Querying of Time Series for Big Data Analytics

Vasileios Zois, University of Southern California, Department of Computer Science
Charalampos Chelmis, University of Southern California, Department of Electrical Engineering
Viktor Prasanna, University of Southern California, Department of Electrical Engineering

ABSTRACT

Time series data emerge naturally in many fields of applied sciences and engineering including but not limited to statistics, signal processing, mathematical finance, weather and power consumption forecasting. Although time series data have been well studied in the past, they still present a challenge to the scientific community. Advanced operations such as classification, segmentation, prediction, anomaly detection and motif discovery are very useful especially for machine learning as well as other scientific fields. The advent of Big Data in almost every scientific domain motivates us to provide an in-depth study of the state of the art approaches associated with techniques for efficient querying of time series. This chapter aims at providing a comprehensive review of the existing solutions related to time series representation, processing, indexing and querying operations.

Keywords: Include some keywords

INTRODUCTION

Data are recorded as series of observations which evolve over time. These observations have a collective value and usually describe a single entity or object which provides a different footprint or impact on the microcosm of the scientific field under consideration. These series of observations are usually described with the term time sequences/series. Researchers typically have to analyze thousands of time series objects at a time, often requiring operations such as searching, classification, clustering, predictive modeling, visualization and summarization as part of their analytic processing and decision making. Over the years a plethora of approaches have emerged attempting to formalize the problem and provide an efficient and generic solution to the support of such operations. The advent of Big Data as a consequence of the increasing use of the internet of things combined with emerging applications on cyber-physical systems has made the problem more complex and demanding.

At the top level analysts from diverse scientific domains need to have guarantees that ensure the reliability of data gathering from different sources in parallel, fast access to stored information for complex analytics to be available and real-time detection of events in monitoring systems. The provision of such guarantees is a great challenge for researchers who have been concerned with the solution of problems related to how data are stored, processed and visualized at the higher level.

Mainly researchers are concerned with univariate (Chatfield, The analysis of time series: an introduction, 2013) time series data and most of the literature is relevant to them. However there are cases where time series can be multivariate (Box, Jenkins, & Reinsel, 2013), (Wang, Zhu, Li, Wan, & Zhang, 2014) or time series graphs (Park, Priebe, & Youssef, 2013), (Wang, Tang, Park, & Priebe, 2014), (Yan & Eidenbenz, 2014). A multivariate time series is described as a sequence of observations with multiple
values at every given point in time. Time series graphs describe a unique category of time sequence observations that refer snapshots of evolving/temporal/dynamic graphs. A collection of these snapshots are what constitute the time series object. This chapter will focus mostly on univariate or multivariate time series and queries related to them. Graph time series are mentioned for completeness and will be discussed in a high level as part of the latter developments in the field related to time series data analysis and querying.

**Definition and Notation for Time Series Data**

Time series data refer to a collection of data points which represent the evolution or behavior of a specific entity in time. Examples include, but are not limited to consumption information from distinct customers on a power grid, stock price closing values (Bao, 2008), patient vital signs as monitored by special equipment and more recently tweets, blog posts. A time series object is usually represented using a vector of consecutive values, the index of which denote the relative time of the observation. Formally, a time series object $S$ is described by the vector $[s_1, s_2, ..., s_m]$ where $s_j$ is the observed value of the j-th time interval. Values are assumed to be presented in the same order as they have been observed, so for $i < j$, $s_i$ appears before $s_j$ in the corresponding vector. The length of the time interval between consecutive values can be fixed or variable. For analysis it is not required to know the exact time of the observations as long as they can be distinguished from each other and order relatively to the time they were observed. Different sampling rates can make it difficult for distinct time series objects to be compared. Interpolation is a standard preprocessing operation used to fill gaps between intervals induced either by incompatible sampling rates or missing values. Interpolation techniques will not be discussed in this chapter but, are mentioned for completeness as part of time series workflows. Other common preprocessing steps include time series normalization, which is effective for accurate shape based matching. Normalization is performed by eliminating the amplitude value through subtracting the mean and dividing with the standard deviation.

Operations on time series data usually fall into two categories, operations related to analysis or to forecasting. Some common operations employed on time series objects include classification (Kamath, Lin, & De Jong, 2014), clustering (Eu etc, Ortega, & Alvarez-Esteban, 2014), motif discovery (Mueen, 2014), query by content (Esling & Agon, 2013) and many others. Efficient implementation of such operations presents a great challenge to researchers who need to ensure high throughput and consistent availability of data (Loboz, Smyl, & Nath, 2010). As time series data are inherently large to be processed entirely in memory there is a need for solutions that consider and minimize the effect of secondary memory access. Sliding window is commonly used to group fixed-size windows so that it can be processed in memory incrementally. However, this technique can increase the number of random I/Os if data are not organized in some way (Anderson, Arlitt, Morrey III, & Veitch, 2009), which will affect overall the execution time. The most common practice which has been studied extensively over the years is indexing of time series. Indexing is very useful as it can be effectively used for shape based matching operations which are common for many of the complex analytics operations that were mentioned previously. Although indexing improves access to the underlying data it does not provide an easy to use management scheme. It is necessary for scientists that are a prime example of people who make use of time series data analytics, to have an easy access to services that support real time querying and gathering of information from different sources. This can be achieved by a definition of some kind of a management system along with a query language suitable for the needs of common user.

In this chapter we will cover the different indexing techniques along with indexing data structures frequently used with time series data. Moreover, existing attempts for proposing time series management systems along with suitable languages for querying the underlying data are discussed.

**TIME SERIES REPRESENTATION**
A major concern when managing time series data is the support of efficient storage and retrieval operations. A single time series object usually consists of several hundred to thousands of values (Chatfield, The analysis of time series: an introduction, 2013). This is a direct result of the sampling rate at which data are gathered, which overall increases the required storage space. Secondary memory is often used to retain the gathered data which need to be accessed directly for query answering. For complex analytics multiple passes over the data are necessary adding to the overall processing time making it highly inefficient to use the secondary memory directly. Motivated by this issue several representation techniques have been proposed, aiming at minimizing the number of I/Os leveraging the main memory for processing thus reducing the overall execution time. In order to resolve this issue different representation method are often coupled with multidimensional data structures, which are being used to prune the search space for relevant objects thus minimizing the number of I/Os. Altering the representation of the original time series object is pivotal to efficient indexing. Multidimensional data structures often suffer from the so called dimensionality curse when attempting to manage high dimensional data such as time series data. Dimensionality curse refer to the series of phenomena arising from handling data of immense dimensions. In the context of data structures the dimensionality curse affects their performance resulting in access complexity which is equivalent to linear scan.

