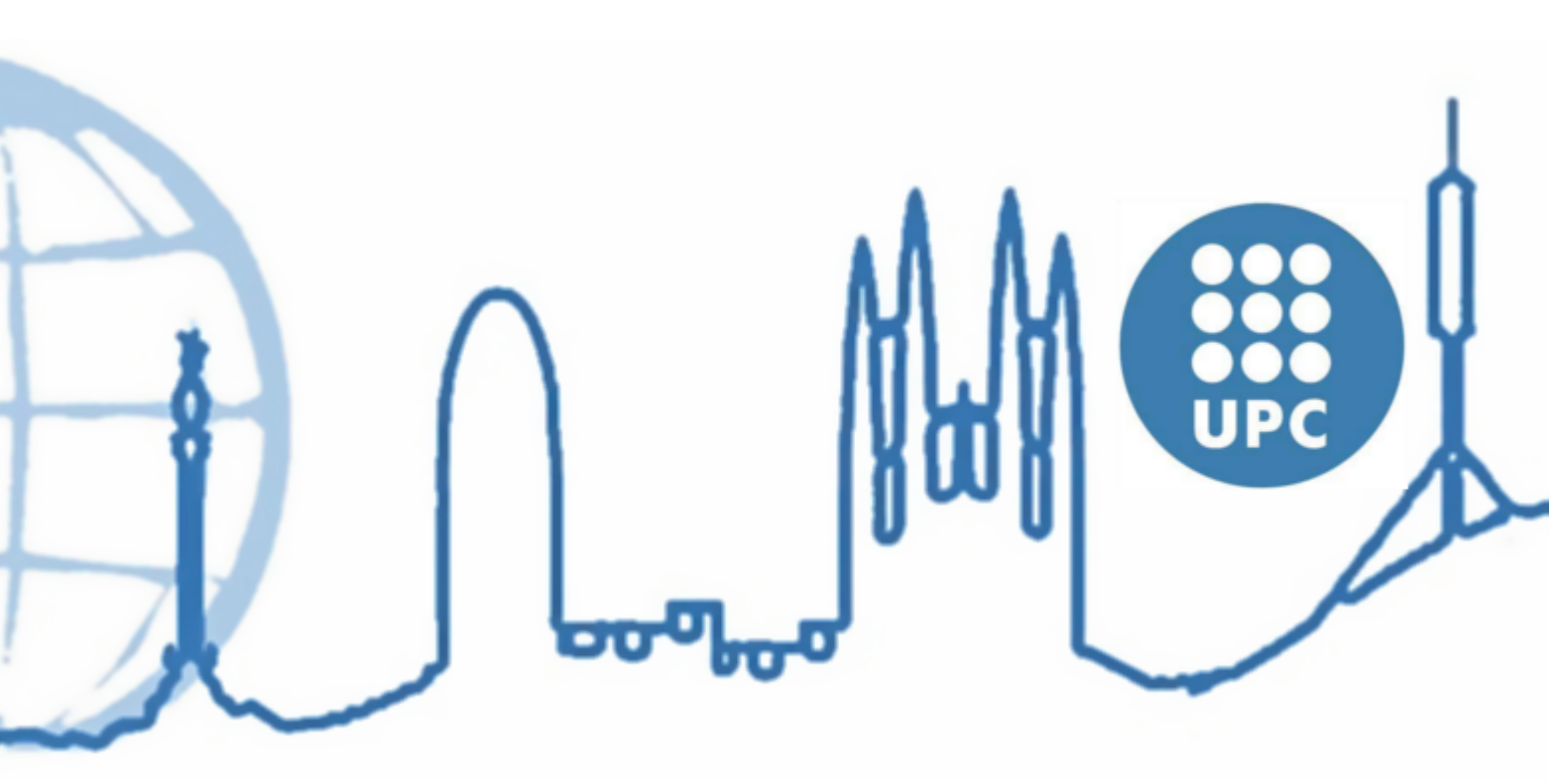

Scalable Distributed Real-Time Clustering for Big Data Streams

Antonio Loureiro Severien

Thesis for Masters Degree in the
European Masters in Distributed Computing

Jury

Supervisors:	Albert Bifet Gianmarco De Francisci Morales Marta Arias
Examiners:	Leandro Navarro Johan Montelius David Carrera

June 25, 2013

Acknowledgements

All my thanks goes to all people involved in this process of growth as a being that thinks. I would like to thank my family, specially my mom Paula Loureiro and aunt Claudia Loureiro, for their support and unstoppable pressure all the way. A great part of this project would not exist without the help of my friend and partner Arinto Murdopo. Thanks to all EMDC students which made the studies as fun as playing in the park. A great thanks to the Yahoo! Research lab in Barcelona who is always ahead in technology and opening such opportunities for students. Thanks Albert Bifet, Gianmarco Morales and Nikolas Kourtellis for speeding up my human learning on clustering algorithms. Great thanks to Matthieu Morel from the Apache S4 team for his patience and help on dissecting the S4 internals. Thanks to Marta Arias for accepting the challenge. Last but not least thanks to all the teachers for the patience and encouragement to pass their knowledge world of computers, even when discussions did not involve computers.

Barcelona, June 25, 2013
Antonio Loureiro Severien

“In all chaos there is a cosmos, in all disorder a secret order.”
– Carl Jung

“The universe is an intelligence test.”
– Timothy Leary

“The creation continues incessantly through the media of man. ”
– Antonio Gaudi

To Marilu, Gabi, Duda and Maroca
May they pave the way for a better world

Abstract

Ever growing volume of data production is the reality we are living in. Social networks, smart cities, telephone networks, the internet are some of the main data producers in the modern world and much of this information is discarded due to the high storage space it would require. Relevant data can be extracted from this massive amount of information and be used to build better cities, offers better services, make predictive analysis, group similar information and many other applications. All of this is possible due to machine learning and data mining where patterns can be found in the ocean of data generated every second. In order to cope with the volume, velocity and variety of data produced a streaming model has been studied were analysis of data has to use low memory and process items only once, therefore only the summary and statistical information is kept reducing enormously the storage requirements. In the highly connected world we live and the advance of grid and cloud computing the missing component to help crunch this large amount of data is the power of distributed computing. Stream Processing Engines (SPE) have revealed to be a more flexible and powerful tool for dealing with BigData, thus a natural evolution from the batch MapReduce tools. The advantage of SPEs is that of processing an unbound, possibly infinite flow of data and returning results in any time. This project merges the concepts of machine learning, streaming model and distributed computing to build a framework for developing, testing and applying algorithms on large volume of data streams. The Scalable Advanced Massive Online Analysis (SAMOA) project is an initiative from the Yahoo! Research labs in Barcelona and abstracts the use of distributed stream processing platforms by using the concept of code once and run everywhere. Our implementation includes a realtime distributed clustering algorithm and a realtime classification algorithm, which can be run on Twitter Storm and Apache S4 platforms.

Keywords

Keywords

Distributed Computing

Big Data

Streaming Model

Machine Learning

Clustering Algorithms

Stream Processing Engine

Cloud Computing

Index

List of Figures

List of Tables

1 Introduction

The Internet is a worldwide platform for sharing information and making business. It has become a crucial medium for people and companies to connect and present themselves to the world. In this heterogeneous scenario data is represented in many forms and created by many sources. The large amount of information stored and produced is increasing every day and pushing the limits of storage and processing. Only in 2012 there was a rate of 2.5 quintillion bytes of data (1 followed by 18 zeros) created per day. Social networks, sensor networks, e-commerce and other data producing systems generate massive amounts of bytes per second. All of this information can and should be used for analyzing customer behavior, predicting trend and decision-making. Such opportunities have pushed the academy and private companies to develop mechanisms for filtering, processing and analyzing large amounts of data.

MapReduce [2] is a programming model presented by Google for processing large amounts of data in a distributed fashion that can be run on commodity resources, thus scaling the processing power to hundreds or thousands of machines. MapReduce is and innovative way of parallelizing processing jobs where every problem has to be adapted into a two step process: a mapping phase and a reducing phase. The mappers takes an input key/value pair and produce a set of key/value pairs that are passed to the reducing job, which will merge the values into a smaller set. The open source "twin brother" of Google/MapReduce and GFS are the Apache Hadoop[1] and Hadoop Distributed File System (HDFS) projects [3] started at Yahoo!. A next level of the MapReduce model are platforms for processing data in streaming where disk I/O operations are reduced for not using files as its source and storage. Data streaming platforms can process information online, therefore they are a great tool for realtime analysis.

Patterns and relations can be extracted from data using methods from

[1] Apache Hadoop: `http://hadoop.apache.org/`

machine learning (ML) and data mining. Machine learning techniques for classifying information and clustering similar data are some of the main goals of these two areas of study. Such techniques are used in many ways and for different purposes. For example, machine learning algorithms can render very accurate predictive models that can be used to predict the price of housing depending on the size or location. Data mining algorithms on the other hand can deduce if a new piece of information is related to another information. The literature is vast in machine learning and data mining algorithms and they have been developed in various programming languages. Tools and frameworks are also available as commercial and open source products. This variety of algorithms, tools and frameworks exists because there is no "one-size fits-all" solution, it depends on the problem scope, the volume of data and the complexity of the data. This works will focus on the volume and use some data mining techniques that use one common clustering algorithm - the k-means.

The use of machine learning in distributed environments for processing massive volumes of data is a recent research area. Some efforts — such as the open source project Mahout [2] — have been implemented in distributed processing frameworks like Hadoop. Mahout applies machine learning on top of Haddoop/HDFS platform, but it is still a work in proggress. Private and commercial machine learning frameworks exist, but it is not the scope of this project to evaluate these technologies. One drawback of using machine learning in MapReduce frameworks is that they process in batch and cannot deal efficiently with evolving data or frequently updated information. In such cases, having algorithms that uses machine learning algorithms in realtime and live data is of great advantage.

