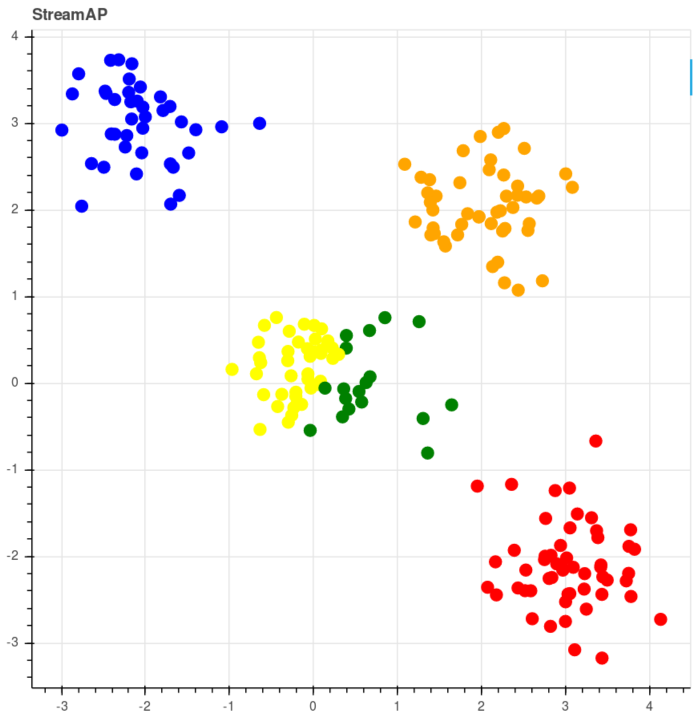

Figure 21: Local clustering in second remote node in epoch 0 for dataset 2

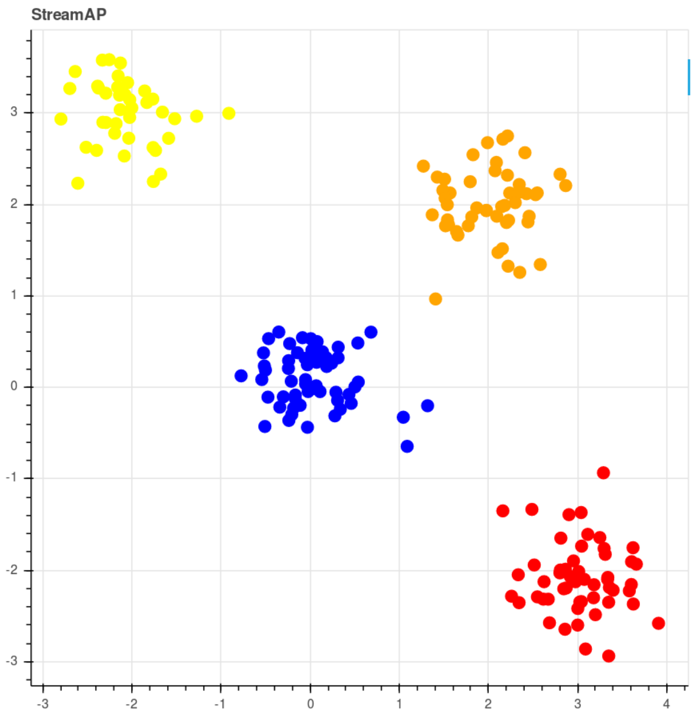

Figure 22: Local clustering in third remote node in epoch 0 for dataset 2

11.2.2 Epoch 20

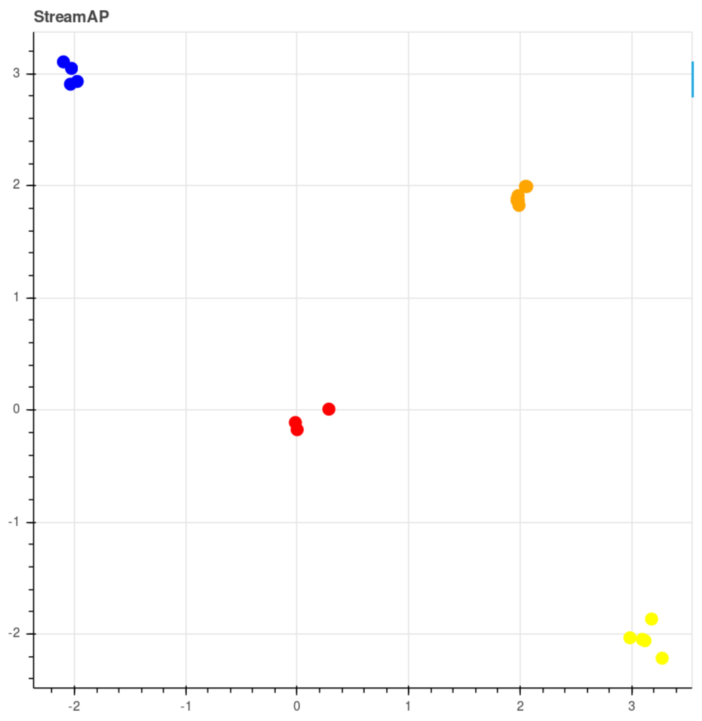

Figure 23: Global clustering in epoch 20 for dataset 2

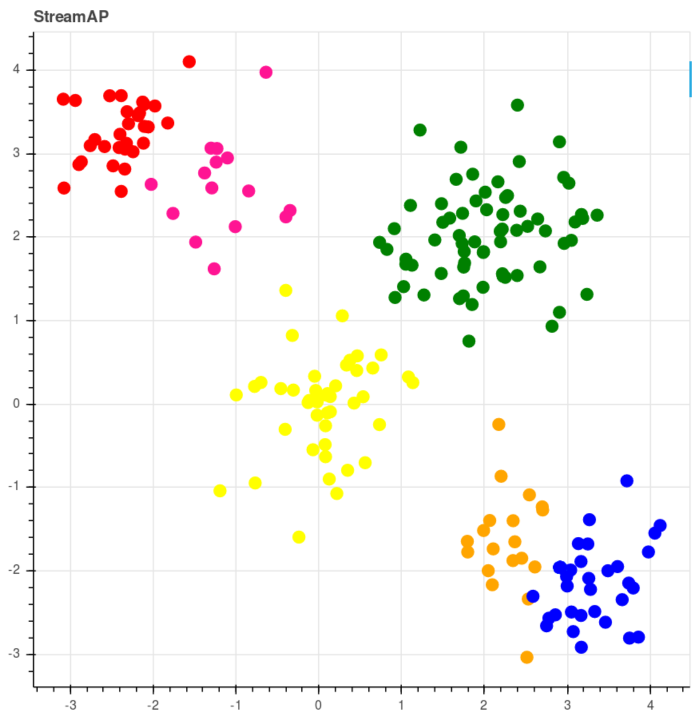

Figure 24: Local clustering in first remote node in epoch 20 for dataset 2

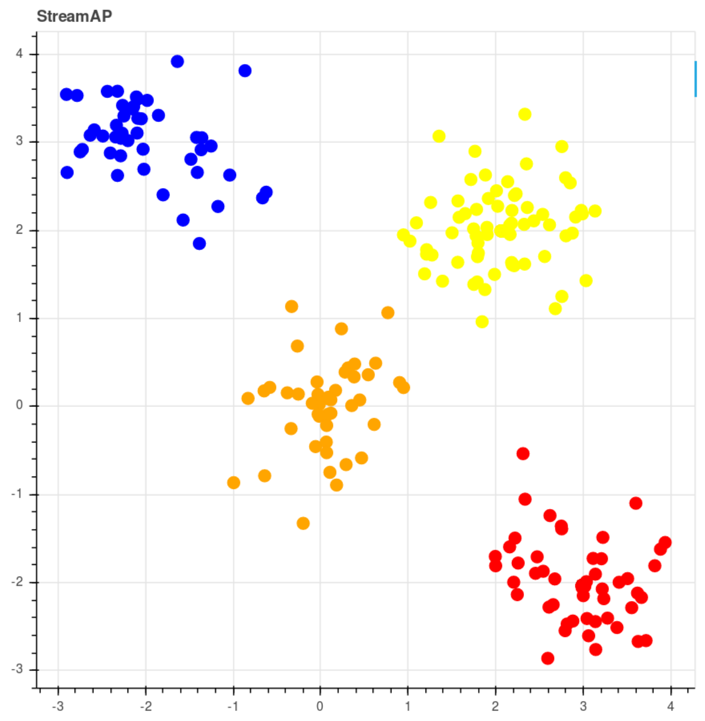