A common method to deal with dimensionality curse is to perform dimensionality reduction as in the case of General Multimedia Indexing algorithm (GEMINI). GEMINI relies on extracting a limited number of features from a time series object, aiming at preserving its general characteristics which are then to be used to identify similarities with a submitted query. Extracting features is achieved by utilizing a function which is defined by the user and corresponds to a transformation of the original object. The transformation coefficients are the derived features used to create an index in main memory. Time series objects resulting from this method can be seen as being defined in a new space referred to as feature space. The principle idea behind this procedure is that objects are represented in low dimensional space which can be efficiently indexed by existing data structures. The choice of a transformation can greatly affect the performance of this method, as the approximation of the original object is correlated to the number of I/Os which are necessary for retrieval of relevant objects to the specified query.

A natural question arises that is related to the selection of a suitable transformation. The quality measure of a transformation is measured as a function of the reconstruction error. A transformation is good, if it achieves a minimal reconstruction error of the original object retaining at most $M$ features. This presents a good metric for comparing different transformation techniques as it can give an initial estimation of the pruning power of the transformation. The most important property to which each proposed transformation needs to adhere to is the so called lower bounding lemma. This property states that the distance between two objects in the feature space needs to lower bound the distance of these objects in the original space (Faloutsos, Ranganathan, & Manolopoulos, 1994). For example, given two time series objects $S$ and $Q$ with their corresponding feature vector $F(S), F(Q)$ that contain the coefficients of a transformation $F$, the following inequality must hold:

$$1)D_F(F(S), F(Q)) \leq D_T(S, Q).$$

Here $D_F$ and $D_T$ denote the distance metrics used in the feature and the original space respectively. Inequality (1) provides a framework for choosing but also evaluating a given transformation as a tighter bound indicates a better approximation of the original object thus reducing the number of false positives. An important observation that was initially made in (Faloutsos, Ranganathan, & Manolopoulos, 1994) was that transformations obeying this equation will not produce any false negatives. Specifically, given a query time series $Q$ and a tolerance value $\epsilon$, it can be ensured that all the time series with distance at most $\epsilon$ from $Q$ are going to be retrieved, only by consulting the indexing structure that holds the transformed objects. False positives are a result of underestimating the true distance between any two time series objects since only their approximate form is considered. The number of false positives can be is small for transformations that have a tight lower bound. In fact, the approximation accuracy increases
with the number of retained features. However, it is usually the case to retain a small number of features, a practice which can be also used as a quality measure for different transformations that achieve a low reconstruction error with the minimal number of features retained. A tradeoff is usually observed between the number of features retained resulting in increased dimensions and the accuracy of the retrieval process where an overhead is induced by the post-processing step which is meant to discard false positive.

Some desirable properties of an efficient indexing schema based on the GEMINI algorithm are presented in (Faloutsos, Ranganathan, & Manolopoulos, 1994) and include efficient insertion and deletion of new data as well as support of various length queries. Support for multiple distance measures is also highlighted in (Keogh, Chakrabarti, Pazzani, & Mehrotra, 2001). In the following section some existing transformation techniques are reviewed, focusing on their ability to support the aforementioned properties.

**Discrete Transformations**

The use of Discrete Fourier Transformation (DFT) as a suitable dimensionality reduction method was introduced in (Faloutsos, Ranganathan, & Manolopoulos, 1994). The authors proposed using few DFT coefficients which were produced by the original time series object to create an index. This idea is supported by the fact that the first few coefficients usually preserve the well the characteristics of a signal making this method ideal for dimensionality reduction. It is also an efficient method based on the existence of the Fast Fourier Transformation (FFT) which can be computed in $O(n \log n)$, where $n$ is the number of observations in a time series object. Finally, it can be proven that the transformation obeys the lower bounding lemma which is based on Parseval’s theorem (Oppenheim & Schafer, 1975). DFT preserves the energy of a given signal suggesting there is a 1-to-1 mapping of the generated coefficients to the original values of a time series object. Considering also that DFT is a linear transformation, the Euclidean Distance (ED) between two time series object is retained, i.e $ED(x,y) = ED(X,Y)$, if $x,y$ are two time series and their corresponding DFT is $X,Y$. Retaining a larger number of Fourier coefficients results in a better approximation of the original object, however, retaining too much information has an inverse effect on the efficiency of indexing. According to the authors keeping 2-3 coefficients are enough for indexing purposes, as DFT concentrates the energy on the first few coefficients (Agrawal, Faloutsos, & Swami, 1993). It was presented in (Ding, Trajcevski, Scheuermann, Wang, & Keogh, 2008) that DFT works best for time series data that present some periodic behavior. Same is true for other transformations that are classified as spectral dimensionality reduction techniques, including Discrete Cosine Transformation (DCT) (Korn, Jagadish, & Faloutsos, 1997) and Chebysev Polynomials (CP) (Cai & Ng, 2004).

The Discrete Wavelet Transformation (DWT) is another dimensionality reduction technique inspired from the field of signal processing. It was first proposed as an indexing schema in (Chan, Fu, & Yu, 2003). An important property of DWT is that it can create multi-resolution representations for a signal, thus enabling a better representation of the original object through the use of its local characteristics in time. Specifically in contrast to DFT which retains information about the global shape of a time series object, DWT retains local information which typically encodes a coarse approximation of the original sequence. Additionally the Haar transform (Chan, Fu, & Yu, 2003) can be computed in linear time as opposed to DFT which requires $O(n \log n)$ steps.
Data Oriented Transformations

An important class of dimensionality reduction techniques focuses on adapting to data specific characteristics in order to provide a more accurate approximation of the original object. Aiming at minimizing the number of features retained these techniques present a good performance when used to index time series of non-periodic shape. In contrast their efficiency drops for periodic data as the limited number of coefficients combined with the way they are extracted (e.g. sampling rate) create an inaccurate approximation of the original series.

Singular Value Decomposition (SVD) (Korn, Jagadish, & Faloutsos, 1997) overcomes the limitations of spectral methods and can be applied to sequences of arbitrary size. Even though SVD can achieve optimal representation it is computationally expensive and needs to be recomputed with the addition of new time series objects. However, there has been some work on approximately constructing the index to accommodate insertions more efficiently (Golyandina & Zhigljavsky, 2013). SVD along with Principle Component Analysis (PCA) can be used to achieve the optimal representation, although their applicability is limited by their computational complexity and their inability to efficiently transform data in main memory.

Piecewise Aggregate Approximation (PAA) is a dimensionality reduction method introduced in (Yi & Faloutsos, 2000), (Keogh, Chakrabarti, Pazzani, & Mehrotra, 2001). PAA resembles a wavelet transform making use of the average values among consecutive intervals. The main difference is that it uses a stable window that divides time series object in equal sized frames and retains the average value for each frame. The method has also linear complexity in the size of the input and presents a good approximation for bursty data (Wang, Mueen, Ding, Trajcevski, Scheuermann, & Keogh, 2013). The authors prove that the transformation lower bounds the ED and also provide extensive experiments that verify the pruning efficiency as well as the computational complexity of this method. In contrast to other approaches, PAA can also support queries of diverse lengths, leveraging on the use of a dedicated distance function defined in the feature space (Keogh, Chakrabarti, Pazzani, & Mehrotra, 2001).