The scope of realtime big data analysis deals with using machine learning algorithms on unbounded streams of data. Data streams can be generated by many different sources such as social networks, sensors, internet traffic, video and many others. To deal with this large volume of possibly unbounded flow of data some distributed stream processing platform have been implemented such as Apache S4[3] and Twitter Storm [4]. These platforms can be considered an evolution of the batch, MapReduce, distributed file system model in the sense that they process flowing data in-

[2]Apache Mahout: `http://mahout.apache.org/`
[3]Yahoo! S4: `http://incubator.apache.org/s4/`
[4]Twitter Storm: `http://storm-project.net/`

stead of always writing and reading from files. Applying machine learning to streaming platforms is the goal of the SAMOA project, where some commonly used machine-learning algorithms will be provided with a pluggable interface for any distributed data stream-processing platform.

1.1 Motivation

This research project is an attempt to create a portable machine learning framework that can work with the various stream processing platforms available. It is also a tool for developers to create new algorithms and test how their implementations scale and behave in a distributed environment, taking into account as many resources available. Using machine learning and data mining on big datasets and on unbounded streams of data is already being used by many entities in the industry and scientific community. Therefore having a flexible, adaptable and tunable open-source tool that can be used by companies, academy and developers easily and efficiently is one of the goals of this project.

1.2 Structure of the Document

The rest of this document is organized as follows. In Chapter 2 there is a review of all the concepts involved in this research, which are applied to the development of SAMOA. Chapter 3 provides an introduction to the different technical areas, projects and studies related to this work. Chapter 4 introduces SAMOA and gives an in-depth view of its design. The details of the distributed clustering algorithm implemented in SAMOA is presented in Chapter 5. The experimental set-up and evaluation is presented in Chapter 6. Finally, Chapter 7 concludes this document by summarizing its main points and future work.

2 Background Concepts

In order to understand better the domain of this project this chapter will present the background concepts and ideas used in this work. The areas involved in this project are machine learning, data mining, streaming model and distributed systems. In these domains some interesting challenges appear in the present information society. The volume of data produced is enormous and much of the produced information is not used due to the lack of resources to store and process them. It would be too expensive to massively store all the information produced, thus inviable. The processing issue is starting to see a feasible horizon, but still has space to evolve. Therefore the important issue is not to store all the data, but to extract relevant statistics, summaries and models from the produced data. To be able to do this there is a need for the technologies presented in the following sections.

2.1 Machine Learning

Machine learning is a branch of artificial intelligence in which systems can learn from data and behave accordingly. Arthur Samuel (1959) defined machine learning as a "field of study that gives the computers the ability to learn without being explicitly programed." The decisions taken from the analysis of patterns and relations between data is not the goal of machine learning itself but of the overall system purpose. An example could be to identify spam emails. A spam filter would apply an appropriate machine learning technique to train the system with a set of real spams and then once a model is built it would apply to new incoming emails and the filtering should be satisfactory if the model is well defined [4].

Extracting information from data is also part of a discipline called Data Mining. These two areas of studies are many time mixed and overlap in many ways. The main difference between both is that machine learning

focuses more on prediction whereas data mining on discovery of patterns and unknown relations on data. A common distinction is the concepts of supervised learning and unsupervised learning. Under supervised learning there is a knowledge about the data label and when correlated a predictive model can be built. On the other hand unsupervised learning there is no information about the data label sand similarities have to be extracted from the data available. Classification algorithms belong to supervised learning and clustering algorithms belong to unsupervised learning. Under further investigation these two techniques can be used together in different ways where clusters of data can be assigned to classes and when new data arrives it can be classified as belonging to a specific class of clustering.

2.1.1 Algorithms

In the world of machine learning and data mining there are many algorithms in supervised, unsupervised and semi-supervised learning. They are used for different goals such as pattern recognition, prediction, classification, information retrieval and clustering. The practical applications of these algorithms are are endless and can permeate any field of study. This project focuses on unsupervised learning, specifically clustering. The concept of clustering in data mining deals with identifying similar information without prior knowledge about the data classes. One issue of clustering is that it is prone to various interpretation and does not have an explicit definition. A famous quote mentioned by Estivill-Castro is that "clustering is in the eye of the beholder", making and allusion to the popular phrase, "Beauty is in the eye of the beholder", where many different approaches can be taken to analyze clusterings [5]. This is also a reason why there are many different kinds of algorithms for clustering.

A widely used algorithms for clustering is the k-means, which purpose is to split a dataset into k distinct groupings. The goal of k-means is to identify similarities between items in a dataset and group them together based on a similarity function. The most common used function is the euclidean distance function, but any other similarity function can be applied. K-means is an interesting options due to its simplicity and speed. In essence it is an optimization problem where a local optimal clustering can be achieved, whereas a global optimal is not guaranteed. It is an heuristic

algorithm in which the final result will be highly dependent on the initial settings, although a fairly good approximation is possible. For this reason choosing the exact solution for the k-means is an NP-hard problem. K-means takes as input a number k of desired clusters and data set $X \in \mathbb{R}^d$. The goal is to choose k centers to minimize the sum of squared Euclidean distance as presented in the following function.

Objective function,

$$\phi = \sum_{x \in X} (min \|x - c\|^2)$$

Historically k-means was discovered by some researchers from different disciplines. The most famous researcher to be coined the author is Lloyd(1957,1982) [6], along with Forgey(1965) [7], Friedman and Rubin(1967) [8], and McQueen(1967) [9]. Since then it has been widely studied and improved. The following Table 1 shows the pseudo-algorithm for k-means. The algorithm works in and iterative way and alternates between two major steps: reassigning the cluster ID of all points in dataset X and updating the cluster centroid based upon the data points in each cluster.

Algorithm 1 $k - means$ algorithm

Require: Dataset X, number of clusters k
  Randomly choose k data points from X
  Use the k points as initial set of cluster representatives C
  repeat
    Reassign points in X to closest cluster mean
    Update m such that mi is cluster ID of the ith point in X
    Update C such that c_j is mean of points in j^th cluster
    Check for convergence of objective function
  until convergence of objective function $F(x)$

  return Set of clusters representatives C, cluster membership vector m

The algorithm is simple and begins by the selection of k points belonging to $\mathbb{R}^d$ space from the desired dataset. Standard k-means has some limitations due to its greedy-descent nature where convergence only happens on local optimums. This characteristic is directly related to the initialization os the centroids. Re-running k-means with different centroids

can improve the results. Avoiding convergence to the local minima as shown in [10] can also improve the final results. As mentioned before, selecting better initial points greatly helps the final outcome, therefore many techniques have been developed with this in mind. Methods for this purpose vary from using random, density based and probabilistic sampling. Others like splitting the dataset in k partitions and picking the centroids from each partition are not good methods. Arthur and Vassilvitskii [11] developed the `k-means++` algorithm, which improves the seeding by using very specific probabilities where the influence of the selected point on the overall potential is taken into account. In a simplistic view, the algorithm goal is to spread the initial seeds apart from each other and avoid seeding the same local clusters. They show that `k-means++` outperforms the standard k-means both in speed and accuracy. A similar, but more simplified version of this seeding is used in this project. The following steps demonstrate the `k-means++` seeding process, which should be injected right after the selection of the initial k data points in the normal k-means. Let the distance function $D(x)$ represent the shortest distance between a point x to the closest center already chosen.

1. Choose one center uniformly at random from the dataset X.
2. Choose the next center from the dataset X with the probability proportional to $\frac{D(x')^2}{\sum_{x \in X} D(x)^2}$.
3. Repeat the previous step until all k centers have been chosen.
4. Proceed with the normal k-means

The k-means complexity is NP-hard for simple configurations such as two dimensional Euclidean space or for an arbitrary number of k clusters in the plane as shown in [12]. Therefore for fixed values of k and dimension d on a dataset with n points the problem can be solved in $O(n^{dk+1} \log n)$ as demonstrated by Inaba et al. in [13].

2.1.2 Tools

K-means clustering is a very common and efficient technique and has been widely applied in many data mining and image processing tools. To name a few it is present in commercial and open-source softwares such as

Windows Excel, MATLAB, WEKA, MOA, Mahout and others. Although the idea behind k-means is simple, there are many challenges when applying k-means to large volumes of data, to data streams and distributed systems.

2.2 Distributed Computing

Distributed computing is the field of studies in computer science where the work is coordinated between a group of machines connected via a network. One can argue that distributed processing can be done in one computer, which is true if the computer is multi-core, thus the communication is in the processing unit level. For this project we will consider distributed computing as processes running in parallel either locally or in a network environment. An essential characteristics of distributed systems is concurrency, where processing is done in parallel and asynchronously; and fault tolerance, where components can fail while maintaining the system running. Examples of distributed systems are peer-to-peer networks, cloud computing platforms, web services, grid computing and the internet itself. A basic view of distributed computing is that of components that work together in a replicated fashion to achieve a common goal without a centralized coordinator. Eliminating a central node can be very complex depending of the system requirements. A master node represents a risk because it is a single point of failure (SPOF), thus leaving the system unavailable if it goes down. There are many techniques that tackle this problem but it is not the objective of this project to present them.

Fault tolerance, replication and coordination is handled by the underlying streaming platforms, therefore our focus is on building a distributed processing topology that will perform well with recent clustering algorithms based on streaming. Most work done on streaming has been focused on a single machine and distribution has been considered in few works. Distribution of ML algorithms has been considered in P2P environment where the data is distributed between machines in a network and the nodes communicate to achieve a global view, which should precise and accurate. Distributed data streaming platforms should not have communication between processes of the same type because they work with direct acyclic graphs (DAG) where the data flows on a "downward" direction. It is possible to introduce cyclic graphs but it may introduce unwanted behavior and risk

of deadlocks.

