Clustering Evolving Data Streams With Apache Cassandra

Jose Fernandez
jrf15@students.uwf.edu

Department of Computer Science, University of West Florida
11000 University Parkway, Bldg. 4
Pensacola, FL 32514
(850) 474-2542

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Jose Fernandez	May 31, 2011	Initial Release	0.1

1 Abstract

Today, there are many organizations that generate data at astonishing rates. Consider the following numbers taken from an IDC research white paper entitled, "The Expanding Digital Universe".

- *Wal-Mart's database of customer transactions is reported to have stored 110 terabytes in 2000, recording tens of millions of transactions per day. By 2004, it had grown to half a petabyte.*
- *In 2006, the amount of data on the Internet was approximately 166 exabytes (166EB) and by 2010, that number had reached nearly 1,000 exabytes. An exabyte is one quintillion bytes, or 1.1 million terabytes.*

It is safe to assume that combined, today's organizations produce millions of data records per minute if not per second of each day. Applying machine learning algorithms to these vast streams of data in order to gain new knowledge from the data also presents new challenges. This paper describes a data stream clustering system whose design addresses these new challenges and at the same time, allows end-users to explore and gain knowledge from the evolving data stream.

2 Introduction

This paper proposes and defines a project whose goal is the adaptation and integration of an existing data stream clustering algorithm (CluStream) to a new breed of open source, distributed and horizontally scalable database system. These new database systems, which are referred to as NoSQL or non-relational database systems, are designed to tackle massive data sets and to perform in near-time. The combination of an efficient clustering algorithm like CluStream, or derivative of, and this new breed of NoSQL database results in a clustering system or framework that is well-equipped to operate within the severe time and space constraints associated with real-time evolving data streams. Through the use of such a system, end-users can gain a much deeper understanding of the data stream and its evolving nature in near-time and over different time horizons. For example, an end-user can explore the data stream's clusters over the last 5 minutes, 24 hours or year. One possible application is for the identification of patterns that reflect network intrusion or cyber attacks.

This project's initial work products may also serve as the basis for an ongoing data stream management system (DSMS) project at the University of West Florida and/or whose work products can be contributed to an open source machine learning community such as Apache Mahout (http://mahout.apache.org). The project incorporates disciplines such as data mining, distributed systems, machine learning, statistics and various areas of mathematics (e.g. linear algebra). As such, it can serve as a learning framework whose extension is realized by subsequent cross-disciplinary group projects.

The NoSQL database that will be the target of this project is Apache Cassandra (http://cassandra.apache.org/). Another example of a NoSQL database is the Apache Hadoop Database or HBase, as it is more commonly referred. Along with being horizontally scalable and fault-tolerant, Cassandra and HBase are open source, distributed, and decentralized systems whose design is based on Amazon's Dynamo and Google's Bigtable column-oriented data models. The reason this project has chosen Cassandra over HBase is due to Cassandra's symmetric architecture, ease of use and the fact that it is a standalone system (i.e., has no dependency on other open source products or components). Thus, the clustering system or framework that this project is to design and implement will be referred to as **CluSandra**.

One of the goals of this project is to design and develop CluSandra as an extendable data stream clustering framework that can be relatively easily extended to host a number of different clustering methods similar to CluStream. Another goal of the project is to extend or adapt the CluStream algorithm for the CluSandra framework. The adaptation process also introduces new feature-functionality.

Similar systems have been designed using commercial data warehouses to perform queries on streaming data. These systems are referred to as streaming or active warehouses. However, these systems are very expensive and based on relational database systems, which are much more difficult to distribute and scale across multiple nodes. For example, attempts to use MySQL for such environments have faltered due to the complicated nature of MySQL's sharding.

3 Data Streams

Unlike a traditional database or data set, a data stream is a series of structured data records (a.k.a., data objects, tuples or n-dimensional vectors) that exhibit these characteristics: very fast arrival rate, temporally ordered, fast evolving, and unbounded. According to Golab & Oszu (2003), *"a data stream is a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of items. It is neither possible to control the order in which items arrive, nor is it feasible to locally store a stream in its entirety" (page 2)*.

In this paper, a stream S is viewed as an unbounded sequence of pairs $\langle s,t \rangle$, where s is a structured data record and $t \in T$ is the timestamp that specifies the arrival time of data record s on stream S. Since data streams comprise structured records, unstructured streams (e.g., audio and video streams) are not considered data streams within the context of this paper.

Example data streams include IP network traffic, web click streams, mobile calls, ATM transactions, credit card transactions, and sensor network traffic. Therefore, a data stream produces a vast and endless amount of data that is assumed to continuously evolve over time, and at times can arrive at exceedingly fast rates. Due to these characteristics, machine learning algorithms (e.g., clustering and classification) that are used to mine or learn from data streams have been designed and implemented within the context of the following time-space related constraints (Bifet & Kirkby, 2009):

1. Data stream records are processed one at a time and processed only once. With a data stream, there is no concept of randomly accessing the data records and the algorithms do not have the capability or luxury of making multiple passes at the data. Storing the data in its entirety on disk or a database management system is not a feasible option, because it is assumed that the amount of data produced is unbounded.
2. The data stream produces an amount of data that far-exceeds available working memory; therefore, the data mining algorithms must be designed to make very efficient use of working memory. This is often accomplished via the use of *synopsis* data structures, which summarize the data and preclude the need to store all data records. As you will see later in this document, the main synopses data structure used by the CluStream algorithm is the microcluster or cluster feature (CF). The microcluster not only summarizes all of the data records that comprise the cluster, but also is a form of data compression.
3. The arrival rate of the data stream's data records can be exceedingly high; therefore, the algorithms must be designed to process the records just as fast or preferably faster than their arrival rate.
4. Early *batch* machine learning algorithms assume that records in the data set are generated at random according to a stationary probability distribution. However, with data streams, the distribution of the data evolves over time; this is referred to as 'concept drift'. The machine learning algorithms are expected to adapt to the concept drift or facilitate its identification.
5. The algorithms are expected to produce results 'anytime'; this is often referred to as an on-demand algorithm.

The following are some examples of applications for data stream processing (Hebrail, 2008).

- The real-time monitoring of information systems that generate vast amounts of data. For example, computer network management, telecommunications call analysis, internet applications (e.g., Google, Ebay, recommendation systems, click stream analysis) and monitoring of power plants.

- Generic software for applications based on streaming data. For example, finance (fraud detection, stock market analysis), sensor networks (e.g., environment, road traffic, weather forecasting, electric power consumption).