An extension of PAA, known as Adaptive Piecewise Constant Approximation (APCA) was introduced in (Keogh, Chakrabarti, Pazzani, & Mehrotra, 2001) and aimed at minimizing the reconstruction error of the original time series object. Similar to PAA a time series object is divided into disjoint segments which can be of variable width. According to APCA, the highly variable parts of a time series should be described using more segments as opposed to parts of low variance. APCA makes use of two values that describe the average value and the right endpoint of a segment. Deciding on the optimal creation of segments is costly, requiring the use of dynamic programming (Pavlidis). When an optimal solution is not required a heuristic can be used to determine the size and number of the individual segments in a time series object. Performing a wavelet compression first and then converting the solution back to the APCA representation results in an algorithm with only a fraction of the computation cost of the dynamic programming approach (Keogh, Chakrabarti, Pazzani, & Mehrotra, 2001). Along with APCA

Figure 1 - Categories of Representation Techniques for Time Series Data.
two distance measures are defined with one being approximate and the other exact meaning that the former does not obey the lower bounding lemma as opposed to the latter which does.

Piecewise Linear Approximation (PLA) approximates a time series object with a set of straight lines while minimizing the overall reconstruction error (Chen Q., Chen, Lian, Liu, & Yu, 2007). The use of PLA as an indexing transformation requires the definition of a distance metric that obeys the lower bounding lemma (Chen Q., Chen, Lian, Liu, & Yu, 2007). PLA retains two coefficients for each line segment which are used to describe the equation of the corresponding line. In the same work a dedicated function is presented and used on these coefficients to lower bound the ED. Indexing with PLA is evaluated against real and synthetic data and compared with such representation methods as APCA and CP achieving a higher pruning power.

Motivated by existing compression methods, the work in (Bagnall, 2006) presented a bitwise representation which is used to create clipped data from the original time series. This method supports efficient shape-based similarity queries. Clipping or hard limiting is the process of transforming time series to a bitwise representation where the digit 1 describes a point that is above the average value and subsequently 0 describes the exact opposite. This technique does not actively reduce the size of the time series object since it requires the same number of bits. However, it results in objects that are highly compressed and can be managed easily using efficient bitwise matching operations. The advantage of using clipped data to represent time series is that there is no need for the definition of extra parameters as opposed to techniques such as APCA, PAA, and IPLA. In the same work, special distance functions are introduced which are designed to lower bound the ED and DTW distances. In the case of the former function, it is indicated by the authors that it can be used for comparison even if one of the time series is not clipped. The creation of the distance function that lower bounds DTW, is based on the work of Keogh & Ratanamahatana, 2005) where two time series that enclose the transformed object are used to compute an approximation of corresponding distance. Finally, it is argued that the discriminatory power of clipped data compared to unclipped data is the same. This suggests that through clipping the characteristics of an object can be retained enabling accurate matching.

The existing time series representation techniques focus mostly on providing an efficient indexing schema to support query by content. Higher level queries such as classification, summarization, anomaly detection and prediction can be coupled together with simple shape matching queries to provide an answer. However, extending simple matching queries to incorporate these operations is non-trivial and requires some extra work to enable efficient processing. Researchers attempted to use existing algorithms from the field of regular expression matching, through transformations that attempt to discretize continuous values and convert them to alphanumeric characters. In (Andre-Jonsson & Badal, 1997), signature files where used as means of representing time series data. There, time series are represented as a set of alphanumeric characters which describe the derivative between consecutive points. The characters are selected based on a predefined alphabet which maps a range of values to specific letters. The attainable space reduction results from the small number of bits that are required to encode the alphabet set. However, accessing the data requires a linear scan of the signature files which is computationally expensive. Moreover, Interactive Matching of Patterns with Advanced Constraints in Time-series (IMPACTS) databases was presented in (Huang & Yu, 1999).

In similar manner as before, continues values are mapped into discrete symbols. Similarly to before the change ratio between consecutive values is mapped to discrete symbols. The values being subjected to this transformation are the minimum and maximum values observed in the equal sized disjoint intervals to which the original time series object is divided. In this work, a suffix tree is utilized to retain the transformed objects. Discretization usually affects the way matching is performed, resulting from symbols which are not partially comparable as numbers. Range queries are supported using the aforementioned transformation, through relaxing the matching criteria to consider symbols as equals within a specific tolerance. This method can support queries of different granularity, biased similarity queries, vague trend queries, regular expression queries and maximum mismatch queries.

Symbolic Aggregate approXimation (SAX) is a transformation based on the symbolic representation of values in a specific, introduced in (Lin, Keogh, Wei, & Lonardi, 2007). SAX was
motivated by the inability of previous solutions to support efficient indexing of time series. Supporting symbolic representation coupled with indexing capabilities was achieved through the combination of dimensionality reduction on top of which discretization techniques will be executed. Initially, PAA is used to extract a certain number of coefficients that describe the characteristics of a time series object. It is followed by a symbol mapping phase where specific symbols are assigned to a specific range of values using a set of breakpoints which are calculated utilizing a Gaussian curve. In the same work, a distance function that measures the similarity of two transformed time series objects is presented, based on the relaxed comparison between symbols, i.e., consecutive symbols assigned to neighboring range intervals are considered equal.

There exist several variation of SAX aiming mostly at improving its indexing performance and expanding its use for billions of time series objects. Examples include iSAX (Shieh & Keogh, 2008) and iSAX2+ (Camerra, Rakthanmanon, & Keogh, 2014). iSAX is an extension of the regular SAX representation designed to make indexing efficient. The resulting word strings are represented with iSAX as integers based on the binary representation of each character created from the breakpoints in regular SAX. Although there can be time series objects that map to words containing characters of different cardinality, the authors prove that it is possible to have a meaningful comparison between them. In fact the introduction of iSAX deals with this issue by lower bounding the distance between the time series of different cardinality. This is achieved through the expansion of the characters with lower cardinality through the addition of the extra missing bits. Overall this is based on the observation which indicates that the breakpoints used in each case for the definition of the corresponding character set are a proper subset of the breakpoints in the time series object represented in a higher cardinality. An indexing structure is also discussed with iSAX taking advantage of each special representation and treating it similarly to a set of string for partial matching. The structure proposed is similar to a suffix tree on the bits of the different characters described by the corresponding integers indicating the breakpoints. Similarly, iSAX2+ utilizes the same framework although it improves on the indexing mechanism through the introduction of a balanced tree structure that can accommodate the transformed objects. Along with the introduction of the corresponding balanced structure a bulk loading algorithm is also discussed and certain node splitting policies.