2.2.1 Why distributed computing?

Distributed systems come in handy when tasks take too long to run in a single machine, or when the volume of data surpasses the amount of memory available. Whenever there is a possibility to distribute and parallelize the work, it is better to do so. By distributing the load between various machines there can be an increase in throughput and a speed-up on the job can be achieved. The distribution and parallelism depends on the nature of the problem. If it is embarrassingly parallel it is a perfect fit for distributed computing, otherwise the load can be distributed but there might be a dependence on results from other machines. In such case the speed-up might be less than expected. Distribution is also important for content storage and peer-to-peer networks are very efficient in this domain. Peers share their resources in a decentralized architecture and use pointers to where the desired resources are located. Distributed computing is widely used in the internet for sharing content and processing resources, as well as in scientific studies with large high-performance multicore machines, which is referred as grid-computing.

2.2.2 Distributed computing platforms and applications

Many distributed systems and computing platforms are available and each is aimed for a specific problem solving. An example is the project SETI@HOME, which uses the idle processing power of computers on screen saver mode to process sounds from outer space. BitTorrent uses peer-to-peer file sharing for a distributed storage of files. Solutions for datacenters are available such as Hadoop, which is a distributed MapReduce framework. Hadoop is being widely used in the industry for recommendation system and processing massive amounts of data.

2.3 Big Data: How big is big?

Big data is a recent term that has appeared to define the large amount of data that surpasses the traditional storage and processing requirements.

Volume, Velocity and Variety, also called the three Vs, is commonly used to characterize big data. Looking at each of the three Vs independently brings challenges to big data analysis.

Volume

The volume of data implies in scaling the storage and being able to perform distributed querying for processing. Solutions for the volume problem are either by using datawarehousing techniques or using parallel processing architecture systems such as Apache Hadoop.

Velocity

The V for velocity deals with the rate in which data is generated and flows into a system. Everyday sensors devices and applications generate unbounded amount of information that can be used in many ways for predictive purposes and analysis. Velocity not only deals with the rate of data generation but also with the speed in which an analysis can be returned from this generated data. Having realtime feedback is crucial when dealing with fast evolving information such as stock markets, social networks, sensor networks, mobile information and many others. Aiming to process these streams of unbounded flow of data some frameworks have emerged like the Apache! S4 and the Twitter Storm platforms.

Variety

One problem in big data is the variety of data representations. Data can have many different formats depending of the source, therefore dealing with this variety of formats can be daunting. Distributed key-value stores, commonly referred as NoSQL databases, come in very handy for dealing with variety due to the unstructured way of storing data. This flexibility provides and advantage when dealing with big data. Traditional relational databases would imply in restructuring the schemas and remodeling when new formats of data appear.

2.3.1 Data streaming platforms

Data streaming platform is another name for Stream Processing Engines (SPE), which are softwares that are built to process potentially unbounded

streams of data. There are many platforms available for stream processing, but most are commercial and will not be discussed here. Early SPEs were constructed as centralized applications that worked on a single machine. As the three Vs concept of BigData became more present on streaming SPEs had to evolve to a distributed environment.

The first SPEs developed were the Aurora [14], STREAM [15] and TelegraphCQ [16] which operated in a centralized fashion. As distributed systems and parallel computing came in, new prototypes were created such as Borealis [17], which is an evolution of Aurora, and Medusa. The most recent SPEs are the Apache! S4 and Twitter Storm. They are inspired in the SPEs and MapReduce model, which uses the actor model theory to build a naturally parallel data processing platform.

The actor model proposes that every entity is a computational entity and they communicate through asynchronous message passing. Actors extend the concept of objects to concurrent computation [18]. Since the actors execute in parallel and communicate asynchronously they are independent processing elements. In this way actors can be created, reconfigured and deleted without compromising the system, that is if there are actors alive and running. This decoupling and independence of execution makes concurrency simpler and easier for developers. Such design withdraws from the developer the concurrency management.

2.4 Data Streaming Algorithms and Applications

A formal definition for data stream S is that of a sequence of arriving data objects x^1, x^2,..., x^N, rendering $S = \{\mathrm{x}^i\}_{i=1}^{N}$, where $(N \to \infty)$. Each data object is an n-dimensional attribute vector $\mathrm{x}^i = [x_j^i]_{j=1}^{n}$ belonging to an attribute space that can be continuous, categorical, or mixed.

In order to analyze data stream algorithms have to be designed to deal with some inherent properties of streams.

- Objects arrive continuously.
- Stream sizes can be unbounded and potentially infinite.
- There is no guarantee in arrival order of data objects.
- Data objects are discarded after being processed.
- The data source might be evolving.

In order to comply with these properties the data stream model has some constraints. Data from the streams cannot be stored due to the large volume, therefore only summaries should be processed and stored. The speed of arrival is fast, thus each item has to be processed in realtime or only once. Data from past might become irrelevant and can even affect negatively the current summaries.

Dealing with these constrains have a direct impact in memory usage and data "age" relevance. Some techniques have been developed to deal with these problems such as sliding window combined with low memory functions. Sliding window is a technique where a window of size W keeps the last W items that arrived and run a desired algorithm on these items. The summary is then kept and the items are then discarded. In this way an up to date statistics is always computed. These statistics are usualy kept by updating the model; in case of classification is could be the decision tree and in clustering could be the final clustering centriods. Data streaming algorithms have been studied by large group of researchers and there is a wide literature on the different techniques which can be found in [19].

3 Related Work

Data mining and machine learning have long since been studied and there is a vast literature on these subjects. A recent and more challenging approach for machine learning and data mining is to deal with the large amount of data produced everyday by the Internet and other systems. Finding patterns on data is very relevant for decision-making and having this information fast or on realtime is a bonus. Leaders and decision makers need to respond fast to evolving trends and behaviors.

Scaling traditional algorithms is not a trivial task due to the fact that speeding up the processing of terabytes is not only dependent on the efficiency of the algorithm but on the hardware available. The algorithms in their essence have to be fast and use the least processing possible to performs large tasks. Therefore to take the most advantage of the resources it is essential to distribute the computation between various machines. Applying machine learning algorithms in distributed fashion imply in altering how the algorithm works, since information has to be shared between the machines. Performing this change and still maintaining the accuracy and performance of the algorithm can be daunting. Communication has to be minimal not to over-use the network, which usually is the bottleneck in a distributed system. To maintain accuracy is to render results that are the same to the original algorithm running on a single machine. Goswami [**?**] exposes that it is not known of any technique that can perform so well in parallel, therefore the technique that has better approximations the best. Many approaches have been developed to tackle this problem.

3.1 Distributed Clustering Algorithms

In [20] some distributed clustering approaches are mentioned and they are separated into two categories; multiple communication round algorithms and centralized ensemble-based methods. The multiple commu-

nications methods require synchronization steps, which incurs in a large amount of message passing. The centralized approach use asynchronous communication and build local models that are transmitted to a centralized aggregator that build a global model. Using asynchronous methods are more interesting due to the independence of components where in a large setting with many processes running there is no problem of slow processes affecting the execution of the algorithm. Some approaches deal with batches but can easily be adapted to the streaming model.

3.2 Stream Clustering Algorithms

Stream clustering algorithms have been developed as an adaptation of traditional clustering algorithms to fit the streaming model and comply to its constraints. Different techniques were created to deal with evolving data such as one pass processing and summarization. Algorithms that can work on streams and still maintain good results as their batch processing relatives. Additionally to how data is processed, some data structures were developed to deal with the memory usage. Stream clustering can be characterized by two steps: data abstraction (also referred as the online component) and the data clustering (also referred as the offline component). The online component deals with extracting only the relevant information in specific data structures to be used later on the offline component step. There are four commonly used data structures: feature vector, prototype array, coreset trees and grids []. For this work we will only consider two - feature vectors and coreset trees - since they are used by algorithms which have k-means as its core clustering algorithm [].

Following is a list of the 13 most relevant data stream clustering algorithms. Each one of them has improvements in performance, memory usage, computational complexity, clustering quality and scalability.

- BIRCH [Zang et. al 1997]
- Scalable k-means [Bradley et al. 1998]
- Stream [Guha et al. 2000]
- Single pass k-means [Farnstrom et al. 2000]
- Stream LSearch [O'Callaghan et al. 2002]

- CluStream [Aggarwal et a;. 2003]
- DenStream [Cao et al. 2006]
- D-Stream [Chen and Tu 2007]
- ODAC [Rodrigues et al. 2006; 2008]
- SWClustering [Zhou et al. 2008]
- ClusTree [Kranen et al. 2011]
- DGClust [Gama et al. 2011]
- StreamKM++ [Ackermann et al. 2012]

All of these algorithms were designed for a single machine and did not take into account a distributed environment. This work focuses on using one of these algorithms — CluStream — in a distributed environment. The goal is to be able to use or implement any of these algorithms in SAMOA without modifying the algorithm design. For the moment Clustream was chosen because it has a stable and reliable implementation in MOA, which will be used as the baseline for some of the experiments.

3.3 Stream Machine Learning Frameworks

Applying machine learning and data mining algorithms in streams of data has become very popular due to its appliances in various scenarios where streams of data are produced. For example, such algorithms can be used on sensor networks, monitoring of power consumption, telephone logs, internet traffic, social networks and many others. Many tools and frameworks are available as commercial and open source projects. The Massive Online Analysis (MOA) framework is a software environment that contains machine learning algorithms for learning on evolving data streams. MOA is related to another project, the Waikato Environment for Knowledge Analysis (WEKA), which is a workbench for batch processing of ML algorithms [21]. Stream processing is different from batch in the sense that the dataset is potentially infinite. In order to process data that arrive in high speed some requirements have to be taken into account. Since data is arriving continuously the memory can easily be filled, thus a low memory approach has to be used. Each example has to be processed at a time and at most once; this is known as a one-pass approach. Another requirement

is that the algorithms should be able to provide a prediction or summaries ate any time, therefore the models have to be constantly updated.