4 Cluster Analysis

An intelligent being cannot treat every object it sees as a unique entity unlike anything else in the universe. It has to put objects in categories so that it may apply its hard-won knowledge about similar objects encountered in the past, to the object at hand.

Steven Pinker, *How the Mind Works*, 1997.

Clustering is the process by which one partitions objects into groups based on similarity. That is, all objects in a particular group are similar to one another, while objects in different groups are quite dissimilar. Clustering is an inherent trait in human beings. As humans, we all instinctively learn how to group objects. For example, early in childhood, we learn how to distinguish between horses, cows, fish and turtles by continuously improving subconscious clustering schemes (Han & Kamber, 2006). Mankind developed language in part by giving these groups names (i.e., nouns), which is a form labeling.

With respect to artificial intelligence and machine learning, cluster analysis can be regarded as "unsupervised pattern recognition". It automates the grouping process and allows the end-user to discover and explore distribution patterns within the data.

Cluster analysis lies at the intersection of many disciplines such as statistics, machine learning, data mining, and linear algebra (Han & Kamber, 2006). It is also used for many applications such as pattern recognition, fraud detection, market research, image processing, network analysis, psychology and biology.

Cluster analysis, which from hence forth is simply referred to simply as clustering, also allows end-users to identify outliers (i.e., data records or clusters that are extremely dissimilar to other records or clusters). For some applications, outliers are of strong interest. Take for example network intrusion detection, credit card fraud, and other types of electronic criminal activity, which are exceptions to the norm.

4.1 Supervised vs. Unsupervised Learning

In most applications, clustering seeks to partition *unlabeled* data records from a large data set into clusters. The process of learning or gaining knowledge from an unlabeled data set is referred to as *unsupervised* learning.

Clustering is somewhat like *classification*, which is another basic type of machine learning algorithm that is also used to partition data records. However, unlike clustering, classification algorithms must be given a set of *labeled* data records, which are taken from the original set, to induce a learning model or function. The act of inducing a learning model from a labeled set of records is referred to as *supervised* learning and the labeled set is called the *training set*. The model, which is simply expressed as $y = f(x)$, predicts the class y of future data records x with high accuracy (Domingos & Hulten, 2000). The resulting model or learner (e.g., decision tree) is then applied to the rest of the data set to partition the unlabeled records into their respective classes. In general, classification algorithms are effective for classifying large static data sets. However, within an evolving data streaming environment, creating a training set is a very expensive undertaking. Many papers have been presented that describe supervised classification algorithms for evolving data streams; however, the respective algorithms' feasibility or practically is in question. For example, how does one create a training set, in real-time, to ensure the learning model adapts, also in real-time, to the evolving data stream? The papers never seem to address this issue. Recently, a handful of papers have emerged that raise this same question and explore the concept of combining clustering with

limited amounts of labeled data to provide a practical means to explore evolving data streams (Masud, Gao, Khan, Han, 2008; Zhang, Zhu, Guo, 2009).

Unlike the supervised classification algorithms, clustering algorithms do not rely on labeled training sets to partition data records into their respective clusters. It is said that clustering is a form of learning by observation instead of learning by example. Clustering takes a somewhat opposite approach from that of supervised learners, because it first processes unlabeled data records to group them into clusters; the clusters can then be assigned labels. Clustering presents an attractive advantage in that it is more easily adapted to changes in the data and can thus be used to identify features that distinguish different clusters (Han & Kamber, 2006). This is ideal for concept drifting data streams.

4.2 Data Types and Distance Measures

Depending on the application, clustering may need to take into account different data types and their preprocessing. The following are some examples of the different data types: numerical or interval scaled, binary, categorical, ordinal, and ratio-scaled. In some cases, clustering must preprocess a mixture of all these types. The CluStream algorithm has been designed to only process numerical data; therefore, this project will only focus on that data type. Future iterations of this system's design can be extended to handle the other data types. This project will also assume that the values for all the data records' attributes or variables are *standardized*; therefore, there is no preprocessing of the data records. In other words, the records' attributes will be presented in the same units of measure. For example, all attributes that represent a height will be presented in meters and will not be a mixture of meters and yards or yards and feet.

Numerical data types are continuous measurements of a roughly linear scale (Han et. al. 2006). Examples of continuous data or variables are crime rates, height, weight, and temperature. When working with data records comprising continuous attributes, the similarity or dissimilarity between individual records is typically quantified by some form of distance measure. There exists a variety of distance measures that are applied to continuous data. The following are some examples: Euclidean distance, city block or Manhattan distance, Minkowski distance, Caneberra distance, Pearson correlation, and Angular separation. Most if not all of these distance measures allow for some form of differential weighting of the variables. For example, a temporal weight may be assigned to the measures that place less importance on the measures as time progresses (e.g., time decay).

The most common distance measurement used for continuous nermical data is the Euclidean measure

$$\mathrm{d(i,j)} = \sqrt{\sum_{k=1}^{n} (x_{ik} - x_{jk})^2}$$

where x_{ik} and x_{jk} are the k^{th} variable for the n-dimensional data records i and j. For example, suppose you have two 2-dimenisonal data records as follows: (1,3) and (4,1). The Euclidean distance between these two records would be the following:

$$\sqrt{(1-4)^2 + (3-1)^2} = 3.60$$

If we view the two records as Euclidean vectors in Euclidean n-space (Figure 1.a), the distance between the two vectors is the length of the line connecting the two vectors' tips (points). The lower the resulting value, the closer (similar) are the two data records. Euclidean distance is an integral part of this project.