DATA STRUCTURES

Representation techniques are often used in combination with multi-dimensional data structures to provide a complete schema that is able to support efficient querying of time series data. Researchers usually suggest R-tree and its variations (R*-tree, R+tree) being suitable for most types of queries. However, there is a significant amount of research associated with proposing multi-dimensional data structures as well as adapting existing ones for the emerging computational models (e.g., distributed computing, peer-to-peer systems). This section will provide an overview of related work on the field of multi-dimensional data structure development, identifying the challenges of managing high-dimensional data.

Multidimensional Data Structures

Research on indexing multi-dimensional data structures includes several attempts as early as 1970s. Early solutions aim at supporting point or range queries using multi-key indexing techniques. A comprehensive survey of early data structures is presented in (Bentley & Friedman, 1979) as discussed in high detail as follows. Projection creates a sequence of records sorted by a specified attribute. Queries are executed based on a specific attribute which can be easily found through binary search and then a brute force method checks a smaller list to match the remaining attributes. Using Cells is basically a technique that divides the space to multiple squares and retains indexes of different granularity. During query time each index needs to be referenced in a hierarchical way to detect relevant data. A natural generalization of binary trees is the so called k-d trees which have been used for indexing multidimensional data in main
memory. Similar to binary trees a key is used to discriminate between two nodes which are chosen from attributes that present the maximum spread of the sub-collection represented by the corresponding node. The median value of this attribute is used to partition between the left and right sub-tree. This technique is similar to projection as it utilizes one attribute from the n available, although it does that per node. Range trees also use the binary tree as main building block. For the 2-dimensional case, range trees are created as a binary tree on a single dimension retaining a sorted array at each node of the second dimension. Similarly for more dimensions a number of sorted lists are used.

The R-tree (Guttman, 1984) was motivated from the need of supporting efficient insert, delete and access operations. It groups nearby objects together using a minimum bounding rectangle (MBR) representation. This procedure recursively creates several tree levels, each having a decreasing size MBR that contains the multidimensional objects. It is usually compared to how B-trees operates on data using an upper and lower bound denoting the number of objects to be retained at each node. These bounds are usually defined as a multiple of the disk block size. The idea is to retrieve relevant data that can be grouped together and fitted in a single block thus spending only one I/O operation. Intermediate nodes of the R-tree, retain rectangles containing several points grouped together and the leaf nodes contain pointers to specific blocks where the points are stored. The search speedup is connected to minimizing the overlap between MBRs of the same level. In this context, a single node needs to be expanded at each level of the tree in order for relevant objects to be retrieved at the leaf level. Although this is a straightforward concept it is often too expensive to optimally decide how to split overflowing nodes in order to create non-overlapping MBRs. Heuristics are utilized to make approximate grouping decisions which on average yield logarithmic search time. However, in the worst case the search complexity will degenerate to linear time. R+ trees were created as an extension to R-trees, aiming to mend the problems induced from sub-optimally splitting nodes during the insertion of new data. The main idea is to replicate the children of each node when an overlap is detected, requiring only the expansion of a single node having the replicated information. In the worst case this technique could double the required space for the structure, although the search complexity is logarithmic in terms of the input size. Another variation of the R-tree is the R*-tree which aims at minimizing the area of overlapping rectangles while also increasing the coverage. It was observed that the optimality of the grouping procedure depended on the order in which points where inserted. Motivated by this the R*-tree enforces a reinsertion policy along with a revised splitting algorithm to achieve a better grouping.

The importance of using an effective splitting policy for overflowing nodes was recognized early and was correlated with the inherit performance of the R-tree data structure as well as its variations. As it was observed the policy that would produce the minimum overlap between rectangles while also maximizing their coverage in the multidimensional space would be able to sustain a logarithmic access time independent from the number of insertions and deletions. This idea was the motivation behind many direct extensions of the R-tree. It also introduced another aspect of research related to R-tree optimization which was concerned with proposing suitable splitting policies. Some examples include the works of (White & Jain, 1996) and (Berchtold, Keim, & Kriegel, 2001). The former paper presents a method that assumes a global view of the dataset and partition points recursively based on the dimension with the highest variance. The latter paper presents a method that attempts to minimize the overlap between rectangles by introducing the concept of a supernode. A supernode can sustain more elements than the maximum possible of a simple node. It presents a better approach relative to overlapping directory node which would require expanding children nodes that are unrelated to the corresponding query.

An interesting approach on the design of a multidimensional data structure is the TV-tree (Lin, Jagadish, & Faloutsos, 1994). This data structure is related to dimensionality reduction and feature extraction techniques. The intuition behind the TV-tree is realizing that data objects consist of features that can effectively discriminate it from other objects. Based on this observation objects are represented in a reduced space based in which enough features are retained from individual objects and are used to discriminate them. The performance of this method depends on the application as it is effective only when dimensions can be ordered by their significance and also if feature vectors exist that allow shifting (i.e matching of coordinates on ordered feature vectors) of active dimensions.
Grouping of objects was effectively achieved by defining hyper-rectangles which then were represented by tree nodes in the R-tree. Alternatively grouping can be achieved with the definition of shapes other than a rectangle, an idea which was first introduced in (White D. A., 1996). There a similar structure to the R-tree, the SS-tree is proposed. The SS-tree partitions multidimensional points into groups which are formed using hyper-spheres. The SS-tree structure has a better performance on nearest neighbor queries (NN) since it divides the space into short diameter regions and more effectively it can support partially matching queries. A variation of the SS-tree is the so called SR-tree introduced in (White D. A., 1996). It aims at creating partitions of points utilizing the intersection between rectangles similar to those defined in R-trees and spheres which are utilized by SS-trees. The performance of the SR-tree was evaluated in comparison with the SS-tree and showed greater pruning capabilities as well as a decreased complexity on the insertion operations.