Figure 25: Local clustering in second remote node in epoch 20 for dataset 2

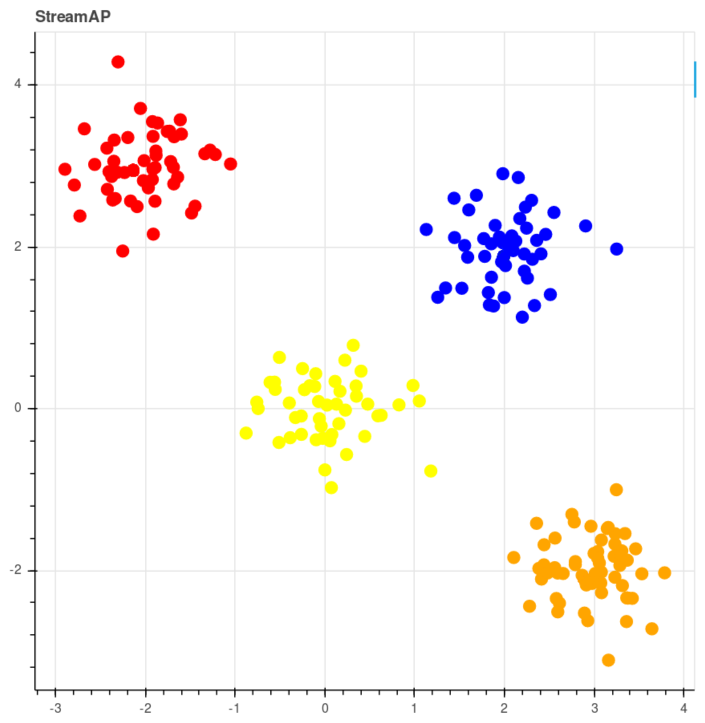

Figure 26: Local clustering in third remote node in epoch 20 for dataset 2

11.2.3 Final clustering, epoch 50

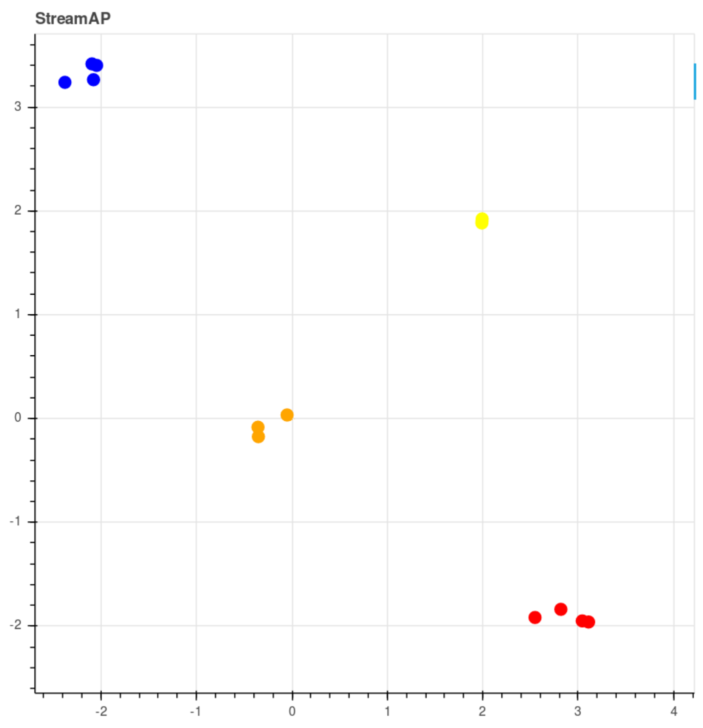

Figure 27: Final Global clustering in epoch 50 for dataset 2

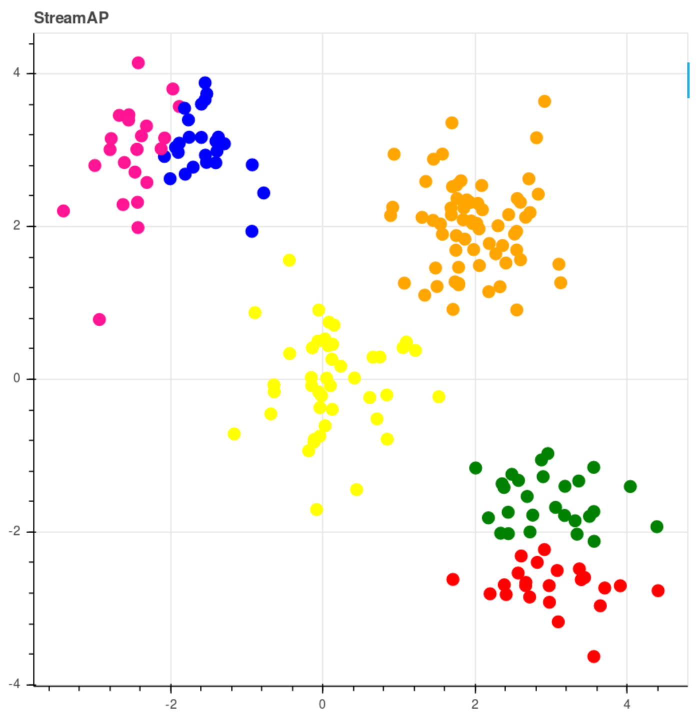

Figure 28: Local clustering in first remote node in epoch 50 for dataset 2

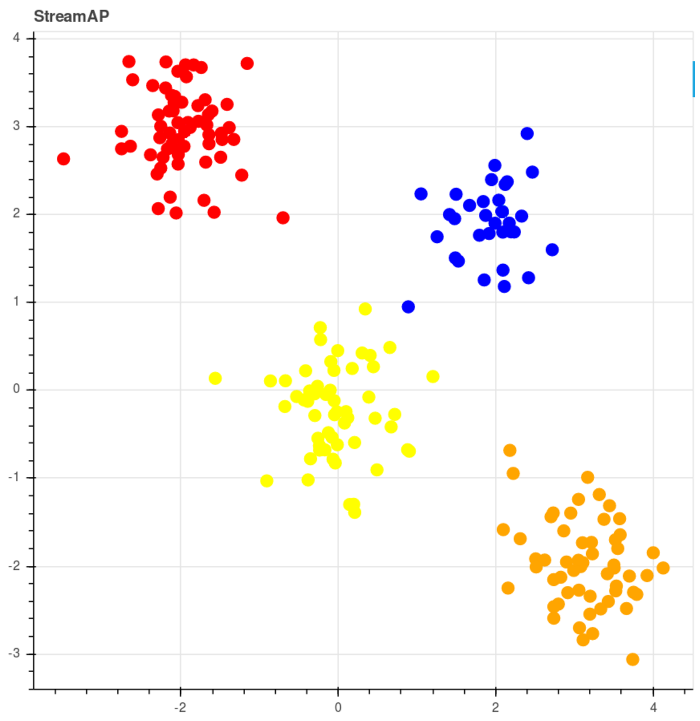

Figure 29: Local clustering in second remote node in epoch 50 for dataset 2