Other open-source data streaming tools are the Very Fast Machine Learning (VFML) toolkit and the Yet Another Learning Framework (YALE). In 2007 YALE became RapidMiner[1] and VFML has been copyrighted since 2003 [22] .

3.4 Distributed Machine Learning Frameworks

As data sets increase in size and are geographically distributed; a centralized approach is not suitable anymore. Some attempts that tackle this problem of large data sets physically distributed includes the use of grid computing, peer-to-peer paradigm and web services.

Mario Cannataro et al. [23] developed the *Knowledge Grid* framework for data mining in distributed grid environments using SOA and web services. Grid computing model is different from traditional distributed computing due to the use of large-scale resources and high performance machines. IN [24] Giannadakis et al. presented *Info Grid* mostly focused on data integration for knowledge discovery. Similar systems with grid are the ones that use distributed shared memory with multiprocessors, using Message Passing Interface (MPI) for communication. In Dhillons et al. work they calculate the speedup in a parallel execution of the k-means algorihtm and also the scaling-up in numbers of dimensions, clusters and data points [25]. Google developed the MapReduce [2] and Yahoo! developed Hadoop, which open-sourced this programming model. Further on, Apache Mahout was developed to use data mining on top of Hadoop. Another common approach for distributed machine learning is using the peer-to-peer paradigm where all nodes have the same responsibilites, with no master nodes or hierarchies.

Such distributed frameworks scale to thousands of machines and huge datasets, but they have the downside of working on batch. Therefore whenever there is an update to the dataset the whole batch has to be reprocessed. This can take hours,days or weeks to render a result.

[1]RapidMiner: `http://rapid-i.com/`

3.5 Distributed Stream Machine Learning

Leveraging machine learning and data mining to render realtime results is the goal of distributed stream machine learning. Companies do not want to wait overnight to offer new up-to-date recommendations, or to take decisions. Realtime big data analysis is an opportunity to integrate all the positive aspects of machine learning, distributed processing environments and data streaming. Some emerging technologies are being built to deal with this problem. Jubatus [26] is a framework for distributed stream machine learning, which includes classification, regression, recommendation and graph mining. Meanwhile it does not include stream clustering techniques.

The advantage of processing streams in a distributed environment is that it has the ability to scale massively and render results at anytime, without the latency inherent to large volume batch processing. It also has the advantage of using a small amount of memory, because it only maintains the summaries of the clustering and free space for processing other incoming points, or other necessary calculations.

3.6 Summary

Machine learning has become popular and has evolved to become an essential tool for predictive analysis and data mining. This popularity has resulted in the development of many tools for specific and generic uses.

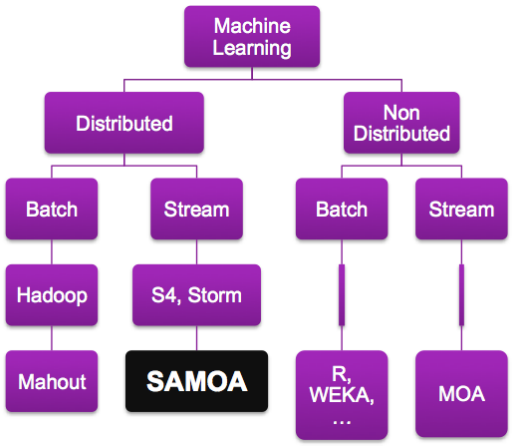

Figure 3.1: View of machine learning tools taxonomy.

This chapter presented some of the most important efforts in applying machine learning to clustering problems in different kinds of infrastructure, architecture and data models. There is still space for advances in this area and the Figure 3.1 illustrates the current state of machine learning tools taxonomy and points where SAMOA fits in this current scenario.

4 Scalable Advanced Massive Online Analysis

Everyday increasing computing power needs tools that can make the best use of the available resources. Traditional tools such as WEKA and the R[1] language offer machine learning algorithms and development environment for use on single machines. MOA, as mentioned in section 3.3, was developed for applying machine learning on data streaming, but also has the constraint of working on a single machine. Taking advantage of distributed processing platforms, Mahout is a framework that runs on top of Hadoop and provides batch machine learning processing on a MapReduce model. Mahout is a very powerful tool and is rapidly being adopted by the industry. Mahout has two issues that can impose some difficulties in developing distributed big data machine learning; the algorithms have to be adapted to work on the MapReduce model and due to the batch processing nature it can be hard to use it for frequently updated data.

Scalable Advanced Massive Online Analysis (SAMOA) is a framework and a library that includes distributed machine learning for data streams with an interface to plug-in different stream processing platforms. SAMOA can be used in two different scenarios; data mining and machine learning on data streams, or developers can implement their own algorithms an run them on production. Another aspect of SAMOA is the stream processing platform abstraction where developers can also add new platforms by using the API available. With these separation of roles the SAMOA project is divided into SAMOA-API and SAMOA-Platform. The SAMOA-API allows developers to develop for SAMOA without worrying about which distributed SPE is going to be used. In the case of new SPEs being released or the interest in integrating another platform, a new SAMOA-Platform module can be added. The first release of SAMOA will support two SPE that are the state or the art on the subject matter; Apache! S4 and Twitter Storm.

[1]R language: `http://www.r-project.org/`

4.1 Apache! S4: Distributed Stream Computing Platform

It is important to understand how the underlying SPEs work in order to develop an abstraction that complies with the different designs available. In this section there is an overview of Apace! S4 internals, since this project was focused on implementing the S4 adapters for SAMOA.

S4 (Simple Scalable Streaming System) is a distributed SPE based on the MapReduce model and uses concepts of the Actors model to process data events. It uses Processing Elements (PEs) which are independent entities that consumes and emits keyed data events in messages. As mentioned in Chapter 2 various commercial engines apply distributed stream computation but are application specific. A review of these systems can be found in [27]. S4 is intended to be an open-source general purpose stream computing platform with a simple programming interface. The basic design principles take into account high availability, scalability, low latency and decentralized architecture [28]. The distributed an symmetric design removes the need for a central master node. The design of S4 borrows many concepts from IBM's Stream Processing Core (SPC) middleware, which is also used for big data analysis [27]. The messaging is implemented as push-based where events are emitted into a stream and directed to specific PEs regarding the event keys. The underlying coordination is done by Zookeeper [1] which assigns S4 tasks to physical cluster nodes. An internal view of a S4 processing node is shown in Figure 4.1. The S4 system is built and deployed in a logical hierarchical model. A deployment of S4 will contain the following components:

- Cluster: a logical cluster can be set up to deploy many S4 applications that can be interconnected by different event streams.
- Node: a node is assigned to a logical cluster and will be available to run a S4 application.
- App: the S4 App runs in a Node and encapsulates tasks that consume and produce event streams. An App contains a graph of PEs connected by Streams.
- PE Prototype: is a definition of a computation unit which contains processing logic.
- PE Instance: is a clone of a PE prototype that has a unique key and state.

- Stream: connects PE instances. A PE emits events in a stream and consumes events from a stream.

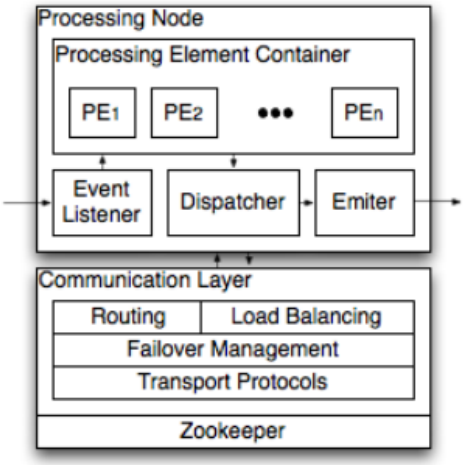

Figure 4.1: S4 processing node internals extracted from [1].

4.2 SAMOA Modules

SAMOA has been designed in a modular fashion to be easily adaptable for new components. In its modular architecture there are the core algorithms and APIs for developers, the interface for external ML algorithms, such as MOA integration, and the SPEs adapter that interfaces with external stream processing platforms. The Figure 4.2 illustrates the SAMOA design in a high level. The execution process is started by specifying the deployment platform and a task with specific parameters. The task is fetched and starts the topology on the specified platform.

4.2.1 SAMOA-API

SAMOA-API is the abstraction for implementing a data processing topology. Its main components are *Processors*, *Processing Items (PI)* and *Streams*. *PIs* contain *Processor* objects. Inside the processors is where the logic and algorithms are coded. The *Streams* provide the connectivity between the *PIs*, therefore one stream always has to be linked to two *PIs*, unless it is an external stream. In order to use external streams an *EntranceProcessingItem* was introduced to interface with external "world" or

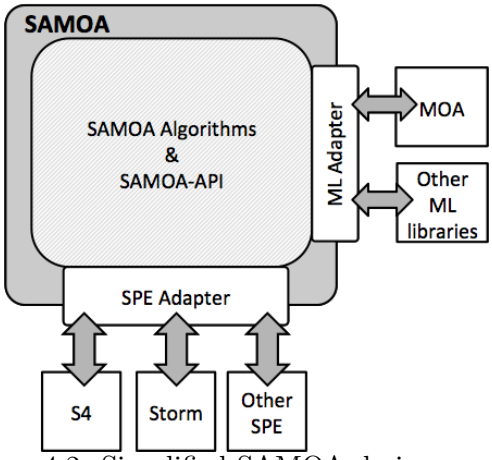

Figure 4.2: Simplified SAMOA design overview

other applications. *Streams* "carry" *ContentEvents* which is a wrapper for the actual data that will be processed. *ContentEvents* can be extended to implement any data structure, depending on the application needs. These components are the nuts and bolts of the SAMOA-API. The assembling part is done by components called *Topology*, *TopologyBuilder* and *Tasks*. The *TopologyBuilder* is used to build and connect the *PIs* and the *Streams* in a *Topology*, whereas *Tasks* is where the topologies are created.