Recent attempts on defining multi-dimensional data structures include the works in (Li, Moon, & Lopez, 2004), (Assent, Krieger, Afschari, & Seidl, 2008) which are dedicated for time series data. Dimensionality reduction techniques are coupled with ideas from existing data structures to achieve a better performance overall. The first paper follows the concept of grouping objects in space using MBRs and introduces a new way to do so with Skyline Bounding Rectangles (SBR). An effective grouping aims at minimizing the overlapping regions between distinct nodes in the index. SBRs are free of internal overlap as they are defined using two skyline bounds (upper and lower bound) and two vertical lines between the skylines at the start and end time point. Since the defined skylines are as large as the indexed time series data, a dimensionality reduction technique is necessary to reduce the space requirements of the resulting data structure. Consequently the pruning power of the index is affected by the tightness of the lower bound introduced by the selected dimensionality reduction method. The defined metric is used to determine the distance of a query to a corresponding SBR. The second paper introduces the TS-tree data structure which is a variant of a B-tree data structure combined with dimensionality reduction using symbolic representation. Again a significant aspect of the corresponding method is to reduce or eliminate the overlap between distinct sub-trees. It relies on separators which are similar to B-tree keys indicating the range of values residing in the children nodes. In this case the leaves of the tree contain groups of time series with values less or more than the separator key respectively. As the authors indicate this method is not very effective with real valued time series. So they suggest quantizing the time series objects based on a fixed alphabet. This is possible using existing dimensionality reduction techniques (e.g SAX). As before the pruning power of the defined index is related to that of the representation technique being used. Through extensive experimentation the authors so the significant improvement in terms of page access and compactness of the specified data structure, when compared to existing solutions for the execution of nearest neighbor queries.

As distributed systems are becoming more predominant in providing solutions for data intensive applications, there is an increasing need in successfully migrating existing data structures or inventing new ones to enable high throughput operations. Distributed systems owe their success to efficient data partitioning mechanisms that enable parallel read and write operations which reduce the latency of insert, update and retrieve operations. Researchers account for that in their solutions aiming at increasing the availability of data for concurrent complex queries. Initial attempts focus mostly on peer-to-peer systems and on range queries for multidimensional-data. Range-Queriable Data Structure (RAQ) (Nazerzadeh & Ghodsi, 2005) is an example of a peer-to-peer data structure supporting range queries on multi-dimensional data. It utilizes a partition tree which divides the k-dimensional space into partitions retained in the nodes of the tree. Leaf nodes contain a single data point and internal points refer to the points existing in their children nodes. The partition tree is employed on top of the network nodes which are responsible for a certain group of data points. Each network node retains links to other nodes based on the structure of the partition tree. This enables forwarding of query requests to nodes that have the relevant data. The idea behind the RAQ is similar to an R-tree where data are grouped in hyper-cubes enabling retrieval of only relevant to data regions.

A fairly recent work on distributed data structures introduced the DGFIndex (Liu, et al., 2014) which is motivated by an emerging cyber-physical system known as Smart Grid. The context is that
monitoring systems deployed on top of a power grid need the ability to handle massive amounts of multi-dimensional data which refer, either to time series data representing consumption information or metadata related to the entity being monitored. Existing data warehousing systems (e.g. Hive) are suitable to replace conventional Relational Database Management Systems in managing massive amounts of information. However, it is important to eliminate redundant I/Os during retrieval operations to improve the availability of data. The DGFIndex was designed utilizing grid files to split the data space in Grid File Units (GFUs) which consist of a GFUKey and GFUValue. The GFUkey is a concatenation of the left lower coordinates of each dimension for the specified region represented. The GFUValue is a pointer in the split containing rows that fall into the given region as well as certain pre-computed aggregate functions (e.g. max, min, sum) which can be defined dynamically. The defined functions need to be additive in order for update operations to be fast. Constructing the index requires the definition of certain parameters related to the splitting policy (interval size on every dimension) and the number of dimensions that constitute the given index. Although this is a static approach it is sufficient for monitoring operations which usually require specific aggregation operations.

**Distance Measures**

Measuring similarity of time series objects is usually achieved using the classic Lp-norm distances. Some commonly used Lp-norms are the Euclidean Distance and Manhattan Distance. They are mostly used for query by content operations but can also be used for complex operations such as motif discovery, classification and clustering operations. Lp-norms can be classified as a family of lock-step/static measures (Ding, Trajcevski, Scheuermann, Wang, & Keogh, 2008) because they statically match points of two sequences that have the same time index. Although these distance metrics can be sufficient for certain applications such as exact matching, they are unable to handle operations like partial matching, phase shift matching.

In many cases fields like signal processing and image recognition need to have the ability to deal with phase shifting to make sense of content queries. This is because there exist applications for which there is no clear understanding of the shape to be matched or it cannot overall be defined given a single example. Other situations arise when a query sequence does not match the size of sequences in our database. Dynamic Time Warping (DTW) is a distance measure designed to deal with these limitations. It is used to match out of phase sequences that are similar given the domain of the application. Detecting movement using a camera or differentiating words in speech recognition present some applications that can be favored while using DTW since the observations under examination often vary time or speed. Given some restrictions like the maximum size of the warping window, DTW manages to find the optimal warping path. The optimal warping path refers to sets of pairs between data points appearing in the sequences being compared, that are of distance in time less than the maximum warping window and manage to give the minimum distance value overall. Although DTW performs better than lock-step/static distance measures and it can be viewed as a generalization of basic matching using a dynamic point association, it is computationally expensive to use and it is not classified as a metric. The latter limitation makes impossible for DTW to be indexed directly making difficult to use it with the indexing methods mentioned previously.

The popularity of DTW is what inspired the scientific community to overcome the difficulties associated with creating an indexing schema for it. (Yi, Jagadish, & Faloutsos, 1998) an indexing method was proposed which was based on the idea of projecting the original time series objects to a k-dimensional space while approximately preserving their distances. A multi-dimensional data structure will then be used to prune the search space while retrieving relevant objects based on the DTW metric. Although this technique achieves significant speedup compared to sequential scanning, it can allow false dismissals. However it is stated by the authors that the probability of false dismissals is very low. The first exact indexing mechanism was introduced in (Kim, Park, & Chu, 2001). The authors of this paper retain 4 features (e.g. Fist, Last, Min, Max values) of the original time series object and use them to create an index in main memory. The index is utilized to retrieve relevant objects that are of distance ε from a
specified query sequence. A post-processing step discards false positives by computing the actual DTW distance. As it was stated this method is exact meaning that it can ensure no false dismissals while pruning the search space from irrelevant objects. However, because of the small number of features utilized the number of false alarms is greatly increased significantly affecting the overall performance. Another important contribution was made in (Keogh & Ratanamahatana, 2005). An improved lower bounding technique was proposed, inspired by the constraints often imposed on DTW during calculation. The authors make use of the warping window to define two time series that enclose the initial time series object. The enclosing time series are defined as the maximum and minimum value respectively for consecutive values appearing in the warping window. Dimensionality reduction is then performed on the generated time series which are then used to prune the search space and find suitable matches based on DTW. This method is an improvement of the previous approaches as it ensures no false dismissals and provides a tighter bound than before thus improving the overall pruning performance of the index.