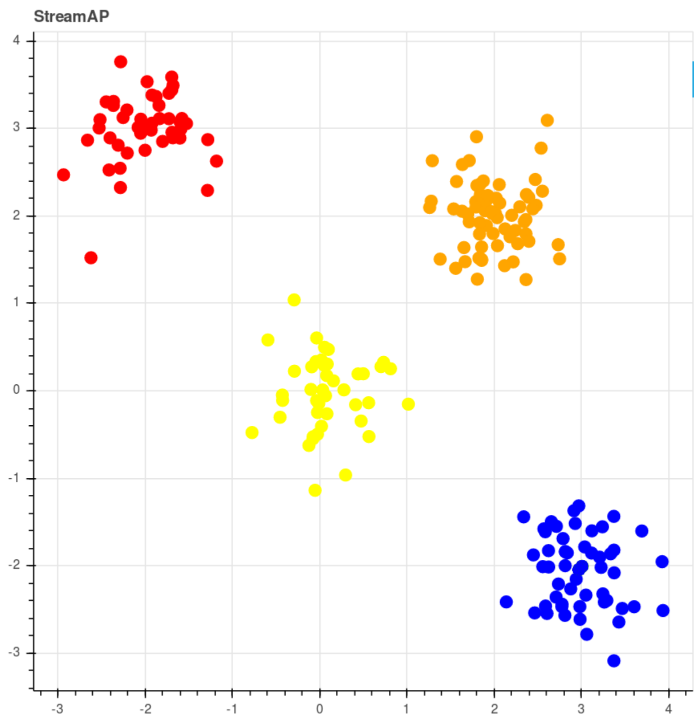

Figure 30: Local clustering in third remote node in epoch 50 for dataset 2

11.2.4 Compare with Central approach

In this dataset the center in both approaches are very close and moreover one cluster is same.

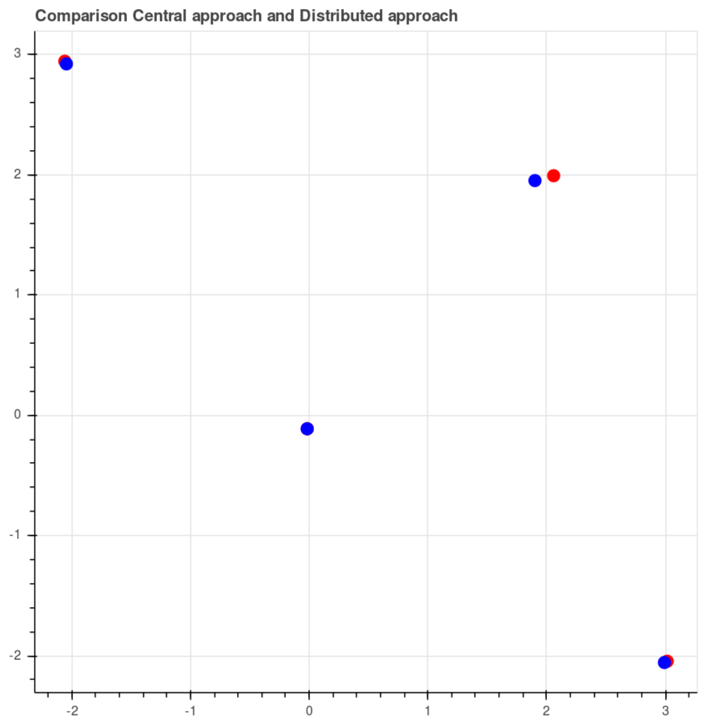

Figure 31: Compare Central approach and Distributed approach

	CENTRAL	DISTRIBUTED
CENTER 1	-2.0390, 2.9458	-2.0453, 2.9239
CENTER 2	2.0589, 1.9933	1.9015, 1.9528
CENTER 3	-0.0169, -0.1120	-0.0169, -0.1120
CENTER 4	2.9987, -2.0867	2.9924, -2.0567