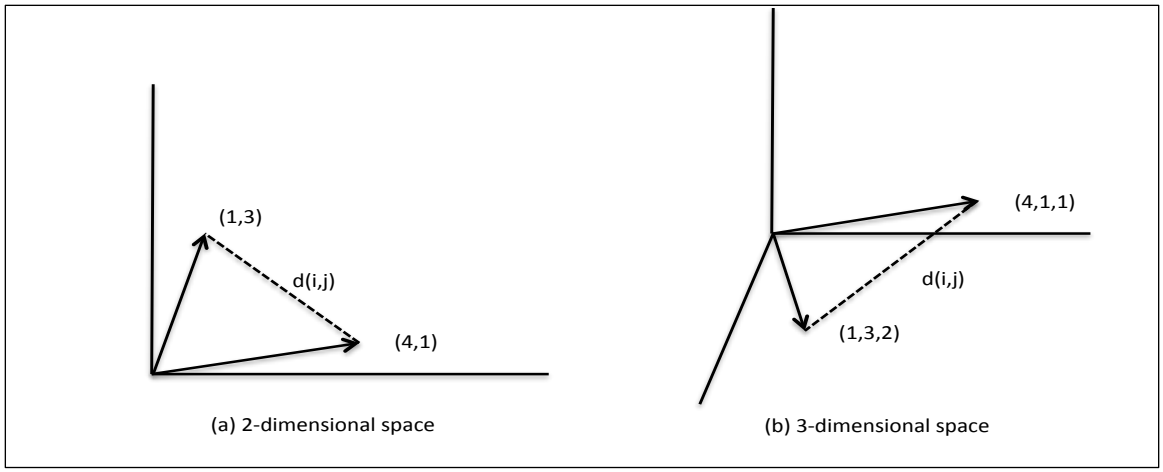

Figure 1. 2 and 3 dimensional Euclidean space

Figure 1 depicts two vectors in 2 and 3-dimensional Euclidean spaces. The Euclidean distance formula can be applied to any n-dimensional vector. A data type that satisfies the requirements of Euclidean space and triangular inequality is also referred to as a *metric* variable or attribute. Triangular inequality states that for any three points in n-dimensional space, i, j, and k, $d(i,j) + d(j,k) \geq d(i,k)$.

For this project, the Euclidean distance is used as the distance measure to determine how similar or close a new data record is to an existing cluster's centroid (mean). It is also used to find the distance between clusters. Given a new data record and set of clusters, the Euclidean distance is used to find the cluster that is closest to the new data record. A threshold parameter is provided that specifies the maximum boundary around the cluster (i.e., its radius). If the new data record's distance to its closest cluster's centroid is within the specified boundary, the new data record is absorbed by the cluster. This form of *partitioning* scheme is similar to the *k-means* method, which assigns objects to their respective clusters based on the object's distance to the cluster's mean. When a new object is assigned or absorbed by a cluster, the cluster's mean is recalculated. More on this in the next section.

In this paper, the following terms are used interchangeably: data record, tuple, vector, and data point.

5 BIRCH

CluStream is a clustering algorithm that is specifically designed for evolving data streams and is derived from the BIRCH (Balanced Iterative Reducing and Clustering) algorithm (Zhang, Ramakrishnan & Livny, 1996). This section provides an overview of BIRCH followed by a section that describes how CluStream extends BIRCH to introduce a temporal dimension to the algorithm. ***This project is heavily based on some of the concepts and structures presented by these two algorithms; therefore, these two algorithms are described at some length.***

There are many types of clusters that are represented by n-dimensional vector data. BIRCH, and thus CluStream, are based on the K-means (center-based) clustering paradigm; therefore, this project targets perhaps the simplest type of cluster, which is referred to as the *spherical Gaussian* cluster. Clusters that manifest non-spherical or arbitrary shapes, such as *correlation* and *non-linear correlation* clusters, are not addressed in this project.

The BIRCH incremental clustering algorithm was designed to mitigate the I/O costs associated with clustering very large multi-dimensional data sets. In general, BIRCH is a batch algorithm that includes and relies on multiple sequential phases of operation and is therefore not well-suited for data stream environments where time is severely constrained. However, it does introduce concepts and a synopsis structure that can be applied to the clustering of data streams. For example, the algorithm can typically find

a good clustering with a single scan of the data (Zhang, et. al., 1996), which is an absolute requirement when having to process data streams. BIRCH also introduces two structures: *cluster feature* (CF) and *cluster feature tree* (CF tree). This project is heavily dependent on the CF; however, it does not make use of the CF tree. An overview of the CF tree will nonetheless be presented to give the reader an appreciation for the decision not to use the CF tree structure. As you'll read, it is the amount of time involved in the maintenance of this structure that, in part, precludes it from being used in a data stream environment.

The CF and CF tree structures are at the core of the BIRCH algorithm, but before discussing these two structures, we'll discuss these three spatial terms: *centroid*, *radius*, and *diameter*. All three are related to vector spaces and are an integral part of the BIRCH algorithm.

Given N n-dimensional data objects (data records, vectors or points) in a cluster, where i = {1,2,3, …, N}, the centroid $\vec{x}_0$, radius R, and diameter D of the cluster are defined as follows:

$$\vec{x}_0 = \frac{\sum_{i=1}^{N} \vec{x}_i}{N}$$

$$R = \sqrt{\frac{\sum_{i=1}^{N} \left(\vec{x}_i - \vec{x}_0\right)^2}{N}}$$

$$D = \sqrt{\frac{\sum_{i=1}^{N}\sum_{j=1}^{N} \left(\vec{x}_i - \vec{x}_j\right)^2}{N\left(N-1\right)}}$$

The centroid is the cluster's mean or center of gravity, the radius is the average distance from the objects within the cluster to their centroid, and the diameter is the average pair-wise distance within the cluster. The radius and diameter measure the tightness of the objects around their cluster's centroid. The radius can also be viewed as the "root mean squared deviation" or RMSD with respect to the centroid. Note that the centroid itself is a data record or vector.

Typically, the first criteria for assigning a new object to a particular cluster is the new objects Euclidean distance (similarity, dissimilarity) to a cluster's centroid. That is, the new object is pegged to it closest cluster.

To calculate the distance between two clusters, you can also apply the Euclidean distance formula to the two clusters' centroids.

The one disadvantage to relying on these spherically related spatial concepts is that you cannot apply the algorithm to clusters that are not by nature spherically shaped; i.e., do not follow a Gaussian distribution.

5.1 The Cluster Feature (CF)

A CF is a 3-tuple or triplet synopsis structure whose elements summarize the statistical information required to calculate the centroid, radius, and diameter of a cluster. In other words, the CF captures summary information for a particular pattern in the data set. The CF vector is formally defined as

$$CF = \langle N, LS, SS \rangle$$

where N is the total number of objects in the cluster, LS (vector) is the linear sum of the objects in the cluster and SS is the squared sum of the objects in the cluster.

$$LS = \sum_{i=1}^{N} \vec{x}_i$$

$$SS = \sum_{i=1}^{N} \vec{x}_i^2$$

For example, suppose you have three 2-dimenisonal data records (2,5), (3,2) and (4,3) in a cluster C_1. The CF of C_1 is $\langle 3,(2+3+4,5+2+3),(4+9+16,25+4+9)\rangle = \langle 3,(9,10),(29,38)\rangle$. From the summary information held by the CF, you can calculate the centroid, radius, and diameter of the cluster and thus there is no need to store all the cluster's objects. What is also interesting about CFs is that they are *additive* and *subtractive*. For example, if you have two clusters, C_1 and C_2 with their respective cluster features, CF_1 and CF_2, the CF that is formed by merging the two clusters is simply $CF_1 + CF_2$.