<table>
<thead>
<tr>
<th>Static Distance Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Lp-Norms</td>
</tr>
<tr>
<td>Elastic Distance Metrics</td>
</tr>
<tr>
<td>Dynamic Time Warping (DTW)</td>
</tr>
<tr>
<td>Edit Distance on Real Sequence (EDR)</td>
</tr>
<tr>
<td>Edit Distance with Real Penalty (ERP)</td>
</tr>
<tr>
<td>Pattern Distance Metrics</td>
</tr>
<tr>
<td>Spatial Assembly Distance (SpADe)</td>
</tr>
</tbody>
</table>

Figure 2 - Categories of Distance Metrics

DTW is classified as an elastic distance measure along with some other distance measures which are usually part of a subcategory in the same group, called edit distance based measures. Edit distance is a distance measure often used to partially match a pair of strings. It measures the minimum number of operations required to change one string into another. Applying an edit distance measure to time series data requires either representing the original object as a sequence of characters or perform some modification to the distance measure itself. Edit distance on Real Sequence (EDR) is a measure based on the latter approach. It was proposed in (Chen, Ozsu, & Oria, 2005) along with several pruning strategies to speed up search on large time series databases. The main idea behind EDR is relaxing the definition of equality by introducing a matching threshold, denoting the admissible maximum distance between two points in order to be considered equal. EDR is useful because it can tolerate outliers produce by wrong measurements, missing data or noise. It does that by having the ability to introduce gaps while considering the best candidates for matching subsequences of two comparing objects. However, EDR has a significant drawback as it does not obey the triangular inequality which makes pruning through indexing approximate by allowing false dismissals. An improvement of EDR is the Edit distance with Real Penalty (ERP) introduced in (Chen & Ng, 2004). Unlike EDR, ERP was designed to obey the triangular inequality thus making it suitable for indexing ensuring the no-false dismissal property. ERP evaluates gaps by measuring the distance of the non-gap element from a fixed point g. The $L_1$ norm is used to measure the effect of non-gaps on the overall measure. Although the latter requirement is similar to that of DTW, ERP is still different as it includes the notion of gaps into its definition. ERP in contrast to EDR, it does not require the definition of a matching threshold and is concerned with penalizing appropriately the introduction of a gap. As the underlying distance function (i.e $L_1$ norm) preserves the triangle inequality then the edit distance measure defined from it also does (Waterman, Smith, & Beyer, 1976).

A rather unique distance measure was presented in (Aßfalg, Kriegel, Kroger, Kunath, Pryakhin, & Renz, 2006). It is designed to answer efficiently the so called threshold queries. The purpose of these
types of queries is to retrieve time series objects that consist of similar threshold crossing intervals. These are intervals in which the observed value crosses a specific threshold $\tau$ defined by the user. The authors that proposed them argue that there are several applications dependent on these types of queries including fields of molecular biology, environmental analysis and in the pharmaceutical industry. Along with the definition of a distance measure for comparing time series with similar threshold crossing intervals, the authors present an indexing mechanism based on space decomposition to index efficiently time series objects based on arbitrary defined thresholds. They successfully perform k-Nearest Neighbor queries based on this distance measure given the provision of a query algorithm they describe in the same paper. Finally they compare the pruning power of their approach against sequential scan and found that it scales very well in terms of the number of objects in the database as well as the size of the time series considered.

Finally a pattern distance measure called Spatial Assembling Distance (SpADe) was proposed in (Chen, Nascimento, Ooi, & Tung, 2007). The goal of this distance measure was to allow matching of time series objects which are both shifted and scaled in time or amplitude respectively. It was also proposed as a distance metric that is tolerant of noise. SpADe detects segments of a time series object that are similar, using a sliding window to describe a local pattern (LP) based on certain features. These features are the position of the pattern in the time series object, the mean amplitude of the data in the LP, the shape signature and the temporal and amplitude scales of LP. The distance of two local patterns is measured using a weighted sum of the absolute differences for the amplitude and shape features. Although the weights can be defined by the user making them easily adjustable for data from diverse application domains, it can be also a disadvantage to have to fine tune the individual parameters each time. Similar to the existing metrics the authors present certain pruning techniques to speedup search and evaluate their method utilizing 1-Nearest Neighbor queries. The results of the experiments indicate that there is some importance on the way the size of the LP is defined. For most cases SpADe can achieve a lower classification error on dataset with smooth shapes. Results of different experiments indicate that SpADe can outperform previous methods (e.g ED, DTW, EDR) in terms of the matching accuracy under time and amplitude shifting, time and amplitude scaling while being more efficient in terms of computational complexity.

**QUERY TYPES AND QUERY LANGUAGES**

Time series data have been studied extensively in attempt to create abstractions that can be effectively used to enable efficient processing of diverse types of queries. Although the data structures are effective enough to handle different data and execution loads, there is an increasing need to create dedicated tools that are easy to use and have the ability to help users perform high level analytics on the underlying data. Successfully dealing with this issue requires the definition of a formal query language that will include a set of operators capable of complex analytics. An alternative approach would be to utilize a diverse set of distance measures that are capable of handling different types of queries. Although the former approach is generic and domain independent it has a major. It is difficult to create a generic solution given the diverse nature of queries that need to be answered. On the other hand the latter approach needs to retain information for every possible type of query and in many cases created distinct data structure for different distance functions. In the following sections both of these approaches will be discussed and analyzed in more depth.

**Types of Queries**

The distance measures defined in the previous section were used in many cases to answer 1 or k-Nearest Neighbor queries. These are types of queries which appear frequently in time series analysis. Some other important categories of queries which can be successfully applied on time series data include range queries, queries by humming(Ghias, Logan, Chamberlin, & Smith, 1995) and interval skyline...
queries. Range queries in the context of time series refer to operations which are used to retrieve all objects that are similar and within a range $\varepsilon$ from the query sequence. Queries by humming are relevant mostly to time series that represent some kind of sound signal. They are mostly used for matching a query tune to a database with recordings (i.e the time series data). As their name suggests their query tune is a sound produced from a person a scenario frequent in applications such as voice recognition and song matching. The challenge in these queries is induced by the potential difference in frequency and amplitude of the input and is often tackled with the use of DTW(Kim & Park, 2013), (Park C. H., 2014).

Interval skyline queries follow a similar idea to threshold crossing queries. Their main goal is to find a set of time series objects that are not being dominated by any other object in the database for a certain interval. A time series $s$ is said to dominate another series $q$ for an interval $[i:j]$ if and only if $\forall k \in [i:j], s[k] \geq q[k]$ and $\exists l \in [i:j], s[l] > q[l]$. Answering this type of query is computationally expensive and challenging given a very large database of time sequences. In (Jiang, 2009) a method was introduced for answering online skyline queries on time series data. This paper is motivated by several applications which find skyline queries necessary including finding webpages with the most hits for a given period of time or identifying users with the highest demand peak through consumption monitoring. Since the skyline computation is well known in the database community applying it naively for each observation is a potential solution. However, it is recognized by the authors that it infeasible to do so in an online manner. The proposed solution requires extracting the maximum and minimum values for fixed size intervals of each time series object and retaining them in a radix tree. During query time the radix tree is used to answer the skyline queries.