Table 2: Centers of clusters in Central and distributed approach

11.3 Dataset 3

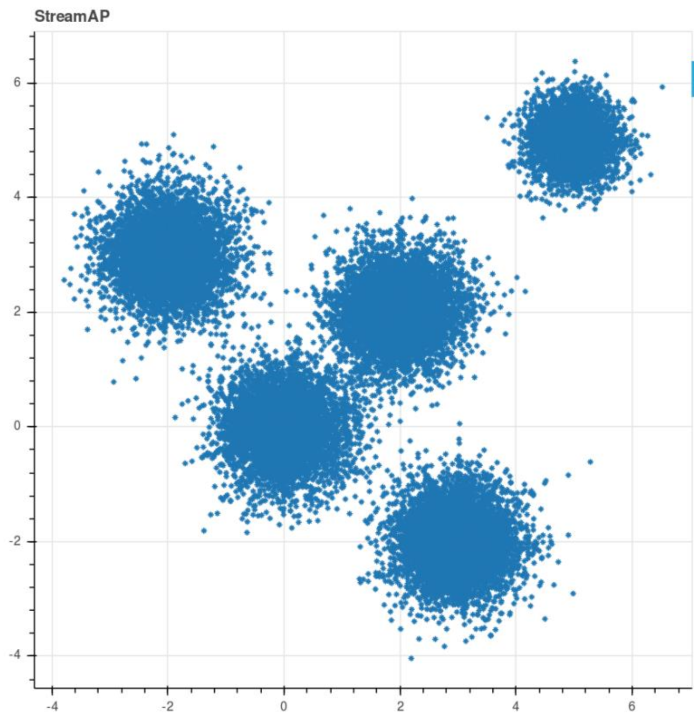

Figure 32: Initial dataset 3

In this dataset we have 5 clusters with center: [0,0,], [2,2], [-2,3], [3,-2], [5,5]. The whole datasets have 30000 points. The cluster with center [5,5] has only 3000 points and gets items only in the first epochs. The batch size is 200 items and that means that the cluster with center [5,5] will get data only in the first 15 epochs. One sensor will send data only from this clusters in the first 15 epochs and two other sensors will send data from the other four clusters, that means that the weight of the cluster with center [5,5] will have two times more items from other four clusters in the first 15 epochs and that means that also its weight will be almost 2 times higher. In this experiment we evaluate what will happen to cluster with center [5,5] after epoch 15.