5.2 The Cluster Feature Tree

The cluster feature tree (CF tree) is an in-memory, height-balanced tree (similar to a b-tree) that serves as an indexing structure for CFs. It is essentially used for the efficient (provides linear scalability) maintenance of all the CFs that have been created. The tree has two parameters: branching factor B and threshold T. Each non-leaf node contains at most B entries of the form $[CF_i, child_i]$, where i = 1, 2, …, B, $child_i$ is a pointer to its i-th child node, and CF_i is the CF of the subcluster represented by this child. A non-leaf node represents a cluster comprising all the subclusters represented by its entries (remember that CFs are additive). A leaf node contains at most L entries, each of the form $[CF_i]$, where i = 1, 2, …, L. A leaf node also represents a subcluster that is represented by all of its CF entries. The leaf nodes are all interconnected as a doubly linked list. All the entries in the leaf node must satisfy the threshold requirement T (i.e., the diameter or radius of CF must be less than T). The tree size is a function of T and the larger T is, the smaller the tree. In other words, each CF is capable of absorbing more data records. More on this in the next section (5.1.3). The CF tree is thus a very compact representation of the data set, because each of the tree's nodes are collections of CFs and not individual data records.

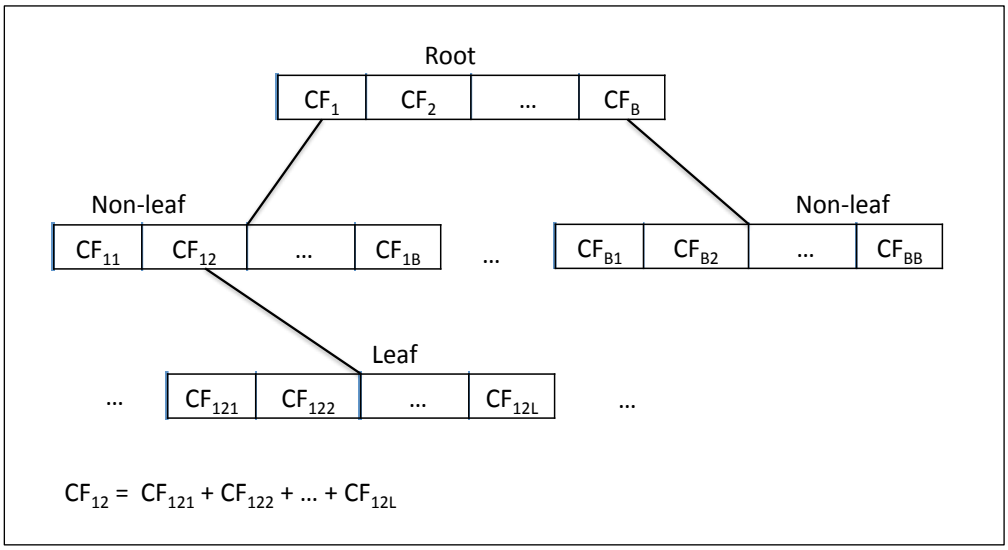

Figure 2. CF tree

The paper that presents the BIRCH method (Zhang, et. al., 1996) does not address persisting the CF tree to secondary storage. Although it does describe how the tree's parameters are derived from a page size. Perhaps it is assumed that the data set will always be available and thus the CF tree can be rebuilt from the data set. This is a luxury that is not available when attempting to cluster data streams.

For this project, there is no requirement to maintain an in-memory CF tree, because all CF-like structures will be persisted to the Cassandra database management system; however, the project will need to design a Cassandra based secondary index (inverted index). This index, which will be based on a Cassandra supercolumn family, will provide efficient access to the CF structures. More information regarding this Cassandra secondary index, and why it is required, will be provided in the project's design specification.

5.3 Phases of Operation

BIRCH is broken down into four phases of operation whose focus is on the maintenance of the CF tree. The fourth phase requires a re-scanning of all the data records to improve the overall quality of the clustering; therefore, it cannot be applied to data streams and is not described in this paper. The following provide a quick overview of the first three phases. More attention is paid to the more important first phase.

Phase 1

This phase starts with an initial threshold value (radius or diameter), scans the data set and inserts objects into the CF tree. The insertion process starts at the root of the tree and recursively descends the tree by choosing the child node that is closest to the new data record being inserted. The Euclidean distance formula can be used for measuring closeness. When the closest leaf entry (CF) is eventually found, it is then determined if it can absorb the new data record. In other words, is the new data record within the specified threshold value? If so, the CF is updated (i.e., absorbs the new record), else a new CF entry is created to absorb the new record. If L is exceeded by having to create a new CF entry for the leaf, then the leaf node is split. When a CF absorbs a new record, the CF information must be updated for all non-leaf nodes that are in the path leading to the absorbing CF. A splitting of a leaf node may also cause non-leaf nodes in the path to also require splitting.

If the algorithm exhausts its allocated memory before scanning all the data, it automatically increases the threshold value (i.e., increases the minimum radius or diameter) and rebuilds a new and smaller CF tree. The rebuilding process uses the entries in the leaf nodes and thus does not have to rescan the data set. When rebuilding the tree, the algorithm also looks for low-density leaf entries. That is, leaf entries whose total number of data records (N) is far fewer than the overall average. These leaf entries are considered "*outliers*" and thus not included in the rebuilding process. The outliers are persisted to disk. In general, rebuilding the CF tree is expensive and would not be practical within the context of a high-speed data stream.

Since each node in the tree has a maximum number of entries, it does not always correspond to a natural cluster. For example, two subclusters that should have been in one cluster end up being split across nodes (Zhang, et. al., 1996). The intent of phase 3 is to address this by applying "global clustering" methods to the CF tree. Another undesirable side-affect is that if two identical data records are inserted at different times, they may very well end up in different leaf nodes. These types of anomalies are addressed by the phase 4 (re-scanning of data) operation.

Phase 2

Phase 2 is an optional phase that is used to normalize the CF tree for phase 3. Phase 2 is similar to phase 1's CF tree rebuilding process in that it's main purpose is to produce a smaller tree and divorce it of any additional outliers. This phase is executed only if phase 3 requires it.

Phase 3

As previously mentioned, phase 3 includes the application of global clustering methods or algorithms to the CF tree in order to address the unwanted side effect of subclusters being split across multiple nodes. The BIRCH paper (Zhang, et. al., 1996) describes several methods for adapting global clustering algorithms, which work on sets of data points, to work on CFs. One method is to treat the CFs centroid as a data point.