4.2.2 SAMOA Platform Specific Modules

The platform modules are implementations specific for a stream processing platform. Meanwhile SAMOA only supports Apache! S4 and Twitter Storm, but can be extended to support others. To implement a platform specific module the developer needs to implement the *PIs* and *Streams* abstractions that relate to the specific components of the platform and how they should be connected. In addition to these two wrappers two other components have to be implemented; the *ComponentFactory* and the *TopologyBuilder*. As their name suggests the *ComponentFactory* implements the factory pattern to provide platform specific items such as processing items and streams, whereas the *TopologyBuilder* uses the *ComponentFactory* to create and build the desired topology with the platform specific components.

An important aspect to note is how the streams are connected to the processing items. Due to specific algorithm design issues we decided to provide three different kind of connections; *Shuffle*, *Key* and *All*. The con-

nection type will determine how the events will be sent to the processing items. The following list defines each connection type:

- *Shuffle*: this connection type distributes the events in a round robin fashion among the destination processing items.
- *Key*: this connection type will use the event key to route the even to the responsible processing item. For example, a processing item is responsible for events with keys from 0 to 3, another for keys 4 to 8, therefore these events will only be processed by their assigned processing item.
- *All*: this connection is special because it sends events in a broadcast manner, where all available processing items receive a copy of the original event.

4.2.2.1 SAMOA-S4

The SAMOA-S4 is a built in module specific to plug-in the Apache! S4 distributed stream processing platform. Apache! S4 is a general purpose, distributed, scalable, fault-tolerant, pluggable platform that allows programmers to develop applications for continuos unbounded stream of data inspired on the MapReduce and Actors model. S4 core items for building a topology are *Processing Elements*, *Streams* and *Events*, which can be directly mapped to SAMOAs *PIs*, *Streams* and *ContentEvent*.

4.2.2.2 SAMOA-Storm

Just like S4, Storm is a distributed realtime computational system for processing unbounded streams of data. Although Storm and S4 are platforms for processing streams of data; they have different design. Storm specific entrance processing items are called *Spout* and normal processing items are called *Bolts*. The events in Storm are tuples of key-values. Storm works in a pull based model, therefore the SAMOA-Storm module has to consider this and provide a consistent abstraction for the *PIs*, *Stream* and *ContentEvents*.

4.3 How does it work?

In practice the use of SAMOA is quite similar as the underlying platforms where the developer implements the business rules in the processing items and connect them by streams. A different aspect is that the logic implemented in the Processor instead of the PI directly; this is a design pattern where the PI is a wrapper allowing the processor to run in any platform. One detail to keep in mind is that once implemented for SAMOA the code can run on any platform which has an adapter. The following code snippet demonstrates the simplicity of the SAMOA-API. This code builds a simple source entrance PI and one other PI connected by a stream.

```
TopologyBuilder topologyBuilder = new TopologyBuilder();
EntranceProcessinItem entranceProcessingItem =
    topologyBuilder.createEntrancePI(new SourceProcessor( ));
Stream stream = topologyBuilder.createStream( entranceProcessingItem );
ProcessingItem processingItem = topologyBuilder.createPI(new Processor( ));
processingItem.connectInputKey(stream);
```

With this simple API the topology can be extended to create larger graphs of data flow. This model provides a greater flexibility for designing distributed algorithms than the MapReduce model, which can only have sequences of mappers and reducers. Therefore it is easier to adapt common algorithms to S4, consequently to SAMOA. The Figure 4.3 illustrates how the SAMOA framework maps to the S4 system. Notice that the task present in SAMOA is a wrapper for the PIs and stream graph, whereas in S4 this would be implemented directly on the App component.

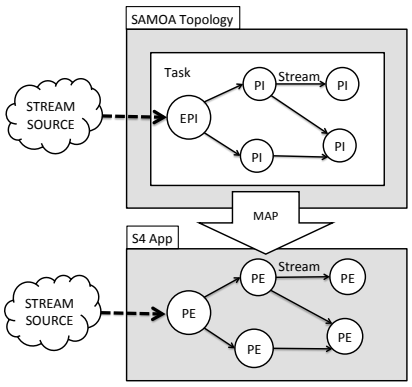

Figure 4.3: Mapping of SAMOA elements to S4 platform.

5 Distributed Clustering Design

In order to validate SAMOA we implemented a distributed clustering algorithm based on the CluStream framework presented by Aggarwal et. al in [29]. The decision for choosing CluStream was due to its good clustering accuracy, dataset scalability and handling of evolving data streams. In addition to the aforementioned aspects, CluStream implements the well known k-means algorithm. The goal was to design a distributed clustering algorithm in SAMOA that is accurate and performs as good as the CluStream implementations in other machine learning tools, such as MOA. SAMOA Clustering algorithm will be evaluated on its accuracy, volume scalability and resource scalability.

5.1 CluStream Algorithm

CluStream is a framework for clustering evolving data streams efficiently. As mentioned before, the clustering problem aims to partition a data set into one or more groups of similar objects by using a distance measure. Since stream clustering cannot maintain all the information used due to memory limitation and neither revisit past information, the algorithm has to keep a small summary of the data received. CluStream efficiently deals with this problem by using an online component and an offline component. The online component analyses the data in one-pass and stores the summary statistics, whereas the offline component can be used by the user for querying the cluster evolution in time. In order to maintain these summary statistics CluStream uses a micro-clustering technique. These micro-clusters are further used by the offline component to create higher level macro-clusters [29].

Every algorithm uses specific data structures to work; in stream clustering algorithms it is not different. CluStream uses specific data structures called *Cluster Feature*(CF) vectors to summarize the large amount of data

on its online phase. This data structure was first introduced in the *BIRCH* algorithm [30]. CF vectors are composed of the number of data objects (N), the linear sum of data objects (LS) and the sum of squares of data objects (SS). In more details the LS and SS are n-dimensional arrays. The CF vectors conserve the properties of incrementality and additivity, which are used to add points to the CF vector or merge CF vectors. In CluStream the CF vectors are referred as micro-clusters with additional time components and called $\overline{CFT}(\mathcal{C})$ for a set of points $\mathcal{C}$. The time components are the sum of timestamps (LST) and the sum of squares timestamp (SST). The incrementality and additivity properties are presented bellow.

1. Incrementality: A new arriving point $x \in \mathbb{R}^d$ can be inserted into a CF vector by updating its statistics summary as follows:

$$LS \leftarrow LS + x$$
$$SS \leftarrow SS + x^2$$
$$N \leftarrow N + 1$$

2. Additivity: A new CF vector can be created by merging two disjoint vectors CF_1 and CF_2 as follows:

$$N = N_1 + N_2$$
$$LS = LS_1 + LS_2$$
$$SS = SS_1 + SS_2$$

With these data structures in hand it is possible to compute the means, the radius and the diameter of clusters as it is represented in the equations 5.1, 5.2, 5.3.

$$centroid = \frac{LS}{N} \tag{5.1}$$

$$radius = \sqrt{\frac{SS}{N} - \left(\frac{LS}{N}\right)^2} \tag{5.2}$$

$$radius = \sqrt{\left(\frac{2N * SS - 2(LS)^2}{N(N-1)}\right)} \tag{5.3}$$

The aforementioned properties are used in CluStream to merge or create micro-clusters. The decision on creating or merging is based on a boundary factor that is relative to its distance to the closest cluster. In the case of creating a new micro-cluster the number o clusters has to be reduced by one to maintain a desired amount of cluster. This is done to save memory and can be achieved by either removing or joining old clusters. In this way CluStream can maintain statistical information of a large amount of dominant micro-clusters over a time horizon. Further on these micro-clusters are used on an offline phase of the algorithm to generate higher level macro-clusters.

The offline phase of the algorithm takes as input a time-horizon h and a number of higher level clusters k. The value of h will determine how much history has to be covered by the algorithm and the value k will determine the granularity of the final clusters. Lower h values retrieves more recent information and higher k renders more detailed clusters. In this phase the macro-clusters are determined by the use of a modified k-means algorithm. The k-means algorithm uses the micro-clusters centroids as *pseudo-points* which are the clustered in higher level clusters. The initialization of the k-means is modified to sample seeds with probability proportional to the amount of points in the micro-cluster. The new seed for a macro-cluster is defined as a weighted centroid for that partition.

5.2 SAMOA Clustering Algorithm

In SAMOA a distributed version of CluStream was implemented using the SAMOA API. The general design of the clustering data flow was divided in three major components represented by the following processing items: data distribution (*DistributrionPI*), local clustering (*LocalClusteringPI*) and global clustering (*GlobalClusteringPI*). In addition to the clustering components a separate module for evaluation was created with a sampling processing item (*SamplingPI*) and an evaluation processing item (*EvaluatorPI*). The evaluation components are used to asses the clustering algorithms by using pre-defined measures, as will be explained in Section 5.3. These components are illustrated in Figure 5.1. This design allows pluggable clustering algorithms to be used in both the local clustering phase and the global clustering phase, where the global clustering uses the output of the local clustering. The definition of each processing item is as

follows:

- *DistributionPI*: this processing item can either generate a source stream or receive an external stream to distribute between the *LocalClusteringPI* instances. The distribution will depend in the connection type defined in Section 4.2.2.
- *LocalClusteringPI*: this processing item applies the desired clustering algorithm to the arriving stream and outputs the results as events in the stream connecting the *GlobalClusteringPI*.
- *GlobalClusteringPI*: the global clustering is responsible to aggregate the events from the local clustering and generate a final global clustering, which can be outputted to the *EvaluatorPI* for evaluation or to any output (file, console or other applications).
- *SamplingPI*: the sampling processing item is part of the evaluation module and will only be activated when the evaluation mode is active. It is responsible for sampling the source stream by a chosen threshold from 0 to 1, where 1 forward all the incoming events to the evaluation processing item. Another feature of the *SamplingPI* is that it also generates the ground truth clustering and ground truth evaluation measures to be used as external evaluations. This is only possible when the ground truth is available in the source stream.
- *EvaluatorPI*: the evaluator processing item can apply any desired measures to the clustering result. The measures adopted for SAMOA Clustering are of intra cluster cohesion and inter cluster separation, which are detailed in the next section.