11.3.1 Epoch 0

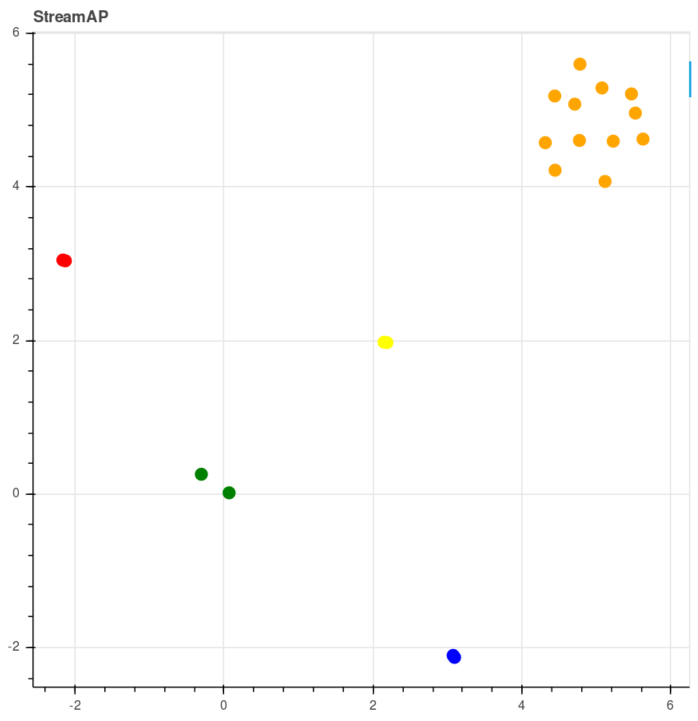

Figure 33: Global clustering in epoch 0 for dataset 3

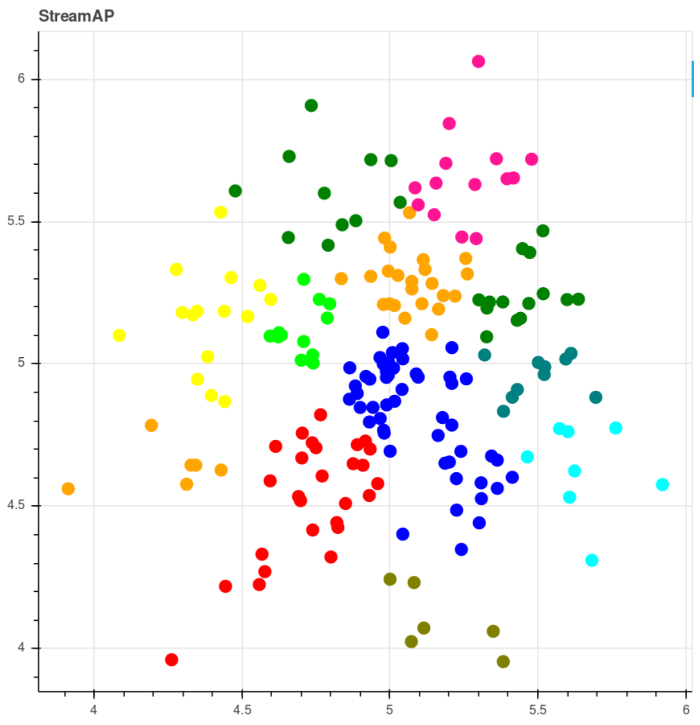

Figure 34: Local clustering in first remote node in epoch 0 for dataset 3

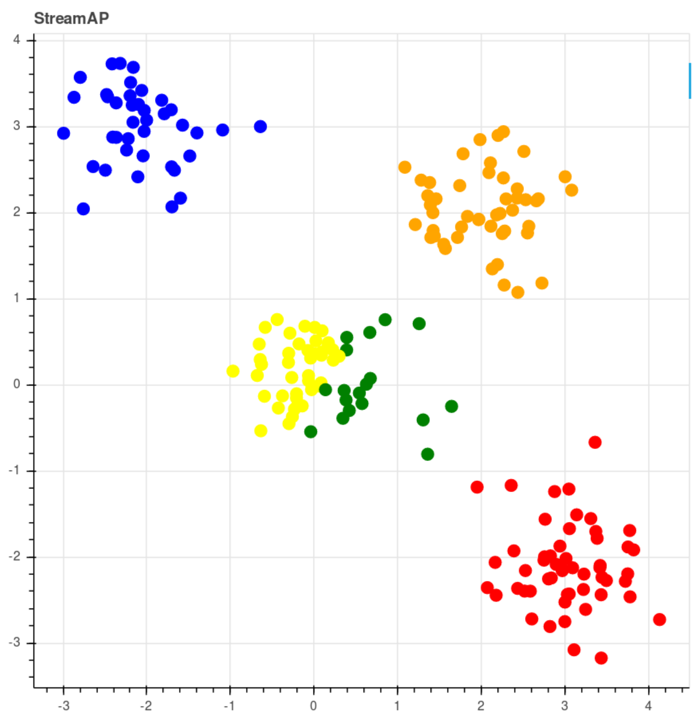

Figure 35 : Local clustering in second remote node in epoch 0 for dataset 3

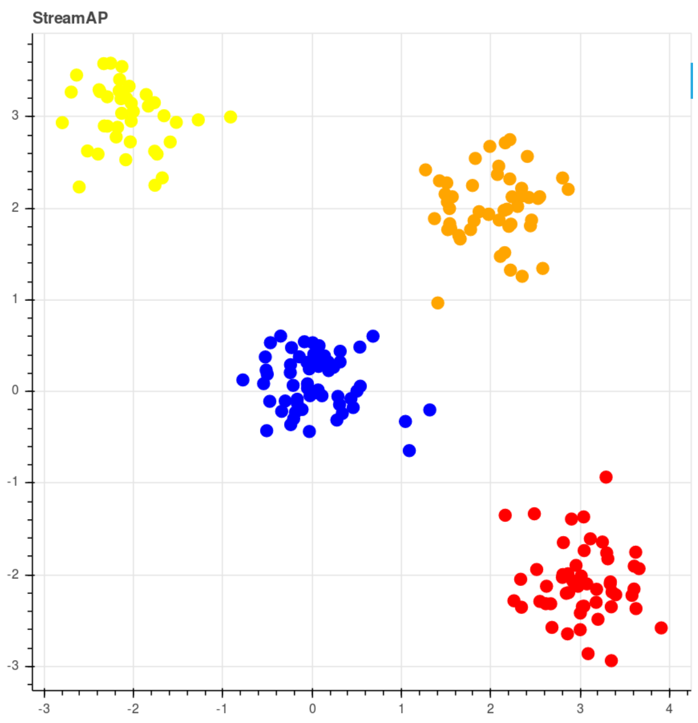

Figure 36: Local clustering in third remote node in epoch 0 for dataset 3

11.3.2 Epoch 20

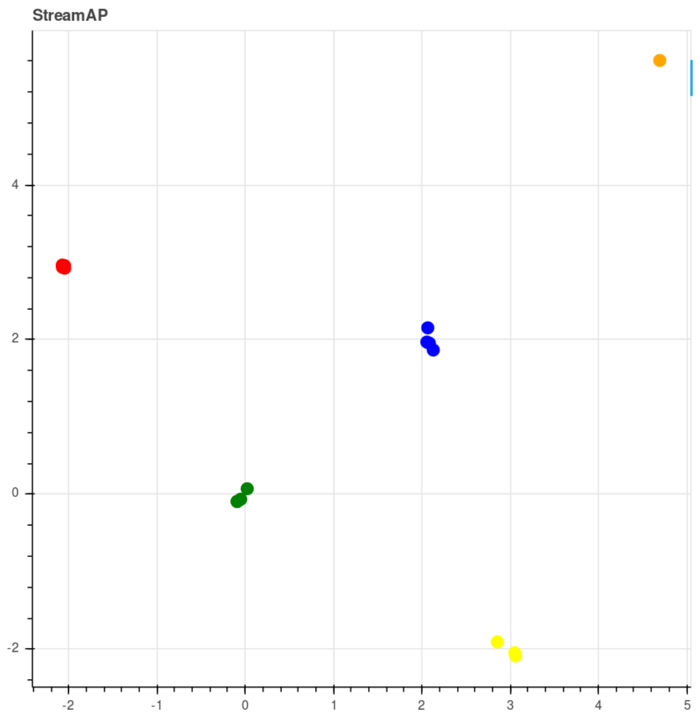

Figure 37 : Global clustering in epoch 20 for dataset 3

At epoch 20 cluster with center [5,5] doesn't receive any items but it still exists in global clustering.

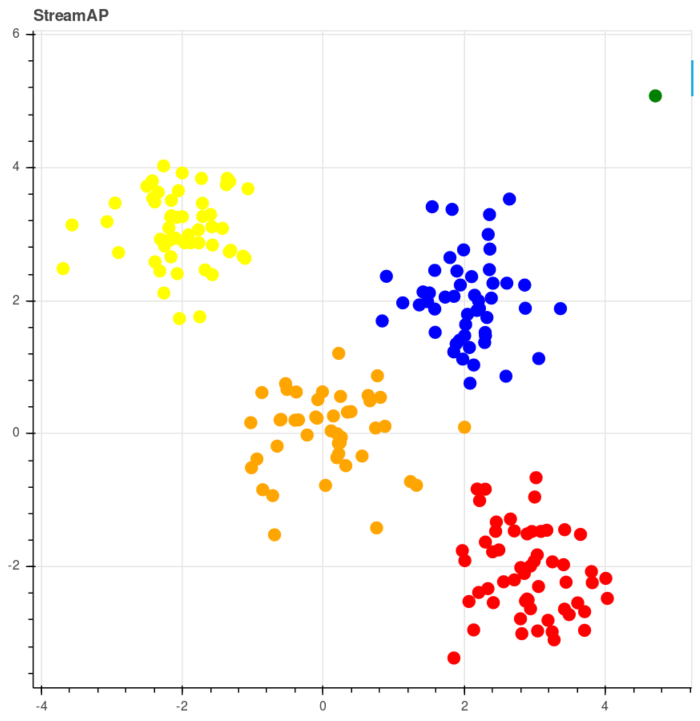

Figure 38: Local clustering in first remote node in epoch 20 for dataset 3

As we can see from the image above, cluster with center [5,5] does not receive any items near [5,5] but only in other four clusters.