![Figure 3- Types of Queries](#)

**Query Languages**

A different approach of answering queries over time series data is based on the definition of dedicated query languages. This approach starts with the formal definition of a structured query language which is then utilized by users to provide a high level description of the matching content that needs to be retrieved. In contrast distance measures have the disadvantage of requiring an explicit description of the corresponding query at fine granularity. Query languages allow users to partially match a high level description of an object which is not fully defined in terms of shape but in terms of behavior as it evolves over time.

The first attempt on a proposal for a unified query language model on time series data was made in (Psaila & Wimmers Mohamed & It, 1995). The authors of this paper introduce the so called Shape Description Language (SDL) composed of an alphabet that describes dedicated shapes and a set of operators that work with the specified alphabet. The symbols of the alphabet are defined by the user and have 4 properties describing the value of the upper bound, lower bound, as well as the initial value and final value of the shape described. A time sequence can then be created by declaring a transition sequence composed of symbols of the defined alphabet. Individual shapes described from transition sequences can be stored and used with defined operators in a similar manner to that of regular expression matching on string objects. The authors also refer to the expressive power of SDL arguing that it can support blurry
matching, natural language to express complex queries and can be implemented efficiently. They support these arguments based on an analysis of the dedicated operators available in SDL and their significance in answering different types of queries. Also in terms of efficiency they present a storage structure which enables efficient pattern matching based on the defined alphabet. Extending SQL to accommodate sequences of data was another approach introduced in (Seshadri, Livny, & Ramakrishnan, 1995) with the so called SEQ model. The main advantage of this model is that it was based on existing tools and it could deal with different types of sequence data. However, the model is comparatively weak in terms of expressing pattern queries. It relies on the regular definition of data in relational database and defines a set of operators dedicated to handling data sequences. Efficient evaluation of queries is based on total ordering of records.

Figure 4 - Categories of Query Languages

An important contribution related to time series languages and queries in general is the TREPL language introduced in (Motakis & Zaniolo, 1997). It aims at supporting aggregation queries which are different from standard shape matching queries discussed until now. Their usage is motivated from the need of complex event detection over sequences of data. Some examples of aggregation queries as presented in the paper are counting aggregates, accumulation and running aggregates, moving window aggregates, temporal grouping aggregates and quantified temporal aggregates. Several of these operations are very important for time series analysis, prediction modeling and clustering of time series objects. In the paper describing TREPL, the authors try to distinguish themselves from standard Time Series Management Systems (TSMS) which assume a complete view of the data for processing. Instead they operate in a completely different domain which is based on active databases. The assumption here is that the reasoning on data is on-line and a response must be available immediately. Important differences between the domains are irregularity in the generation rate of observations/values/events that are monitored and the difference in response time and storage space needed to produce a decision. The language itself has been created as an extension to EPL (Giuffrida & Zaniolo, 1994), a composite event language for relational databases to detect simple events. TREPL works by defining events to be detected called rules. These rules can be used to define simple events and combined to create more complex ones. Some complex events that can be described include sequence, conjunction or disjunction of distinct events. Along with the definition of rules that denote specific events, the language provides a set of operators, mainly the star operator, to implement aggregation queries. TREPL allows users to define their own aggregation queries combining rules with the defined operators.

The SQL/LPP (Limited Patience Patterns)(Perng & Parker, 1999) language is another important addition in the family of time series languages. The premise supporting the development of such a language is motivated by the lack of previous solutions to provide shape matching capabilities over streaming data sequences. As an extension of SQL, SQL/LPP is described as an extension of SQL that can efficiently manage data sequences. It relies on incremental computation on the individual segments of
a sliding window using an attribute queue. Since pattern matching is often implemented utilizing a sliding window on the sequence under consideration, attribute queues can be used to incrementally compute the level of matching and emit an answer when the matching conditions are satisfied. The main building blocks of the SQL/LPP language are pattern functions which are used to define the pattern specifications. Initially some elementary pattern functions are defined by the authors but the overall goal is to allow users to extend the language with a definition of custom pattern functions. An important addition to SQL/LPP is the clause BY SEARCHING which allows the user to search the time series objects given a specific pattern. The BY SEARCHING clause can accommodate multiple patterns at a time which are treated as different streams and can be merged to one stream using the SYNC ON clause. The advantages of SQL/LPP is the fact that it uses already existing tools (e.g SQL) and is based on incremental computation which is realized looking at the sliding window process employed to find patterns in streaming data.

Some other extensions of SQL to treat streaming time series data include SQL-TS/ESL-TS(Sadri, 2001). (Bai, Luo, Thakkar, & Zaniolo, 2005) which provide additional clauses to define streams as well as the way they are processed using the PARTITION BY clause. These extensions also support aggregation queries which are either as continuous or final indicating the lifespan of the computation for an answer to be available. Similar to previously proposed solutions, users have the ability to define their own aggregates while extending them to deal with blocking or non-blocking computations. Non-blocking aggregates can be used only with streams of data as opposed to blocking which can both employed over a stream or database table. Finally, some other languages for streaming data include the Continuous Query Language (CQL) (Arasu, Babu, & Widom, 2006) and Trend Pattern Query Language (TPQL) (Imamura, Takayama, & Munaka, 2012). CQL is presented as a language similar to SQL having the ability to support continuous queries over streaming data. TPQL is based on CQL and is mainly used for anomaly detection through the use of newly defined convolution operator for identifying trend patterns.

TIME SERIES MANAGEMENT SYSTEMS

A combination of indexing techniques, representation methods and query languages for time series data motivated the design of holistic systems capable of handling time series data. Time Series Management Systems (TSMS), have to provide a diverse set of operations to the user with many of them being domain independent. However, these operations are inherently complex, computationally expensive and unique in nature to be supported by conventional database systems (i.e RDBMS). For this reason many attempts have been made in the past to formalize the design specifications and implement dedicated TSMSs.