5.3 Evaluation Criteria

Assessing and evaluating clustering algorithms can be a very challenging task. Jain and Dubes mention in 1988 that "the validating of clustering is the most difficult and frustrating part of a cluster analysis" [31]. This statement reflects what Estvill-Castro said about the clustering having different interpretations depending on the perspective. Clustering can be

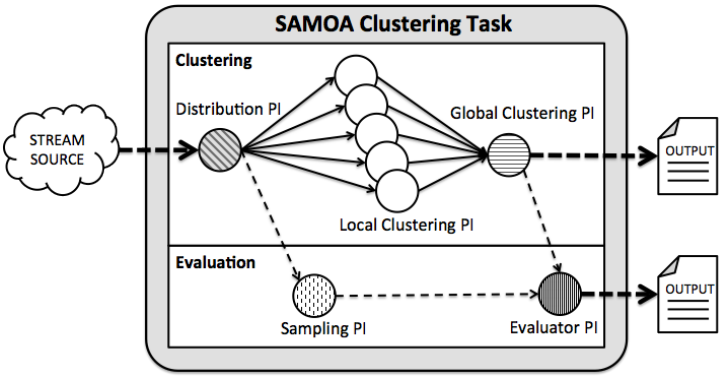

Figure 5.1: SAMOA Clustering Task topology.

evaluated in two main categories: *external* and *internal* evaluation. The main difference between them is that external validation takes into account the matching against some external structure, whereas the internal only uses its internal attributes for validation. An external structure can be represented by the ground truth, present on synthetic data but not often found on real data. In the literature there are many different measures to validate clustering and a review of some external measures can be found in [32]. For the purpose of this project the measures chosen was that of *cohesion* and *separation* that can both be used as internal and external evaluations by following some conditions. The following items describe in details the measures:

- *Cohesion* is the measure of how closely related are the items in a cluster. It is measured by the sum of square error (SSE) between each point x in a cluster C and the cluster mean m_i.

$$SSE = \sum_i \sum_{x \in C_i} (x - m_i)^2 \tag{5.4}$$

- *Separation* measures how distant the clusters are from each other. This is achieved by calculating the between-cluster sum of squares (BSS), taking into account the overall clustering mean m. This measure is also directly related to the weight (amount of points) in a cluster C_i.

$$BSS = \sum_i |C_i|(m - m_i)^2 \tag{5.5}$$

Highly cohesive clusters might be considered better than less cohesive. With these measures some actions can be taken, such as splitting less co-

hesive clusters and merging not so separate ones. An interesting property of these measures is that they are complementary and their sum renders a constant. Therefore the measures for a clustering with $k = 1$ will give a value of BSS equal to zero, therefore the value of SSE will be the constant. Taking this into account the measures are good candidates for evaluating the SAMOA distributed clustering design. Since the cohesion can only be measured if all the points available, it can only be assessed when the evaluation components are active. On the global phase of the clustering the separation factor (BSS) can be found because the final clustering are weighted clusters. Further on the cohesion (SSE) can be inferred by calculating the difference between the constant and BSS. When the ground truth is available it is possible to find the value of SSE and BSS by using the real clustering centroids. More on the evaluation method will be discussed in the next chapter.

6 Experiments and Evaluation

The experiments on the SAMOA distributed clustering algorithm were done with the goal of assessing two main criterions: scalability and clustering quality. In order o evaluate the scalability the algorithm will be run with an increasing amount of resources, thus the performance can be analyzed in different environment configurations. The expected behavior should be an increase in performance while scaling up the system. The clustering quality will be assessed with different dataset configuration, where a noise factor will be added to the data stream. It is known that the k-means clustering algorithm is sensitive to outliers, therefore the clustering algorithm might find poor results in face of increasing noise in the dataset.

6.1 Test Environment

The experiments were conducted on a cluster of four high memory server. Each server is equipped with a dual-quadcore Intel Xeon processor of type E5620 with clock speed of 2.4MHz and hyperthreading, which provides the processing power of 16 cores, and 48GB of RAM. Synthetic datasets of evolving data streams were used to validate the algorithm. The data set generator takes as input the k amount of clusters, the dimensionality of points, the noise threshold and the speed of cluster movement. The clustering algorithm also takes as input the k amount of final clusters to be found, the amount of intermediary microclusters and the parallelism level. The SAMOA framework will have as input the amount of nodes to be launched. All experiments were conducted with node parallelism of four, which means that the parallel processes are distributed amongst the available nodes. The parameters used for the experiments are listed in the Table 6.1. A special attention on the experiments with different speeds. The speed mechanism moves the cluster kernel a predefined distance of 0.1 units every N points. The value of the last experiments was set 12000

Parameters			
Speed	Dimension	Noise	Parallelism
500	3	0.0	1
500	3	0.0	8
500	3	0.0	16
500	3	0.1	1
500	3	0.1	8
500	3	0.1	16
500	15	0.0	1
500	15	0.0	8
500	15	0.0	16
500	15	0.1	1
500	15	0.1	8
500	15	0.1	16
12000	3	0.0	1
12000	3	0.1	1

Table 6.1: Parameters used on the experiments, summing up to 14 different configurations.

so that the clusters would move slower and the cohesion and separation measures could be better visualized. The experiments with speed 500 were evaluated every 5 seconds and the last two were taken every 2 seconds. The speed does not interfere on the results because the scalability evaluation can be measured separately from the clustering quality. The data set generator creates a stream of random instances and a predefined amount of instances limit can be defined. For the purpose of the experiments the limit used was of 5 million instance, which made the experiments last enough time to gather relevant results.

The next Sections 6.2 and 6.3 will present the experiments results for scalability and clustering quality in charts and plots. Further on in Section 6.4 there will be a discussion on the results and on possible future works to improve the evaluation of SAMOA.

6.2 Scalability Results

Measuring scalability is essential for evaluating distributed systems and how well a system scales will determine the amount of maximum resources that can be allocated for a task. The Figures 6.1, 6.2 and 6.3 show the results on the scalability level. In order to determine the ground level of SAMOA we tested the algorithm against a well known stream machine learning tool — MOA — and the baseline is plotted in Figure 6.1.

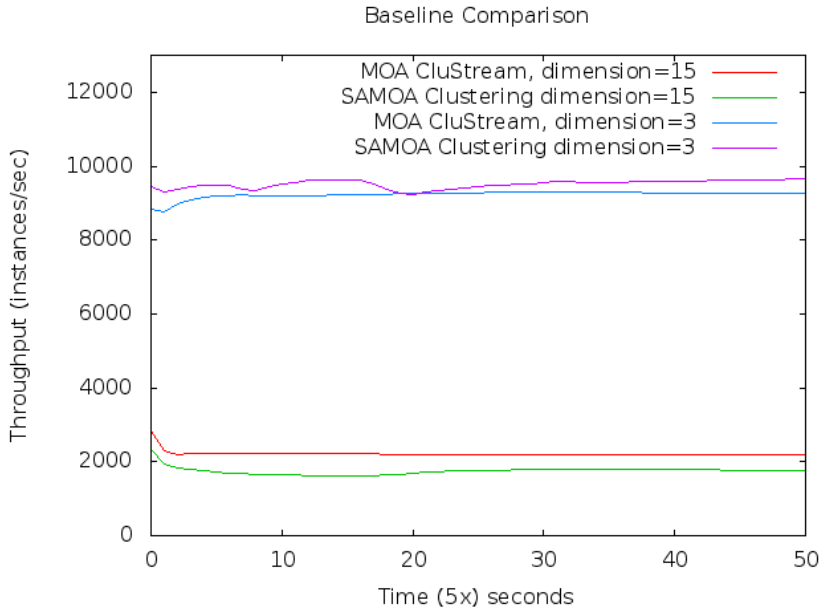

Figure 6.1: Baseline results in comparison with MOA.

As can be noticed by the timeline plot the SAMOA algorithm performs close to MOA in relation to the throughput. On instances with 3 dimensions SAMOA and MOA can process around 9000 instances per second, with SAMOA having slightly better performance on some times. When dealing with instances of higher dimension, in this case 15, the throughput drops to around 2000 instances per second due to the complexity of the computation. MOA performs better on instances with dimension 15; this might be due to the layered design of SAMOA, although a more rigorous experimentation should be done to assess this result.

The next step in the experiments was to measure the overall average throughput over a period of time and increase the parallelism level using instances with dimensions 3 and 15. The results are illustrated in the bar chart on Figure 6.2. The first bars shows values that can relate to the base-

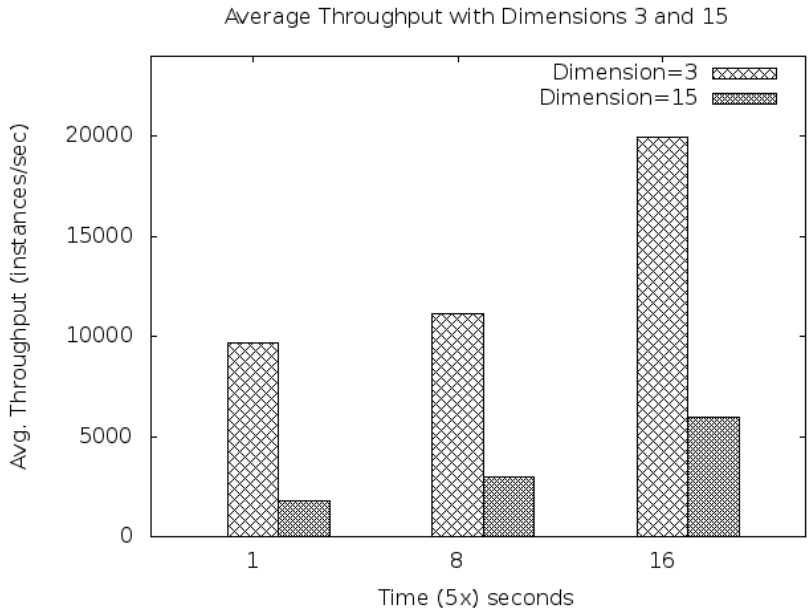

Figure 6.2: Overall average throughput for instances with dimensionality 3 and 15 using different parallelism configuration.

line, where the processing of instances with dimensions 3 and 15 renders a throughput of around 10 thousand and 2 thousand respectively. The bars of parallelism level 8 and 16 show an increase in the performance. Increasing the parallelism from 8 to 16 gives almost a 2x performance increase. The maximum average performance is seen with parallelism 16 where the SAMOA clustering algorithm reaches 20 thousand instances per second with 3 dimensional instances and 5 thousand for dimension 15.