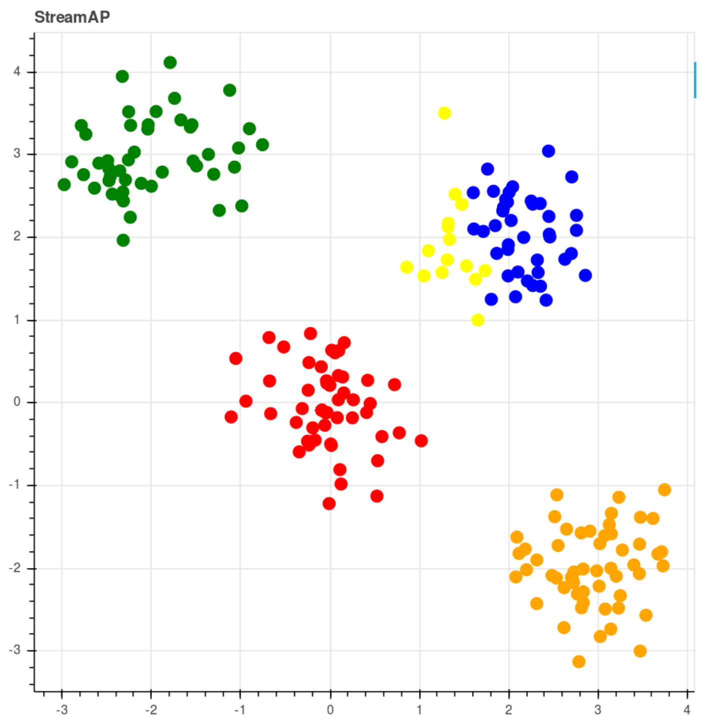

Figure 39: Local clustering in second remote node in epoch 20 for dataset 3

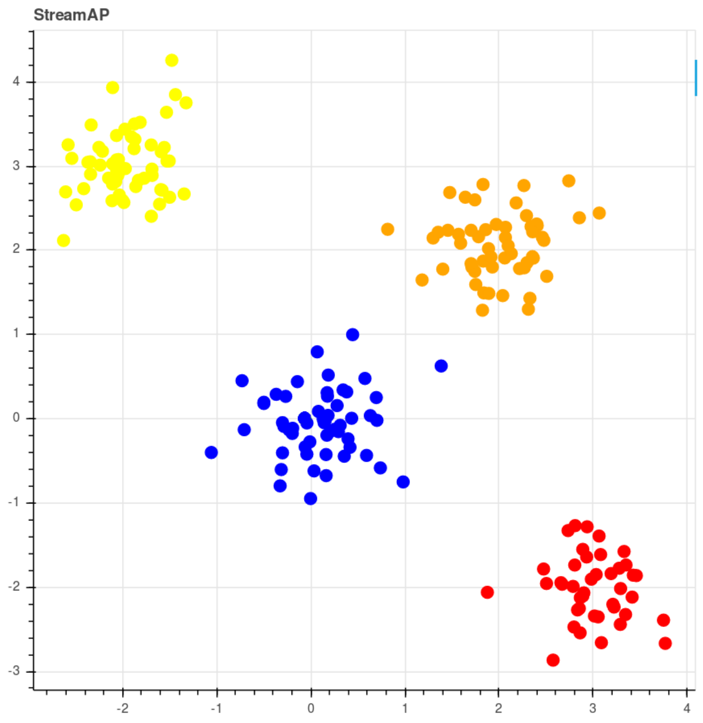

Figure 40 : Local clustering in third remote node in epoch 20 for dataset 3

11.3.3 Final clustering, epoch 50

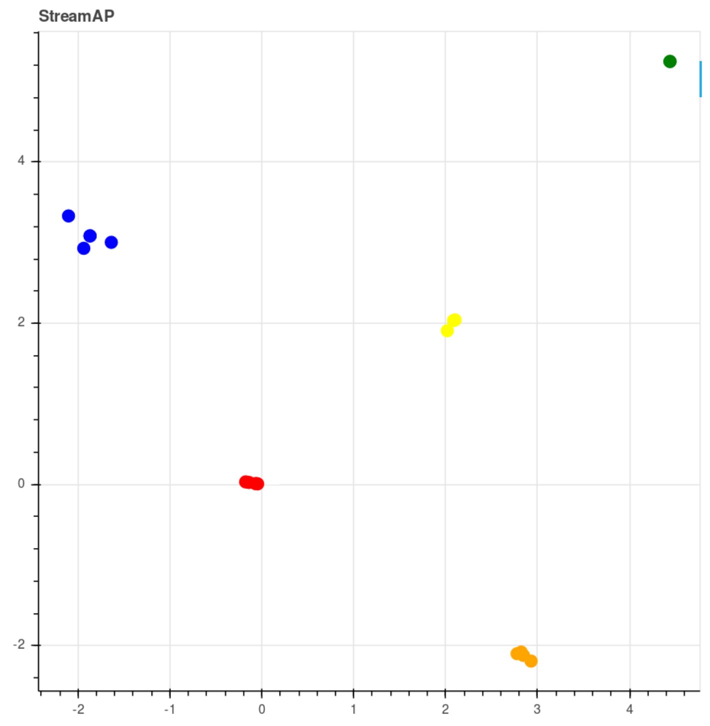

Figure 41: Final Global clustering in epoch 50 for dataset 3

As we can see from the image above the cluster with center [5,5] it still exists in the global clustering.