6 CluStream

As previously mentioned, CluStream is a clustering algorithm that is specifically designed for evolving data streams and borrows from the BIRCH algorithm. From an analysis perspective, the general intent of CluStream is to address the one-pass constraint imposed by data streams on the design of existing data stream clustering algorithms. For example, the results of applying a one-pass clustering algorithm, like BIRCH, to a data stream lasting 1 or 2 years would be dominated by outdated data. CluStream allows end-users to explore the data stream over different time windows or horizons and thus get an understanding of how and when the data stream has evolved over time.

CluStream tackles the one-pass constraint by dividing the clustering process into two phases of operation that are meant to operate simultaneously, if desired. The first, which is referred to as the *online* phase, efficiently computes and stores summary statistics about the data stream in *microclusters*. A microcluster is an extension of the CF whereby the CF is given a temporal dimension. The second phase, which is referred to as the *offline* phase, allows end-users to perform *macroclustering* operations on the microclusters.

Macroclustering is the process by which end-users can explore the microclusters over different time horizons. To accomplish this, CluStream uses a *tilted time frame* model for maintaining the microclusters. The tilted time frame approach stores sets of microclusters (snapshots) at different levels of granularity based on elapsed time. In other words, as time passes, the microclusters are merged into coarser snapshots.

6.1 Microcluster

As previously mentioned in section 3, a stream S is viewed as an unbounded sequence of pairs $\langle s,t \rangle$, where s is a structured data record and $t \in T$ is the timestamp that specifies the arrival time of data record s on stream S. The CluStream microcluster extends the CF by adding two temporal scalars to the CF. The first scalar is the sum of the timestamps of all the data records whose summary information is contained in the CF and the second is the sum of the squares of the timestamps. Thus the CF triplet is extended as follows,

$$CF = \langle N, LS, SS, ST, SST \rangle$$

where ST is the sum of the timestamps and SST is the squared sum of the timestamps. Note that this new extended CF retains its additive and subtractive properties. From henceforth, this new structure is referred to as a CFT or microcluster.

6.2 Online Phase

CluStream's online microclustering phase collects and maintains the statistical information in such a manner that the offline macroclustering phase can make effective use of the information. For example, macroclustering over different time horizons and exploring the evolution of the data stream over these horizons.

The algorithm defines a fixed set of q microclusters that it creates and maintains in-memory. This set, $\mathcal{M} = \{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_q\}$ is referred to as the current online snapshot with q being the maximum number of microclusters in the snapshot, with each microcluster in $\mathcal{M}$ being given a unique *id*. The value for q is based on the available amount of memory; however, there is no mention of how q is calculated. However, it is stated that that the algorithm is very sensitive to any further additions to the microcluster data structure (i.e., the CFT), as this negatively impacts q. This type of space constraint is one that CluSandra lifts by relying on Cassandra to serve as a highly reliable and scalable real-time distributed data store.

The updating of the microclusters is similar to that of the BIRCH method. When a new data record is presented by the data stream, it is assigned to the closest microcluster $\mathcal{M}_i$ in $\mathcal{M}$. Most of the algorithm's time is spent finding the closest microcluster in $\mathcal{M}$. After finding the closest microcluster $\mathcal{M}_i$, it is then determined if $\mathcal{M}_i$ can absorb the new record based on a maximum boundary threshold. CluStream defines this threshold as, "a factor of t of the RMSD of the data points in $\mathcal{M}_i$ from the centroid", where RMSD is the root mean squared deviation. This is just another way of referring to the radius of the cluster times sum factor t. For example, if t = 1.6 and the radius is 2, then the maximum boundary threshold is 3.2. The microcluster $\mathcal{M}_i$ is updated whenever it absorbs a new data record. If $\mathcal{M}_i$ has only one entry, the RMSD cannot be calculated; therefore, for those clusters having only one entry, the maximum boundary is derived as the distance from $\mathcal{M}_i$ to its closest neighbor times a factor r.

If $\mathcal{M}_i$ cannot absorb the new data record, a new microcluster must be created to host the data record. To conserve memory, this requires that either one of the existing microclusters in $\mathcal{M}$ be deleted or two microclusters be merged. Only those microclusters that are determined to be *outliers* are removed from $\mathcal{M}$. If an outlier cannot be found in $\mathcal{M}$, two microclusters are merged.

CluStream uses the temporal scalars (ST, SST) of the microcluster, in combination with a user-specified threshold δ, to look for an outlier microcluster in $\mathcal{M}$. The ST and SST scalars allows CluStream to calculate the mean and standard deviation of the arrival times of the data records in $\mathcal{M}$'s microclusters. CluStream assumes the arrival times adhere to a normal distribution. With the mean and standard deviation of the arrival times calculated, CluStream calculates a "*relevance stamp*" for each of the microclusters. A microcluster whose relevance stamp is less than the threshold δ is considered an outlier and subject to removal from $\mathcal{M}$. If all the relevance stamps are recent enough, then it is most likely that there will be no microclusters in $\mathcal{M}$ whose relevance stamp is less than δ. If and when this occurs, CluStream merges the two closest microclusters in $\mathcal{M}$ and assigns the resulting merged microcluster a *listid* that is used to identify the clusters that were merged to create this new merged microcluster. So as time progresses, one microcluster may end up comprising many individual microclusters.

Unlike BIRCH, CluStream does not utilize a tree structure to maintain its microclusters. At certain time intervals, and while $\mathcal{M}$ is being maintained as described above, $\mathcal{M}$ is persisted to secondary storage. Each instance of $\mathcal{M}$ that is persisted is referred to as a **snapshot**. CluStream employs a logarithmic based time interval scheme, which it refers to as a **pyramidal time frame,** to store the snapshots. This technique guarantees that all individual microclusters in $\mathcal{M}$ are persisted prior to removal from $\mathcal{M}$ or being merged with another microcluster in $\mathcal{M}$. This allows a persisted and merged microcluster (i.e., those having a listid) in a snapshot to be broken down (via the microclusters subtractive property) into its constituent (individual), finer-grained microclusters during the macroclustering portion of the process. The opposite is also available, whereby the additive property allows finer-grained/individual and merged microclusters to be merged into more course grained microclusters that cover specified time horizons.