Many of the approaches described can be incorporated into a formal database system. However it is not clear what types of conventional operation are needed since many scientific domains that utilize time series analysis techniques have different needs. A first attempt to formalize the requirements of a TSMS was made in (Dreyer, Dittrich, & Schmidt, 1994). The challenges related to creating a TSMS and the limitations of traditional database systems for time series data-centric applications were discussed in[?]. An analysis of the necessary capabilities of a TSMS is better understood when realizing the different requirements it must fulfill which are related to structural, functional, data exchange and synchronization requirements. Structural requirements refer to the way time series data are maintained and utilized. In many cases a continuous representation is useful allowing primitive operations which involve periodicity transformation (e.g weekly to daily values), derivation of new time series objects (e.g data flow measured from stock price closing values) or filling missing values (e.g interpolation). Also raw data can be grouped together as part of data analytics operations. Functional requirements refer to the supported operations such as queries, aggregation function or general types of analytics (e.g prediction, clustering, and classification). Many of the relevant analytics tools have been part of specialized packages indicating that there is no need of re-implementing them a TSMS. It makes sense for the developed system to play a supportive role providing preprocessing operations (e.g moving average, transformation
and streaming windows) before the final data analysis stage. The authors mention as a requirement the creation of an interactive interface for updating and analyzing data, which is a higher level of sophistication not particularly necessary for a data management system in this author’s opinion. Lastly, another important requirement is related to data synchronization. The inherit size of a time series objects as well as the rate at which new data are generated, does not only challenge their availability to the users but also complicates real-time decision making. The first problem is also recognized by the authors of the respective paper.

Sensor networks are the predominant application domain where time series data are generated in bulk. Inspired from this application, many researchers aim at creating systems which the ability to robustly process information in parallel from different sources. Efficiently processing streaming data without any loss of information and dealing with the continuous increase of the data size are challenges not only limited to the sensor network domain. In many cases time series data are generated from a diverse set of applications including resource monitoring in server farms, fraud detection through browsing history monitoring, user visitation of a webpage for advertisement purposes and similar operations observed from web data. There is a broad spectrum of systems proposed for managing time series data which aim to tackle the previously mentioned challenges. These include DataGarage (Loboz, Smyl, & Nath, 2010), TSDB (Deri, Mainardi, & Fusco, 2012), TSDS(Weigel, Lindholm, Wilson, & Faden, 2010), Dremel (Melnik, Shivakumar, Tolton, & Vassilakis, 2010), Vertica(Lamb, Fuller, Tran, Vandiver, Doshi, & Bear, 2012), Dataseries (Anderson, Arlitt, Morrey III, & Veitch, 2009) and Respawn (Buevich, Wright, Sargent, & Rowe, 2013). Some other known systems include OpenTSDB (Sigoure, 2012), InfluxDB and RRDTOOL (Oetiker, 2005) which will be mentioned for the completeness although not discussed in detail since there is no published work for them.

It has been established that conventional DBMS systems are not enough to handle the sheer volume of time series data. NoSQL systems and techniques have been introduced as a good alternative for managing data of extreme size. However, these systems usually lack the ability of performing complex analytics over the stored data. Time series is a special case of data which rely on both efficient storage and complex operations making it only natural for researchers to suggest solutions utilizing a hybrid model based on RDBMS and NoSQL systems. DataGarage is a hybrid system used for resource monitoring in server farms. It benefits from the organization of data in tables which are subsequently stored to files on top of a distributed file-system. This leverages the use of a complex querying interface available to SQL systems to the high throughput provided by parallel accessing of files using the MapReduce programming model. This hybrid architecture aims at storage efficiency, high query performance and a simple interface for query execution. It is based on the definition of wide-tables which are used to retain the corresponding time series data. The basic idea is to create a single table for each resource monitored by the system. As a consequence of this the number of wide-tables is increased making it more difficult to handle them efficiently. Storing them as files in a distributed file-system helps alleviate the access overhead. Complex queries are processed in parallel on local instances of an embedded SQL database and results are aggregated.

TSDB has been developed as an improvement of RRDTOOL. It organizes data per column in contrast to conventional DBMSs which organize data per row, a method that can enable fast updating operations. This design decision is based on the realization that the time it takes to consolidate data in the database restricts the minimum sampling frequency over the monitored sources. However, this organization makes it difficult for data to be efficiently retrieved. TSDB uses an indexing schema associating time series objects that are not stale with a unique identifier. The identifier can subsequently be used for efficient access of the whole object. Different compression techniques are utilized to reduce the data footprint in the database. TSDB aims at providing a framework that efficiently stores, retrieves and updates time series objects. However, it lacks a formal API where elementary operations on the data are available and can be extended by the user. TSDS tries to fill this gap by providing an API capable of operating over the stored data hiding implementation details of the underlying time series database. In the paper the authors recognize the need for operations such as filtering, data summarization, indexing and many other which are mainly motivated by different scientific disciplines (e.g financial analysis, weather
analysis). In general this framework is designed to provide simple operations useful in many cases for complex analytics or data visualization.

Dremel is introduced as a column oriented database capable of handling trillions of aggregation queries. It was developed at Google and works on top of the Google Filesystem (GFS). The main idea behind it is the organization of data into columns a technique also promoted by TSDB. It optimized for storing nested data in multiple columns which are then traversed efficiently through a series of indices connecting them. The nested sequences are organized using a multi-level serving tree which processes queries in a hierarchical way. The corresponding system is compared to record based map-reduce and column based mapreduce outperforming both of them in terms of execution time for aggregation queries.

Vertica is a similar system created on top of C-store (Stonebraker, et al., 2005) a column oriented datastore which is read rather than write optimized. Vertica considers two distinct workloads: transactional which are characterized by many queries per second on a handful of tuples and analytic characterized by small number of requests referring to large chunks of the underlying data. Time series analytics usually require fast access to the stored data for real time decision making. This is why Vertica is optimized for the second type of workloads. The data are modeled as tables consisting of distinct columns but are organized in projections (i.e sorted groups of attribute subsets) a technique utilized from C-store. An important feature of Vertica is the inclusion of a design tool called Database Designer (DBD) which automatically creates the optimal projections, based on a representative query workload taking into account the load overhead and the space requirements. Lastly, Dataseries is a publicly available time series database that differs from the previously described solutions in the way data is organized. Dataseries assumes a set of records which contain multiple attributes of different type and creates groups of records if they have the same attributes and attribute fields. These groups are called extents and their type is called extent type. The database is composed of Dataseries files which contain one or many extents. Along with their proposed data organization model, the authors present a programming model that enables analysis, update and conversion of data to alternate file forms operations.

Respawn is a time series datastore motivated by the existence of streaming data on sensors networks and the need for event detection at the same time. In particular Respawn is designed to deal with challenges emerging in sensor networks which include real time updating of time series as well as real time query support for clients. It leverages these operations using edge nodes which are responsible for gathering sensor readings from remote sensors and communicating updated data to a cloud node. The cloud node is responsible for handling requests from clients wishing to get information on the sensor readings. Data are managed from the Bodytrack Datastore (BTDS) which is a multi-resolution datastore for time series.

CONCLUSION

Bibliography


Camera, A. a., Rakthanmanon, T., & Keogh, E. (2014). Beyond one billion time series: indexing and mining very large time series collections with iSAX2+. *Knowledge and information systems, 39*(1), 123--151.