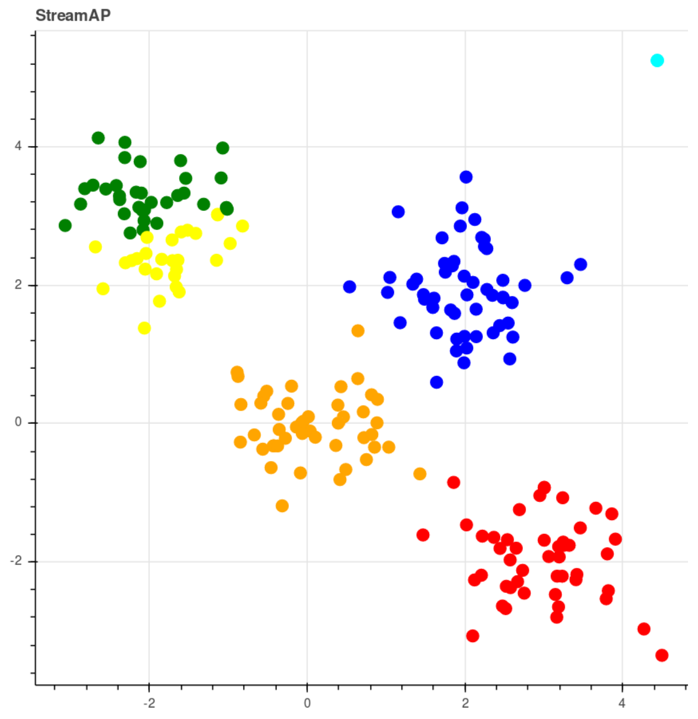

Figure 42: Local clustering in first remote node in epoch 50 for dataset 3

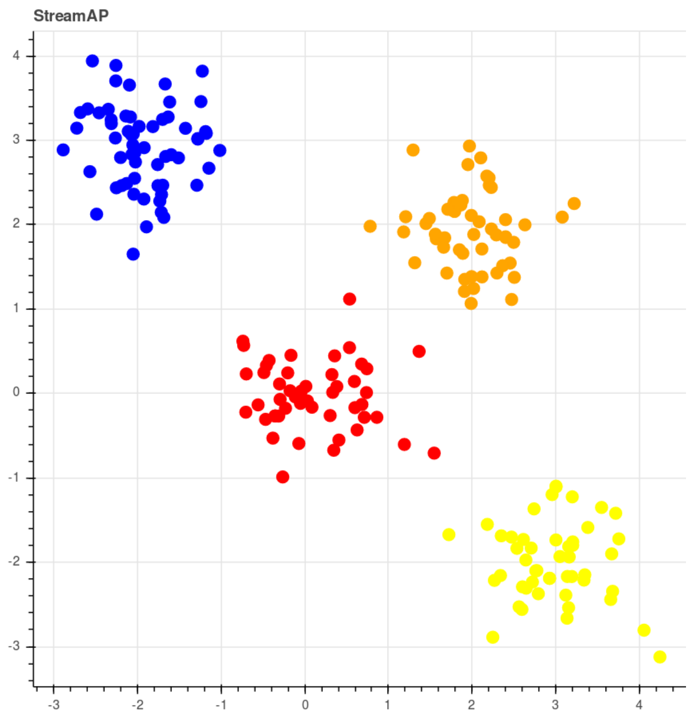

Figure 43: Local clustering in second remote node in epoch 50 for dataset 3

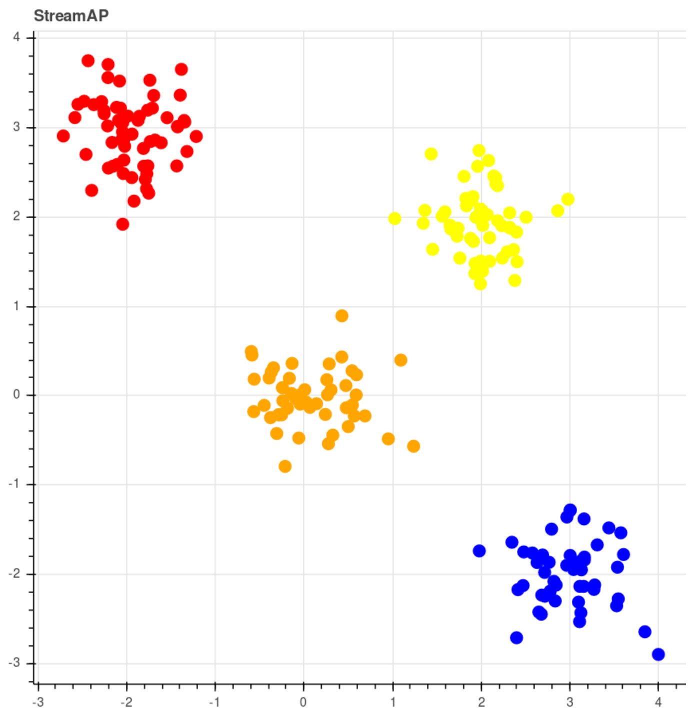

Figure 44: Local clustering in third remote node in epoch 50 for dataset 3

In this experiment we noticed that the cluster with center [5,5] remains till the end, we must stress that the decay factor (λ) was too small λ = 0,01 and when we evaluate the algorithm with this dataset but with higher λ = 10 we noticed that the cluster with center [5,5] disappeared after many epochs.

11.3.4 Compare with Central approach

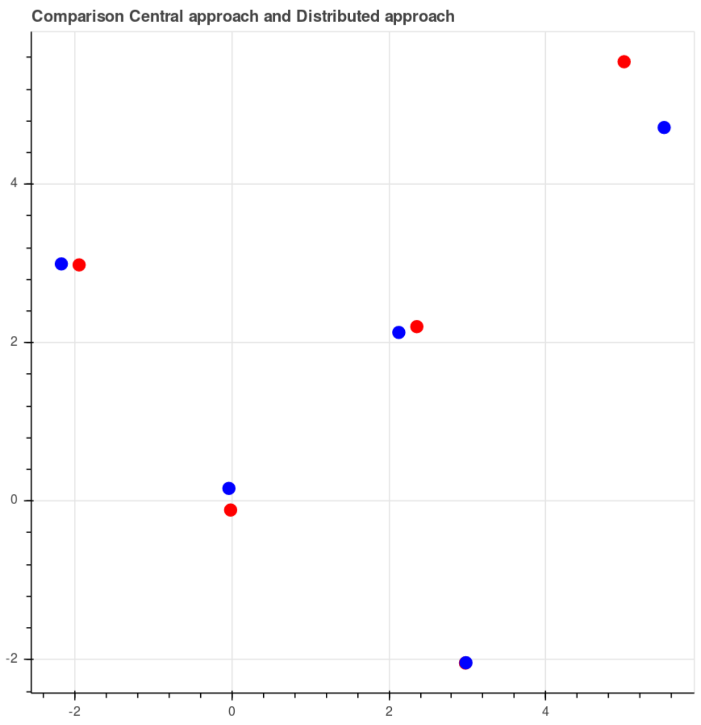

Figure 45: Compare Central approach and Distributed approach

	CENTRAL	DISTRIBUTED
CENTER 1	4.9976, 5.5464	5.5079, 4.7160
CENTER 2	-1.9469, 2.9821	-2.1735, 2.9946
CENTER 3	2.3562, 2.2028	2.1264, 2.1294
CENTER 4	-0.0169, -0.1120	-0.0387, 0.1612
CENTER 5	2.9753, -2.0429	2.9819, -2.0388