CluStream classifies snapshots into orders, which can vary from 1 to $\log_2(T)$, where T is the amount of clock time elapsed since the beginning of the stream (Aggarwal et. al., 2003). The number of snapshots stored over a period of T time units is

$$(\alpha + 1) * \log_2(T)$$

For example, if $\alpha = 2$ and the time unit or granularity is 1 second, then the number of snapshots maintained over 100 years is as follows:

$$(2+1)*\log_2(100*365*24*60*60) \approx 95$$

CluSandra will not have to operate within the same space (memory) constraints as CluStream; therefore, for CluSandra's microclustering process, microclusters do not have to be merged and there is no need to search for possible outliers that can be targeted for removal from $\mathcal{M}$. If a new microcluster is required, it will be created by a CluSandra microclustering agent (CMA) and simply added to the Cassandra data store. Also, the CMAs will write directly to the Cassandra data store; therefore, there is no need for a periodic time interval scheme that persists $\mathcal{M}$ to secondary storage.

CluStream cannot guarantee that microclusters will not be lost. After a microcluster is added to $\mathcal{M}$, there exists the possibility that the machine may fail prior to the microcluster being persisted, in which case it will be lost. Also, CluStream does not address $\mathcal{M}$'s recoverability. If the machine fails and is restarted, CluStream cannot recover $\mathcal{M}$'s state prior to the failure. In other words, there is no transactional integrity applied to $\mathcal{M}$.

6.3 Curse of Dimensionality

Conventional clustering methods based on K-means (e.g., BIRCH and CluStream) that have been designed to identify clusters from the whole data space suffer from what is called, *curse of dimensionality* (Plant & Bohm, 2008). This curse is directly related to the sparsity of the data inherent in high-dimensional vectors/data records. With increasing dimensional space, all pairs of data records tend to be almost equidistant from one another. Another problem with high-dimensional data is that there exists clusters embedded within with subspaces of the high dimensional data; at times, different clusters may exists in different subspaces. As a result, it is often unrealistic to apply distance-based measures to high-dimensional data records for discovering clusters (Aggarwal, Han, Wang, Yu, 2004). Relatively recent work uses a technique called, *projected clustering* to address this curse. Projected clustering discovers clusters based on specific subsets of the dimensions, which in turn alleviates the sparsity issue. Even though sparsity of the data may preclude discovering meaningful clusters from on all the dimensions, some subset of the dimensions can always be found that yield meaningful clusters (Aggarwal et. al., 2004). The people behind CluStream designed another CF-based algorithm called, HPStream whose design attempts to address this curse of dimensionality. The CluSandra project does not attempt to address this curse and works with a conventional clustering method and full-dimensional data records. The project assumes that the data record's number of dimensions are relatively low and thus the initial clustering algorithm employed by CluSandra does not fall victim to this curse. However, it should be noted that CluSandra is a clustering framework, and not just a specific clustering algorithm, that can accommodate many CF-based algorithms. Future work may apply HPStream, or derivative of, to the CluSandra framework.

7 Apache Cassandra

Cassandra is an open source, distributed NoSQL or non-relational database management system (DMS). It provides massive scalability up to hundreds of terabytes, fault tolerance, availability across multiple machines or data centers, a decentralized shared-nothing architecture (i.e., no master/slave), tunable consistency, and it includes multi-language client support. It is also easily configured and deployed across multiple geographically distributed machines. Cassandra has out-of-the-box support for geographical distribution of data (Hewitt, 2010).

Cassandra is an open source project that is maintained by the Apache Software Foundation (ASF) and is available at http://cassandra.apache.org. Cassandra originated at Facebook in 2007 and the code was

donated to Google Code in July 2008. In March of 2009, it was then donated to the ASF to surround it with a stronger and broader open source development community. It started at the ASF as an incubator project and quickly reached top-level project status in February of 2010.

Cassandra is a very good fit for serving as the database for a clustering system that is designed to collect and analyze the vast amounts of data produced by an unbounded and evolving data stream. However, it is assumed that the data produced by an unbounded data stream is infinite and thus cannot be feasibly stored in any database system; be it relational, non-relational or hierarchical. Therefore, the CluSandra framework does not store each and every data stream record in the Cassandra DMS. Instead, CluSandra stores the microclusters or CFT structures previously described. Again, these are the synopsis data structures that are used to collect summarized statistical information regarding the data stream records.

The following provide a brief overview of some of Cassandra's more important features.

Distributed

Cassandra's distributed architecture allows it to be deployed and executed across multiple machines. The collection of machines upon which a Cassandra instance has been deployed is referred to as a cluster. Each machine in the cluster is referred to as a node and all nodes in a cluster are identically configured. The nodes can be distributed geographically, locally or combination of both. The cluster, which can comprise one or many nodes, is viewed by the end-user as a single unified whole. For upfront learning, development and test purposes, the cluster can comprise one node; however, Cassandra's benefits are best realized when distributed across many nodes.

Decentralized

Unlike other DMSs, a Cassandra cluster has no central master node that coordinates activities across slave nodes. In a Cassandra cluster, all the nodes are peers to one another and the data is automatically replicated across the nodes. In this type of distributed peer-to-peer architecture, which is also called symmetrical, there is no single point of failure and thus it is key to Cassandra's ability to provide very high levels of availability. It is also much simpler to set up than a master/slave configuration, because all nodes are identical and thus you don't need any special knowledge to set up the cluster.

Elastically Scalable

Scalability is a very important feature that allows a system to continue serving a greater number of requests with little degradation in performance (Hewitt, 2010). There are two types of scaling: vertical and horizontal. Vertical scaling, which is the simpler of the two, is the process by which you add more hardware and memory to your existing machine. Horizontal scaling is the process of adding more machines, whereby the data is spread across all the machines in such a fashion that no one machine carries the burden of processing requests. In a horizontally scalable system with a replicated data store, there must also exist some form of mechanism that keeps the data synchronized across all the machines.

Cassandra includes a special form of horizontal scaling called, elastic scalability. This type of scalability can seamlessly scale up and back down. When elastically scaling up, you add new nodes to the cluster and have them automatically begin participating without introducing any disruption or reconfiguration to the cluster. And the opposite is true when you elastically scale back down. That is, you can remove nodes from the cluster without disrupting the cluster.

This ability to seamlessly add and remove nodes from the cluster is what allows Cassandra to provide very high levels of availability and fault tolerance. If one node in the cluster fails, the other nodes automatically recognize this and account for the lost node.

Tuneable Consistency

Consistency means that a database read will always return the most recently written value. It is the 'C' in the transactional ACID properties of atomicity, consistency, isolation and durability. A description of consistency and all its implications for distributed and replicated data stores is beyond the scope of this paper. However, we should briefly describe what "tuneable consistency" means as this is a feature that is unique to Cassandra.