A good way to visualize the performance of the algorithm on a stream model is to see its behavior over time. The line plot on Figure 6.3 shows the average cumulative throughput over time with different parallelism levels for 3 dimensional instances. All three configurations reach a plateau indicating the threshold presented on Figure 6.2. The line representing the parallelism level 8 has a stable performance since the beginning of the execution, whereas the other two have a time span of increasing performance until reaching a stable threshold.

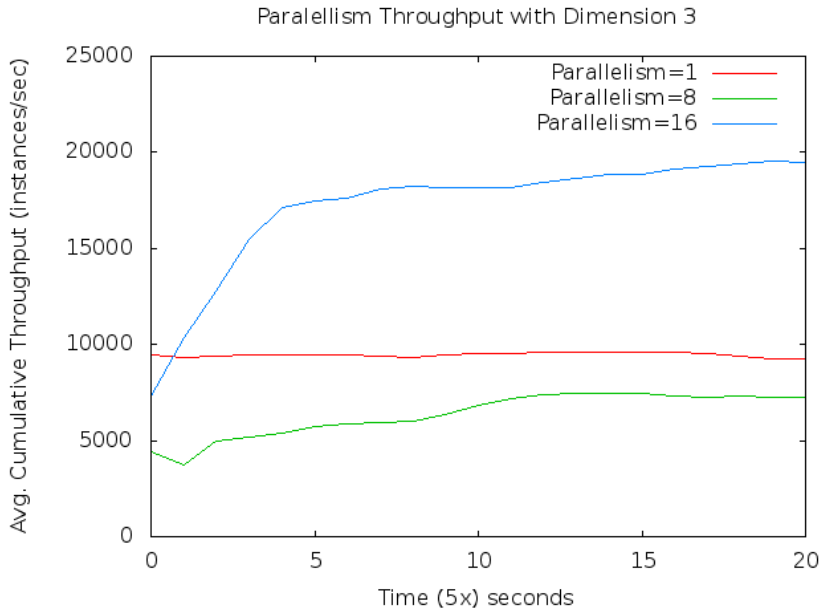

Figure 6.3: Throughput over time with different parallelism 1,8,16 and dimensionality 3.

6.3 Clustering Quality Results

Assessing the quality of clusters is an important part of data mining and are essential to validate the clustering algorithms. This section presents the results of the cohesion (SSE) and separation (BSS) experiments on evolving data streams. The following plots show how well the algorithm works by comparing internal and external measures to the ground truth provided by the stream generator.

Figure 6.4 shows the BSS, which is the between-cluster separation factor for internal, external and ground truth measures. In spite of the confusing overlapping of results, it shows that on a dataset with 0% of noise the SAMOA clustering can follow relatively well the ground truth. For external measures, where the algorithm uses all the instances received to calculate the BSS it is quite accurate, as can be seen by the overlapping lines. On the other hand, the internal evaluation — where only the weighted microclusters are used — deviates from the ground truth from time steps 50 to 90. This behavior can happen once clusters start getting closer to each other. Further on the Figure 6.5 shows the ratio relation between the BSS measures and the ground truth. On this plot, the closer the values are to 1 the better. The external evaluation keeps most of it values between 0.8 and 1 rendering a good approximation. For the internal

measure an accentuated drop is present from time steps 50 to 90, which reflects the gap present in Figure 6.4 during the same period.

The previous experiment and measures can be viewed as the baseline for clustering quality. In cases where there are gaps is where decision should be taken as in merging clusters that are too close to each other. The next plots show the algorithms behavior under a more "polluted" dataset, where a noise factor of 10% is introduced. Adding noise produces some outliers that can affect the clustering results; outliers are quite common in real datasets. Both plots in Figure 6.6 and 6.7 present the same metrics as in Fig 6.4 and 6.5 respectively, but with an additional noise factor. By looking at the BSS plot one can see that the internal and external measures take very different paths from the ground truth and from each other, although they follow similar overall "wavy" movement. In this case the internal evaluation renders a smooth path due to the fact the it uses the weighted centroids as items and is less affected by the noise present, whereas the external is more affected by the presence of outliers arriving. Figure 6.7 shows a relative similar behavior on the ratio values between internal and external measures. The large amount of deep drops on the ratio shows a poor approximation to the ground truth.

6.4 Discussion and Future Work

The results extracted from the experiments indicate a that SAMOA Clustering algorithm is prone for scaling up. Although these are preliminary results they show the potential that SAMOA has to be used on a large scale distributed environment. The experiments have rendered good results but there can be improvements on the setup. The machines used were not dedicated machines and shared processing with other users, thus to have a more accurate performance evaluation dedicated machines must be used. Another aspect is the amount of resources used to scale, which was not a large setup. Future experiments should test the scaling limits of the system, testing it on hundreds of nodes.

The use of cohesion and separation measures turned out to be a good clustering quality assessment, where on clean datasets the algorithm was very close to the ground truth. On "dirty" datasets it tracked the poor clustering quality, which can be used to develop and improve techniques

for dealing with outliers. Other experiments should be done to evaluate on even noisier datasets. An additional experiment is to check the influence of the parallelism level in the final clustering quality. Future works in the evaluation would include other measures to evaluate the clustering algorithms for different conditions.

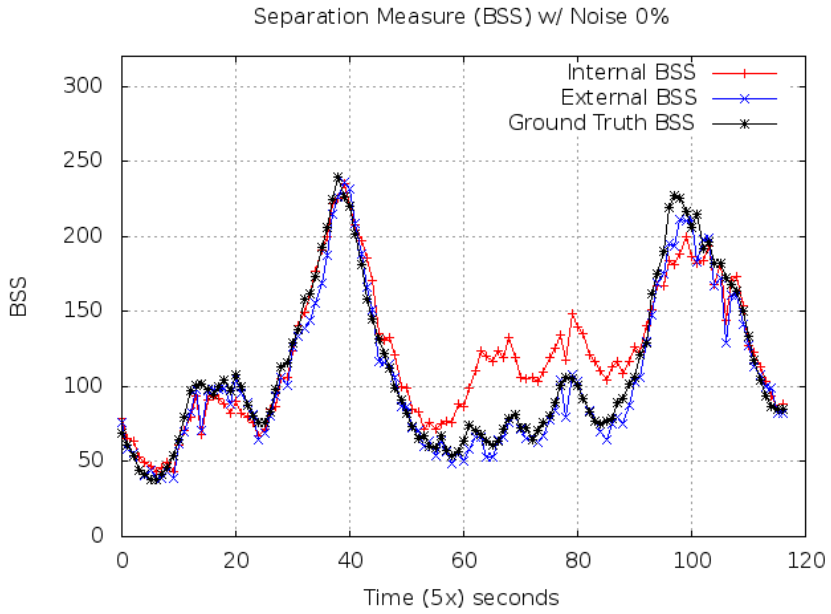

Figure 6.4: BSS separation value over time for clustering generation with noise 0%.

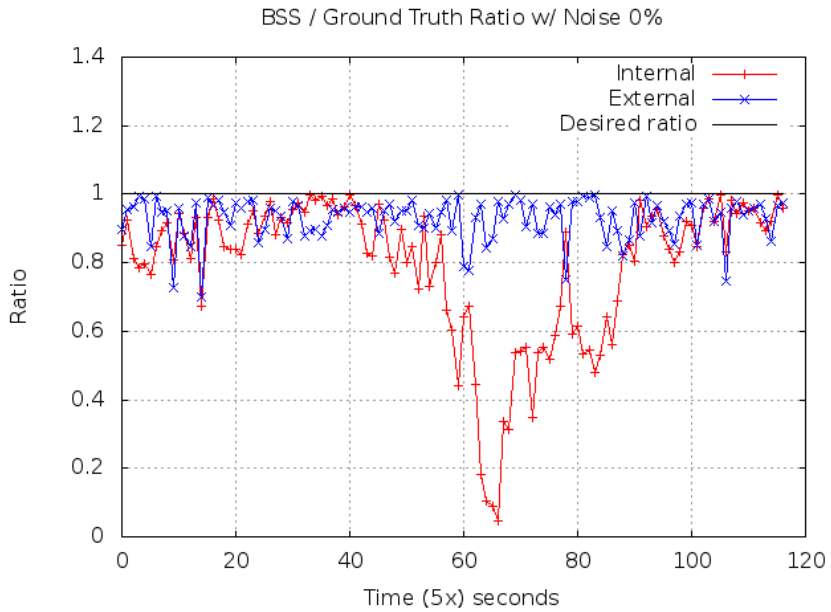

Figure 6.5: Ratio of external and internal separation with the ground truth BSS on a dataset with 0% noise.