Table 3: Centers of clusters in Central and distributed approach

12 Future Work

During this work we came against some problems that we must deal including the way the downtime of a node is handled, how to achieve synchronization of the nodes, how to add new nodes in the whole process and how to improve the accuracy of algorithm. Below we present our suggestions and some issues that we should take into consideration in a future work:

- We can save in every epoch the state of the algorithm (list of exemplars, weight of exemplars, timestamp, etc., …). In this way if a node is down when it will be live again it will continue to consume data from message broker and work on its previous state. In our current implementation this can be implemented easily with the use of *pickle* library.
- As our main goal in this work is to decrease the network traffic, we definitely need to use a serialization library, there are very new libraries that can reduce the size of transferred objects very much and also the processing time, if we think that the remote sites may be poor in hardware resources, this will be very beneficial not only for the network but also for the computing time needed for serialization and deserialization.
- A major problem we observed during our work is how to synchronize all these nodes. For our work we just give the start all the nodes the same time but in a real system this is may not possible for many reasons. For example, the data that consumes a node it can be many times higher than in another node, that means that *StreamAP* algorithm will be finish slower in the node with higher traffic and sends its exemplars later to CS. In our current implementation we just wait in CS to consume data from every remote node but in a real system this is possible to jeopardize the whole process as if one node is much slower than the other ones, the whole process will be delayed and if we not wait for this node and we just consume all the data that are in the message broker, we will ignore a node with many data and this will mispresent the global clustering.
- In our two-level clustering approach, we try start from a distributed environment and finally we assemble all the data in CS, we consider that the data aren't too many as they are only the exemplars, but what will happen if the remote sites are too many and send all their exemplars? In this case we can try a three-level clustering, that means to have two or more CS and a CS higher in hierarchy of them to make a global clustering in their exemplars, or just to use the traditional big data way and make the *StreamAP* to run distributed. In the first approach we will probably have a loss in the accuracy of the system and in second approach we will have to deal with the complexity of making this algorithm in a distributed fashion.
- More experiments in different use cases. For example, what happens when new nodes are inserted to the system or leave the system, how the system can handle the different response time of every node. All of these use cases must be tested in a variety of datasets and number of remote sites.
- Better use of new tools and technologies. There plenty of new tools that can be used to optimize our system. Tools for better deployment that can reduce the complexity and the deployment time that can give us a boost in experimentation of algorithm in

different uses cases, better message broker configuration that can help us to synchronize better the remote sites and many more that can be used in a real system.

- Algorithm optimization. In our current implementation when we get the results of global clustering we just updates the local exemplars weight from the exemplars we get from CS, in this way if the batch of the data that arrives in the next batch may belong to another exemplar from global clustering we just ignore them and assign them to an exemplar of the local clustering, our suggestion is to keep all global clustering exemplars in remote sites and not only the intersection with the local clustering, in this way we believe that we reduce the loss of the accuracy during the clustering process.

13 Conclusion

In this work we make an introduction to big data phenomenon and in special conditions that we have to deal with when we try to apply data mining in streaming data and how a distributed approach can help. We implemented a distributed version of the *StreamAP* algorithm and examine the impact of this approach in the accuracy of algorithm in a variety of use cases and how possible is this approach to be applied in a real case scenario. We evaluated how the algorithm behaves if new clusters are created when the stream flows and when we give different emphasis in historical data. In conclusion we can say that this distributed fashion of *StreamAP* algorithm doesn't have a big impact in the accuracy of the algorithm but on the other hand we have many different issues needed to be solved if we want to apply this algorithm in a real case scenario. The main problem is the synchronization of the remote sites, when they send their results to CS there are many reasons that can lead to different response times of this nodes, such as the number of data this nodes handle, the distance of these nodes from CS, the efficiency of each node and network issues that we must take into consideration. In our approach we suggest the use of a message broker that provides as better error handling and easiness in configuration and implementation and can also help in remote nodes synchronization. For example, when remote nodes send their exemplars in the message broker, the CS can consume the local clustering of remote sites in a constant time interval, in this way we can have all the results of the remote sites when the CS will execute *StreamAP* to make a global clustering, but this also means that the global clustering is delayed.

14 References

[1] Maria Halkidi, Iordanis Koutsopoulos, “Online Clustering of Distributed Data using Belief Propagation Techniques”.

[2] https://www.datamation.com/big-data/big-data-use-cases.html

[3] Peter Mell, timothy Grance, “The NIST Definition of Cloud Computing”.

[4] https://aws.amazon.com/streaming-data/

[5] Jure Leskovec, Anand Rajaraman, Jeffrey David Ullman, “Mining of massive datasets”.

[6] B. J. Frey and D. Dueck, “Clustering by passing messages between data points”, Science, vol. 315,2007.

[7] https://www.python.org/downloads/release/python-350/

[8] http://scikit-learn.org/stable/

[9] http://cython.org/

[10] https://www.rabbitmq.com/getstarted.html

[11] https://aws.amazon.com/message-queue/benefits/

[12] https://kafka.apache.org/

[13] http://zeromq.org/

[14] https://docs.python.org/3/library/pickle.html

[15] https://developers.google.com/protocol-buffers/

[16] https://avro.apache.org/

[17] https://thrift.apache.org/

[18] https://matplotlib.org/

[19] https://plot.ly/

[20] https://seaborn.pydata.org/

[21] https://bokeh.pydata.org/en/latest/

[22] https://www.docker.com/