Tuneably consistent means that Cassandra allows the client to decide on the level of consistency in balance with the level of availability. The following, taken from Hewitt (2010), provides a bit more on this feature:

> *In Cassandra, consistency is not an all-or-nothing proposition, so we might more accurately term it "tuneable consistency" because the client can control the number of replicas to block on for all updates. This is done by setting the consistency level against the replication factor.*
>
> *The replication factor lets you decide how much you want to pay in performance to gain more consistency. You set the replication factor to the number of nodes in the cluster you want the updates to propagate to (remember that an update means any add, update, or delete operation).*
>
> *The consistency level is a setting that clients must specify on every operation and that allows you to decide how many replicas in the cluster must acknowledge a write operation or respond to a read operation in order to be considered successful. That's the part where Cassandra has pushed the decision for determining consistency out to the client.*

Column Oriented

Cassandra is referred to as a column-oriented non-relational database that uses *sparse* multidimensional hashtables to represent its data structures. With Cassandra, a row can have one or more columns; however, unlike a relational database, each row doesn't have to have the same columns as other rows of its type. Thus the term, "sparse".

The sparse multidimensional hashtables are a feature, because they alleviate the end-user from having to decide upon a data model ahead of time. This gives the end-user a lot of flexibility and allows for the on-the-fly modifications of feature-functionality. This may be handy in those cases where a development project is not given much time for up-front design analysis or the business requirements are rather fluid. Instead of designing a pristine data model and then designing queries around the model as in RDBMS, you are free to think of your queries first, and then provide the data that answers them (Hewitt, 2010).

High Performance

The following, which has also been taken from Hewitt (2010), provides an excellent quick overview regarding Cassandra's performance.

> *Cassandra was designed specifically from the ground up to take full advantage of multiprocessor/multicore machines, and to run across many dozens of these machines housed in multiple data centers. It scales consistently and seamlessly to hundreds of terabytes. Cassandra has been shown to perform exceptionally well under heavy load. It consistently can show very fast throughput for writes per second on a basic commodity workstation. As you add more servers, you can maintain all of Cassandra's desirable properties without sacrificing performance.*

Cassandra was picked for this project, because of all the above and more. However, the features that stand out the most are its extremely low cost of ownership, simplicity, availability and scalability. Also contributing to the decision is the support provided by it strong open source community of developers.

8 CluSandra

This section provides a high-level overview of the CluSandra framework. The CluSandra design specification, which serves as the implementation team's guide, provides much more detailed information on CluSandra's components and corresponding Cassandra data model.

As previously mentioned, CluSandra combines concepts from BIRCH and CluStream with Cassandra to provide a high-performance, distributed, scalable and fault-tolerant data stream clustering system. However, it is also a goal of this project to design certain aspects of CluSandra to serve as a framework that can accommodate other CF-based clustering algorithms (e.g., HPStream).

To meet the most demanding multi-data stream environments, CluSandra can elastically scale from a one or two node Cassandra cluster to a cluster comprising 10s if not 100s of nodes. The "multi-data stream" statement does imply that CluSandra is capable of concurrently processing multiple data streams.

Like CluStream, CluSandra tackles the one-pass data stream constraint by dividing the data stream clustering process into the two previously described operating phases, i.e., online microclustering and offline macroclustering. Microclustering operates in real-time, computing and storing summary statistics about the data stream in microclusters. This functionality can be found in CluSandra's MicroCluster Agent (MCA) component, which is deployed onto each and every Cassandra node. To take advantage of multiprocessor/multicore based nodes, the MCA can be configured to spawn multiple independent threads of execution. Macroclustering is the offline process by which end-users create and submit queries against the stored microclusters to discover, explore and learn from the evolving data stream.

As CluStream extended the BIRCH CF data structure to produce the CFT structure or microcluster, so does CluSandra by extending CluStream's microcluster as follows:

$$CFT = \langle N, LS, SS, ST, SST, CT, LAT, ID, LISTID \rangle$$

The CluSandra microcluster's CT and LAT parameters are two timestamp scalars that specify the creation time and last absorption time of the microcluster, respectively. More precisely, CT records the time the microcluster was created and the LAT records the time that the microcluster last absorbed a data stream data record. All timestamps in CluSandra are measured in the number of milliseconds that have elapsed since Unix or Posix epoch time, which is January 1,1970 00:00:00 GMT. The ID and IDLIST are also new; however, they were mentioned in the CluStream literature, but not reflected in the CluStream CFT.

Depending on the distribution of the data stream, as it evolves over time, microclusters of all sizes appear, disappear, and may reappear. The number and sizes of the microclusters are a factor of not only the data stream's evolutionary pattern, but also of the boundary and expiry-time thresholds specified for the microclusters. For example, the larger the boundary threshold, the fewer microclusters, because each microcluster will be able to absorb more data records. The expiry-time specifies how long a microcluster can remain active without having absorbed any data records. For example, if the expiry-time is 5 seconds and a particular microcluster has not absorbed a data record in that amount of time, then it will become inactive and represents a data pattern that has appeared and disappeared within the data stream. Recall that a stream S is viewed as an unbounded sequence of pairs $\langle s,t \rangle$, where s is a structured data record and $t \in T$ is the timestamp that specifies the arrival time of data record s on stream S. When a microcluster absorbs a data record, its LAT parameter is assigned the record's timestamp value and not the current time.

An inactive microcluster is no longer capable of absorbing data records; however, during macroclustering, it can be merged with other inactive and active microclusters to form a supercluster (more on superclusters later in this section). CluSandra's online microclustering phase, therefore, works within a specified sliding time window and updates only those microclusters that are within that time window. This minimizes the number of microclusters that the MCA selects for updating. The MCA does not issue a database select against Cassandra to acquire this active set for each and every data record that arrives. The MCA instead processes a batch or chunk of time stamped data records against one selected set of active microclusters. So every time a chunk of records is read in from the stream, the MCA selects a set of microclusters within the active time window for those records. Recall that there is at least one instance of a MCA running on each and every Cassandra node. Additional details on the MCA and how chunks of data stream data records are dispersed and load-balanced across the MCAs can be found in the CluSandra design specification.

As previously noted, an inactive microcluster represents a pattern in the data stream that appeared and then disappeared. Over time, a particular pattern in the data stream may also appear, disappear and then reappear. This is reflected or captured by two microclusters with identical spatial values, but different temporal values. The amount of time a pattern was active can be calculated by the CT and LAT parameters and more detailed statistical analysis can be performed based on the other temporal, as well as spatial parameters. For example, the temporal density of the data records and their spatial density with respect to one another and/or their centroid.