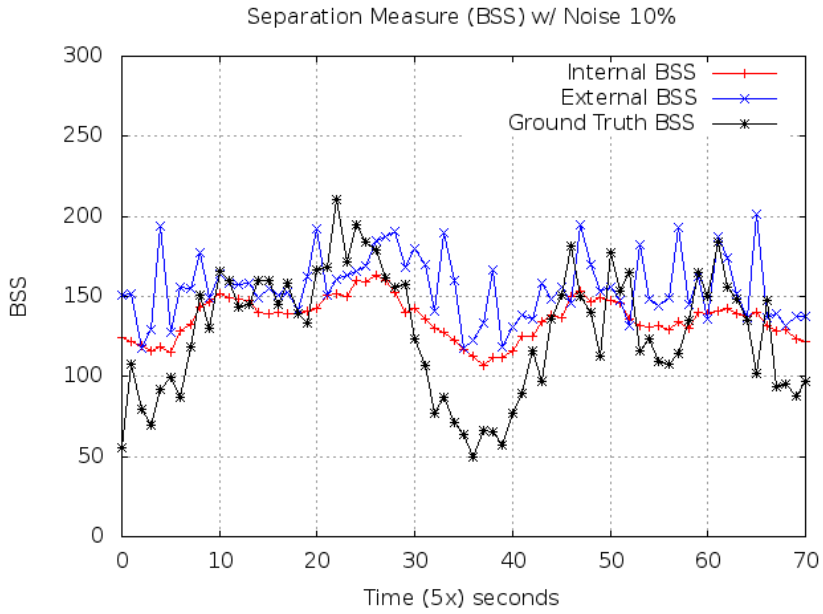

Figure 6.6: BSS separation value over time for clustering generation with noise 10%.

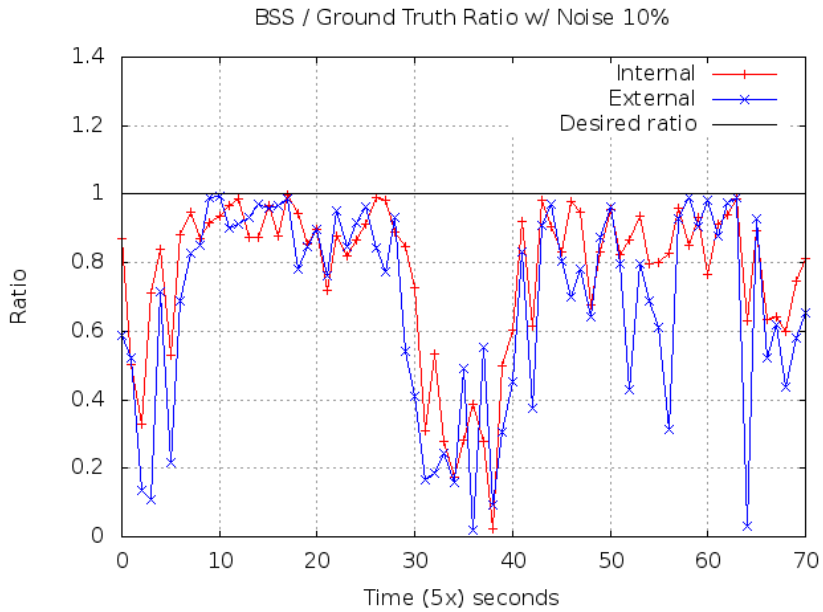

Figure 6.7: Ratio of external and internal separation with the ground truth separation on a dataset with 10% noise.

7 Conclusions

7.1 Conclusions

The SAMOA project is a distributed streaming platform-independent machine learning framework used to develop and test algorithms in large scale infrastructures. SAMOA will be released as an open-source project for the community to extend, experiment and use it to its fullest capacity. In this project we have developed the integration of SAMOA to the Apache S4 distributed streaming platform using the SAMOAs adapter layer and tested it by implementing a distributed clustering algorithm based on Aggarwal CluStream algorithm. The SAMOA Clustering was developed using the SAMOA API for creating data flow topologies, which can be deployed on any distributed SPE that can be plugged to SAMOA. Based on the SAMOA API and the concepts of the actors model, the algorithms have to be designed to work asynchronously and have its logic implemented using processing item components. By connecting the processing items to streams a data flow topology can be built to tackle the most diverse problems in distributed stream machine learning. This flexibility has an advantage over the MapReduce programming model, where problems and algorithms can be designed with any kind processing elements configuration besides mappers and reducers. By conducting experiments we have tested SAMOAs scalability and SAMOAs clustering quality. The results showed that the algorithms perform well in a distributed environment and can scale-up to support higher loads. The quality of the clustering has been assessed by the amount of intra cluster cohesion and on how far is the inter cluster separation. These measures intend to show that highly cohesive clusters are better than clusters with low cohesion and the further away the clusters are the greater the distinction between the partitions. SAMOA is a project under development and need further development and testing, but already shows its potential as distributed stream machine learning

framework

Bibliography

[1] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: wait-free coordination for internet-scale systems," in *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, USENIXATC'10, (Berkeley, CA, USA), pp. 11–11, USENIX Association, 2010.

[2] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, pp. 107–113, Jan. 2008.

[3] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and T Technologies (MSST)*, MSST '10, (Washington, DC, USA), pp. 1–10, IEEE Computer Society, 2010.

[4] K. Tretyakov, "Machine learning techniques in spam filtering," tech. rep., Institute of Computer Science, University of Tartu, 2004.

[5] V. Estivill-Castro, "Why so many clustering algorithms: a position paper," *SIGKDD Explor. Newsl.*, vol. 4, pp. 65–75, June 2002.

[6] S. Lloyd, "Least squares quantization in pcm," *Information Theory, IEEE Transactions on*, vol. 28, no. 2, pp. 129–137, 1982.

[7] E. Forgy, "Cluster analysis of multivariate data: Efficiency versus interpretability of classification," *Biometrics*, vol. 21, no. 3, pp. 768–769, 1965.

[8] H. P. Friedman and J. Rubin, "On Some Invariant Criteria for Grouping Data," *Journal of The American Statistical Association*, vol. 62, pp. 1159–1178, 1967.

[9] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. Fifth Berkeley Symp. on Math. Statist. and Prob.*, vol. 1, pp. 281–297, Univ. of Calif. Press, 1967.

[10] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "A local search approximation algorithm for k-means clustering," in *Proceedings of the eighteenth annual symposium on Computational geometry*, SCG '02, (New York, NY, USA), pp. 10–18, ACM, 2002.

[11] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1027–1035, 2007.

[12] M. Mahajan, P. Nimbhorkar, and K. Varadarajan, "The planar k-means problem is np-hard," in *WALCOM: Algorithms and Computation* (S. Das and R. Uehara, eds.), vol. 5431 of *Lecture Notes in Computer Science*, pp. 274–285, Springer Berlin Heidelberg, 2009.

[13] M. Inaba, N. Katoh, and H. Imai, "Applications of weighted voronoi diagrams and randomization to variance-based k-clustering: (extended abstract)," in *Proceedings of the tenth annual symposium on Computational geometry*, SCG '94, (New York, NY, USA), pp. 332–339, ACM, 1994.

[14] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: a new model and architecture for data stream management," 2003.

[15] T. S. Group, "Stream: The stanford stream data manager," 2003.

[16] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah, "Telegraphcq: Continuous dataflow processing for an uncertan world," 2003.

[17] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik, "The Design of the Borealis Stream Processing Engine," in *Second Biennial Conference on Innovative Data Systems Research (CIDR 2005)*, (Asilomar, CA), January 2005.

[18] G. Agha, *Actors: a model of concurrent computation in distributed systems*. Cambridge, MA, USA: MIT Press, 1986.

[19] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Mining data streams: a review," *SIGMOD Rec.*, vol. 34, pp. 18–26, June 2005.

[20] S. Bandyopadhyay, C. Giannella, U. Maulik, H. Kargupta, K. Liu, and S. Datta, "Clustering distributed data streams in peer-to-peer environments," *Information Sciences*, vol. 176, no. 14, pp. 1952 – 1985, 2006. ¡ce:title¿Streaming Data Mining¡/ce:title¿.

[21] A. Bifet, G. Holmes, B. Pfahringer, J. Read, P. Kranen, H. Kremer, T. Jansen, and T. Seidl, "Moa: A real-time analytics open source framework," in *Machine Learning and Knowledge Discovery in Databases* (D. Gunopulos, T. Hofmann, D. Malerba, and M. Vazirgiannis, eds.), vol. 6913 of *Lecture Notes in Computer Science*, pp. 617–620, Springer Berlin Heidelberg, 2011.

[22] G. Hulten and P. Domingos, "VFML – a toolkit for mining high-speed time-changing data streams," 2003.

[23] M. Cannataro, A. Congiusta, A. Pugliese, D. Talia, and P. Trunfio, "Distributed data mining on grids: services, tools, and applications," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 34, no. 6, pp. 2451–2465, 2004.

[24] N. Giannadakis, A. Rowe, M. Ghanem, and Y.-k. Guo, "Infogrid: providing information integration for knowledge discovery," *Inf. Sci. Inf. Comput. Sci.*, vol. 155, pp. 199–226, Oct. 2003.

[25] I. Dhillon and D. Modha, "A data-clustering algorithm on distributed memory multiprocessors," *Large-Scale Parallel Data Mining*, pp. 802–802, 2000.

[26] "Jubatus: Distributed online machine learning framework," 05 2013.

[27] L. Amini, H. Andrade, R. Bhagwan, F. Eskesen, R. King, Y. Park, and C. Venkatramani, "Spc: A distributed, scalable platform for data mining," in *In Proceedings of the Workshop on Data Mining Standards, Services and Platforms, DM-SSP*, 2006.

[28] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform," in *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*, pp. 170–177, 2010.

[29] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for clustering evolving data streams," in *Proceedings of the 29th international conference on Very large data bases - Volume 29*, VLDB '03, pp. 81–92, VLDB Endowment, 2003.

[30] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: An efficient data clustering method for very large databases," 1996.

[31] A. K. Jain and R. C. Dubes, *Algorithms for clustering data*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988.

[32] J. Wu, H. Xiong, and J. Chen, "Adapting the right measures for k-means clustering," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 877–886, 2009.