Recall that BIRCH, CluStream, and CluSandra are all based upon cluster boundary thresholds that are spherically related (i.e., radius and diameter); therefore, only cluster shapes of a spherical or concave nature can be applied. For example, the pattern depicted in figure 3.a would be a likely candidate, while the one in figure 3.b would not.

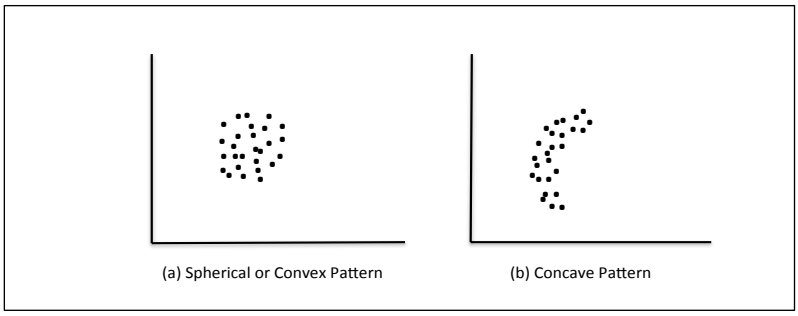

Figure 3 Concave and Convex Scatter Plots or Patterns

CluSandra introduces the supercluster, which is created when two or more microclusters are merged via their additive properties. A supercluster can also be reduced or even eliminated via its subtractive property. The ID parameter uniquely identifies a micro or supercluster and LISTID is a collection or vector of IDs that identify a supercluster's constituent parts (i.e., other super or microclusters).

Superclusters are created by the end-user during CluSandra's macroclustering process or phase of operation. Superclusters are created based on a specified distance measure (similarity) and time horizon. CluSandra superclustering is a type of *agglomerative* clustering procedure whereby individuals or groups of individuals are merged based upon their proximity to one another (Everitt et. al. 2001). However, unlike traditional agglomertive procedures where the results of a merger cannot be undone or separated, CluSandra's superclusters can be undone. What makes this possible is the subtractive property of the CF. Agglomerative procedures are probably the most widely used of the hierarchical clustering methods (Everitt et. al. 2001). When a supercluster is created, the earliest LAT of its constituent parts is used as the supercluster's LAT. The CT and LAT thus identify the time horizon that the supercluster spans.

Additional parameters may be added to the CluSandra CFT structure. For example, a marker that can be used to quickly identify it as being either a micro or supercluster and thus be used as a filter for select queries.

This first release of CluSandra will deliver the core data collection portion of the system and a set of canned queries that can be executed against CluSandra's microcluster data store. This set of queries can be used to test the data store and demonstrate aspects of macroclustering. However, this first release will not go as far as delivering any visual or graphical front-end to the data store nor any sort of query engine. That functionality can be assigned to one or a series of follow-up projects.

9 Conclusion

This paper has proposed and defined, at a high-level, the basis for a clustering analysis system/framework, called CluSandra, whose design targets the evolving data stream. CluSandra is based upon and extends the BIRCH and CluStream algorithms and is combined with the Apache Cassandra database system to lift the time and space constraints imposed by the evolving data stream. It, therefore, greatly facilitates the exploration and analysis of the stream's evolution over varying time horizons while at the same time, operating in real-time to collect statistical information for the stream.

Over the years, many clustering algorithms have been developed for various applications. Relatively few have been applied or analyzed within the context of the evolving data stream environment (Aggarwal et. al. 2003). CluSandra builds upon BIRCH and CluStream to not only provide a data stream clustering system, but also a unique and flexible framework from which existing and new clustering algorithms can be applied to the evolving data stream. Examples of future algorithms or works that can be developed within the CluSandra framework: addressing data streams comprised of varying data types and their standardization, incorporating new proximity measures that can detected clusters of various shapes, addressing missing data values, and addressing the curse of dimensionality problem.

The system itself can also serve as an example or basis for more sophisticated clustering systems that can tackle different data types, beyond just numerical types, and a mixture of data types.

10 References

Domingos, P., Hulten, G. (2000). Mining high-speed data streams. *Knowledge Discovery and Data Mining*, pages 71-80.

Aggarwal, C.C., Han, J., Wang, J., Yu, P.S. (2003). A framework for clustering evolving data streams. Proceedings of the 29th VLDB Conference, Berlin, Germany.

Aggarwal, C.C., Han, J., Wang, J., Yu, P.S. (2004). A framework for projected clustering of high dimensional data streams. Proceedings of the 30th VLDB Conference, Toronto, Canada.

Gama, J. (2010). Knowledge discovery from data streams. Boca Raton, FL: Chapman & Hall/CRC

Bifet, B. and Kirkby, R. (2009). Data stream mining. University of Waikato, New Zealand: Centre for Open Software Innovation

Hewitt, E. (2010). Cassandra, the definitive guide. Sebastopol, CA: O'Reilly Media, Inc.

Zhang, T., Ramakrishnan, R., Livny, M. (1996). BIRCH: An efficient data clustering method for very large databases. ACM SIGMOD, pages 103-114

Golab, L., Ozsu, M.T. (2003). Issues in data stream management. SIGMOD Record, Vol. 32, No. 2

Hebrail, G. (2008). Introduction to data stream querying and processing. *International Workshop on Data Stream Processing and Management*. Beijing.

Han, J., Kamber, M. (2006). Data mining, concepts and techniques (2nd edition). San Francisco, CA: Morgan Kaufmann Publishers.

Everitt, B.S., Landau, S., Leese M. (2001). Cluster analysis. (4th edition). London, England: Arnold Publishers

Zhang, P., Zhu, X., Guo, L. (2009). Mining data streams with labeled and unlabeled training examples. Ninth IEEE International Conference on Data Mining, pages 627-635.

Masud, M., Gao, J., Latifur, K., Han, J. (2008). A practical approach to classify data streams: training with limited amount of labeled data. Dept. of Computer Science, University of Dallas @ Texas, Dept. of Computer Science, University of Illinois @ Urbana Champaign

Giuseppe, D., Deniz H., Madan J., Gunavardhan K., Avinash L., Alex P., Swaminathan S., Peter V., Werner V. (2007). Dynamo: Amazon's Highly Available Key-value Store. Stevenson, WA: ACM

Plant, C., Bohm, C. (2008). Novel trends in clustering. Technishe Universitat Munchen, Munchen Germany, Ludwig Maximialns Universitat Munchen Munich Germany